Systém pro správu objednávek výrobní firmy

Bc. Jozef Kováč

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jozef Kováč**
Osobní číslo:	**A20132**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Systém pro správu objednávek výrobní firmy**
Téma práce anglicky:	**Order Management System for a Manufacturing Company**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj a zabezpečení webových aplikací.
2. Zaměřte se především na frameworky .NET Core a Blazor.
3. Zpracujte přehled architektonických a návrhových vzorů vhodných pro tvorbu dané webové aplikace.
4. Navrhněte aplikaci, definujte funkční a nefunkční požadavky, případy použití, modely tříd a drátové modely uživatelského rozhraní.
5. Navrhněte způsob zabezpečení komunikace mezi klientem a serverem.
6. Realizujte vývoj navržené aplikace a popište její klíčové části.
7. Demonstrujte výsledky a formulujte závěr.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. .NET documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-4]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/
2. ASP.NET Core Blazor documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-4]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor
3. Microsoft SQL documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-4]. Dostupné z: https://docs.microsoft.com/en-us/sql
4. PERES, Ricardo. Mastering ASP.NET Core 2.0: MVC patterns, configuration, routing, deployment, and more. Birmingham: Packt, 2017, xi, 471 s. ISBN 9781787283688.
5. PECINOVSKÝ, Rudolf. Návrhové vzory: [33 vzorových postupů pro objektové programování]. Brno: Computer Press, 2007, 527 s. ISBN 9788025115824.
6. FREEMAN, Adam. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 8th ed. 2020. Berkeley, CA: APress, 2020 ;, 1 online zdroj (XXIX, 1080 stran). ISBN 9781484254400.
7. FREEMAN, Adam. Pro Entity Framework Core 2 for ASP.NET Core MVC. [Berkeley, CA]: Apress, 2018, 1 online zdroj. ISBN 9781484234358.

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.5.2022 Jozef Kováč, v. r.

ABSTRAKT

Diplomová práca si kladie za úlohu navrhnúť a skonštruovať systém pokrývajúci základné úkony evidenčnej a administratívnej povahy v rámci firmy vykonávajúcej výrobnú činnosť. Teoretická časť práce skúma aktuálne dostupné technológie, nástroje a návrhové vzory, popisuje ich kľúčové charakteristiky a určuje vhodnosť týchto riešení pre tvorbu webového systému stredne veľkého rozsahu. Praktická časť práce je následne zameraná na stručnú analýzu, modeláciu a samotnú implementáciu konštruovaného riešenia formou internej webovej aplikácie založenej na architektúre Klient - Server a návrhovom vzore MVC za využitia frameworku Blazor.

Kľúčové slová: administratíva, framework Blazor, evidencia objednávok, Klient - Server, MVC, výroba, webové technológie, webový systém, zabezpečenie

ABSTRACT

The diploma thesis aims at designing and constructing a system to cover fundamental evidentiary and administrative tasks in manufacturing company environment. The theoretical part of thesis explores currently available technologies, tools and design patterns while describing their key traits and determining their suitability for developing a middle-sized web-based system. Practical part focuses on brief analysis, modeling, and implementation of proposed solution in the form of an internal web application based on Client - Server architecture and MVC design pattern using the Blazor framework.

Keywords: administration, Blazor framework, record of orders, Client - Server, MVC, manufacturing, web technologies, web system, security

Ďakujem vedúcemu práce Ing. et Ing. Erikovi Královi, Ph.D. za cenné návrhy, nápady, usmerňovanie a početné konzultácie v priebehu tvorby diplomovej práce – jeho ochota sa neoceniteľným spôsobom podpísala na kvalitatívnom aspekte jej finálnej podoby.

Ďalej veľká vďaka patrí všetkým priateľom a blízkym, ktorí ma počas štúdia podporovali a zdieľali so mnou šťastné aj ťažké chvíle.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Výpočtová technika už dlhý čas významným spôsobom ovplyvňuje smer napredovania ľudskej spoločnosti – a to vo všetkých predstaviteľných smeroch vrátane kultúry, zábavy, možností trávenia voľného času, ale aj mnohých praktických oblastiach sprostredkovaním jednoduchých a dostupných spôsobov komunikácie a plánovania. Jedno z odvetví ktoré bolo masívnym rozšírením informatizácie zasiahnuté najvýraznejšie je práve priemysel – nielen výroba samotná a jej evidencia, plánovanie či optimalizácie, ale najmä súvisiace úkony administratívnej či dokumentačnej povahy.

Bohužiaľ, mnohé výrobné firmy v dnešnej dobe stále nevyužívajú moderné možnosti informatizácie efektívnym spôsobom - a to ani v rámci integrálnych častí ako je napríklad interná firemná administratíva. Prostriedok, ktorý by mal riadenie firmy uľahčiť sa tak paradoxne nezriedka stáva nočnou morou vo forme enormného množstva historických dokumentov, tabuliek a nástrojov ktoré variabilným vekom, vzájomnou nekompatibilitou, rozporuplnosťou obsiahnutých dát či prostým neošetrením prípadnej ľudskej chyby prispievajú k celkovej nekonzistencii, chaotickosti a problematickosti hoci aj základných administratívnych úkonov.

Typickým príkladom užitia prítomným prakticky v každej výrobne zameranej firme je správa objednávok – evidencia objednávok a zákaziek pomáha nielen udržiavať prehľad o produktivite výroby a získať cenné štatistické dáta, ale taktiež umožňuje v rámci komunikácie so zákazníkmi sprostredkovaním dôkazového materiálu vo vhodnej podobe riešiť mnoho otázok právneho či obchodného charakteru ešte predtým, ako prerastú z nedorozumenia v incident negatívne ovplyvňujúci dobré obchodné vzťahy.

Profesionálne „krabicové“ softwarové riešenia na komplexné zastrešenie firemnej administratívy samozrejme existujú – a spolu s nimi existuje plejáda dôvodov, prečo ich firmy odmietajú používať. Medzi najčastejšie patrí príliš vysoká cena, prehnaná komplexnosť či nedostatočné pokrytie špecifických potrieb firmy daným software. Konatelia firiem menšieho až stredného rozsahu môžu nadobúdať pocit, že vynakladajú zdroje na funkcionalitu, ktorú vzhľadom na objem či povahu činnosti firmy nevyužijú. Potenciálnou alternatívou ku „krabicovým“ riešeniam môže byť tvorba firemného software na mieru. Tieto systémy možno neobsahujú široké spektrum pokročilých funkcionalít, ale zato sa vyznačujú vysokou mierou personalizácie obsahu pre potreby konkrétnej firmy a jednoduchou modifikovateľnosťou v rámci budúceho vývoja.

Diplomová práca si kladie za cieľ realizovať a popísať konštrukciu interného firemného systému pre správu objednávok a súvisiace administratívne úkony s využitím kombinácie osvedčených a moderných webových technológií takým spôsobom, aby bola dosiahnutá čo najvyššia miera funkčnosti a užívateľskej prívetivosti bez zbytočných kompromisov v oblasti výkonu či infraštruktúrnej kvality aplikácie.

Teoretická časť práce sa zameriava na prezentáciu súčasného stavu populárnych webových technológií – analýzou ich špecifík, silných a slabých stránok a pozície na trhu deskriptívnym spôsobom približuje čitateľovi charakteristické črty webového vývoja – od objasnenia pojmov a všeobecnej architektúry cez popis populárnych technológií a frameworkov pre tvorbu webových aplikácií až po zaužívané návrhové vzory či spôsoby zabezpečenia internetovej komunikácie – zvláštnu pozornosť pritom venuje frameworkom *.NET Core* a *Blazor*, ktoré sú nosnými technológiami implementovaného systému.

Praktická časť práce je orientovaná na návrh a zostrojenie webovej aplikácie pre správu objednávok v kontexte výrobne zameranej firmy. Analytická fáza zahŕňa niektoré z najčastejšie využívaných prostriedkov v procese návrhu software - napr. zber a analýza požiadaviek, zostrojenie diagramu prípadov použitia či konceptuálny návrh grafického užívateľského rozhrania. Spracovanie implementačnej fázy následne systematickým spôsobom popisuje samotné funkčné celky naprogramovanej aplikácie vrátane ich štruktúry, významu v celkovom kontexte systému a prípadného popisu charakteristík technického spracovania.

I. TEORETICKÁ ČASŤ

1 VÝVOJ WEBOVÝCH APLIKÁCIÍ

1.1 Problematika

Sprístupnenie Internetu verejnosti začalo novú éru informatizácie naprieč ktorou bolo konzistentne možné pozorovať strmý nárast počtu zariadení pripojených do siete – sprvu množstvo užívateľov internetu rástlo opatrne, s príchodom cenovo dostupnej a užívateľsky prívetivej infraštruktúry už exponenciálne [1]. Široké spektrum inteligentných zariadení, úspechy v implementácii myšlienky Internetu Vecí (*IoT – Internet of Things*) a prínosy nasadenia technológie blockchain do všemožných odvetví priemyslu nasvedčujú tomu, že tento trend bude pokračovať aj naďalej [2].

Masívny rozmach využitia online priestoru najmä v posledných dvoch dekádach [1] viedol k viacerým pridruženým efektom – s rapídnym nárastom prenosovej kapacity, ktorú boli poskytovatelia internetových služieb schopní poskytovať v kombinácii s neustálym rozvojom výkonu výpočtovej techniky napredovala aj snaha o tvorbu webových stránok ktoré budú esteticky krajšie, premyslenejšie, užívateľsky prívetivejšie a stále viac nabité funkcionalitami. Tento trend je možné pozorovať aj dnes – v rámci webového vývoja máme možnosť vnímať niektoré ustálené technológie, ktoré sú využívané už od jeho vzniku, kontinuálne rozširované a zdokonaľované tak, aby podporovali moderné potreby a trendy. Taktiež neustále vzniká široké spektrum nových a nádejných technologických riešení – programovacie jazyky, vývojové frameworky, rozšírenia, pluginy, knižnice či dokonca komplexné systémy schopné dynamickej tvorby webového obsahu každým dňom súperia o svoje miesto na trhu webového vývoja. Trh práce túto skutočnosť pochopiteľne reflektuje [3] – stále väčšie množstvo súkromných osôb či firiem sa pokúša dostať informáciu o svojej existencii a službách k potenciálnym klientom, alebo tieto služby prostredníctvom Internetu priamo poskytuje [viď. kapitola *1.3 – „Webové Stránky“*].

1.1.1 Využitie webových systémov

Využitie webových technológií v súčasnosti už nie je obmedzované tematikou či rozsahom vyvíjaného softwaru – ukázalo sa, že pri správnej implementácii mechanizmov zabezpečenia, voľbe vhodných nástrojov, architektúry a optimalizácií sú mocným nástrojom použiteľným k riešeniu prakticky ľubovoľného problému.

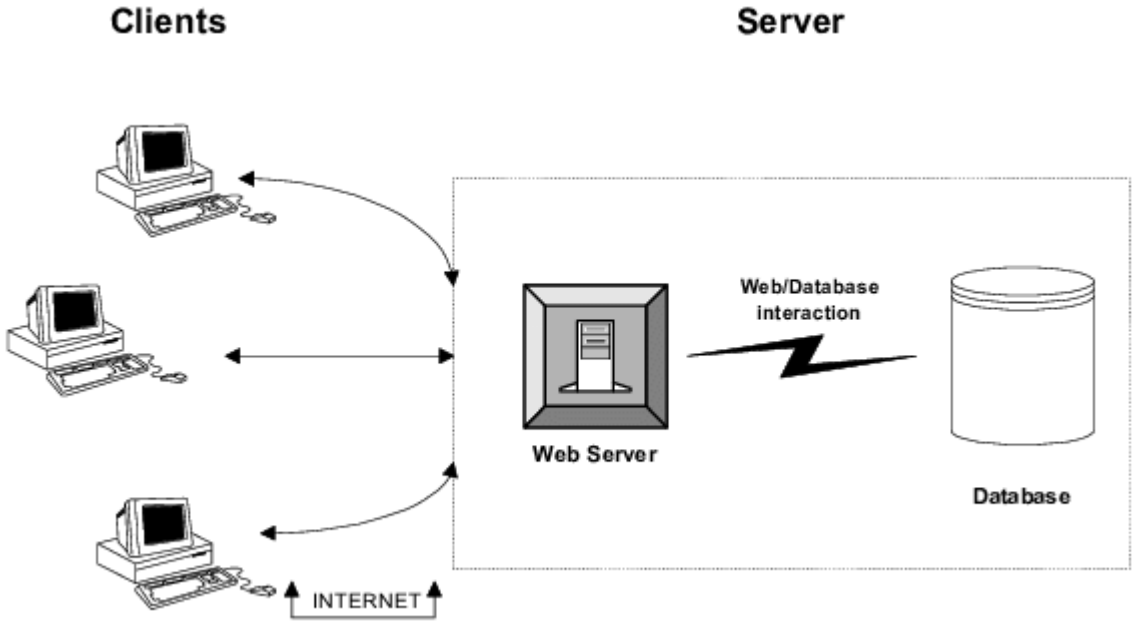
Sektory ľudskej činnosti, ktoré nejakým spôsobom adoptujú ľubovoľný typ softwarových riešení, sú prioritne orientované v oblastiach úzko súvisiacich s využitím výpočtovej techniky či sférach s nutnosťou digitalizácie a udržiavania vysoko špecializovanej vedomostnej bázy [3]. Dáta však ukazujú zaujímavú skutočnosť – rozšírenie softwarových riešení prebieha nielen v oblastiach a odvetviach prístupných prostredníctvom Internetu zvonku, ale aj zvnútra – je zrejmé, že optimalizácie výroby, systémy interného firemného manažmentu, evidencie či administratívy sú vítanými prostriedkami umožňujúcimi skvalitnenie a zjednodušenie výrobného procesu samotného, alebo s ním súvisiacich aktivít.

Téza vhodnosti implementácie internej firemnej administratívy formou webovej služby je naďalej podporovaná prirodzenou viacdielnou architektúrou webových služieb [viď. kapitola *1.2 – „Obecná štruktúra“*]. Táto platforma umožňuje svižný vývoj vďaka dostupnosti pestrej škály nástrojov a modulov, rozsiahlych možností miešania a spolupráce rôznorodých technológií prostredníctvom štandardizovaných spôsobov komunikácie [4] a sofistikované spôsoby zabezpečenia s vysokou mierou modifikovateľnosti.

1.2 Obecná štruktúra

Interakcia medzi jednotlivými uzlami (zariadeniami) v rámci ľubovoľnej webovej stránky – podobne ako akákoľvek iná sieťová prevádzka - musí prebiehať na základe určitých pevne daných mechanizmov, implementovať vhodné prenosové protokoly a podliehať štandardom platným pre dané odvetvie. Štruktúra webových aplikácií je špecifická v tom zmysle, že vo väčšine prípadov musí principiálne ísť o minimálne dva samostatné celky [5] – časť aplikácie dostupnú z webového prehliadača pristupujúceho k internej funkcionalite spôsobmi vopred definovanými kompozíciou užívateľského rozhrania a časť aplikácie, ktorá je schopná tieto požiadavky prijímať, spracovávať a formulovať ku nim odpoveď vo vhodnom formáte [5, 6]. U webových aplikácií je nutné zvážiť aj potenciálne nerovnakú násobnosť týchto častí – obslužný celok (Server) zvyčajne postačuje v jedinej inštancii. Obsluhovaných celkov (Klient) ale môže byť viacero – a v drvivej väčšine prípadov skutočne je (napr. viacero súčasne nakupujúcich zákazníkov jediného internetového obchodu).

Komunikácia v rámci webových služieb je z nízkoúrovňového pohľadu vlastne iba jedným z prípadov *IPC* (*Inter-Process Communication*) na báze komunikačnej štruktúry typu *Socket*, ktorá prebieha medzi procesom webového prehliadača klienta a procesom reprezentujúcim službu serveru, ktorá načúva na vopred stanovených portoch s cieľom zachytiť, obslúžiť a odpovedať na prichádzajúce žiadosti o spojenie [7].



Obrázok 1. Komunikačný model Klient - Server s využitím prenosu prostredníctvom Internetu [8]

Výskyt interakčného modelu Klient - Server síce nie je obmedzený výhradne na webové aplikácie, ale práve webové aplikácie sú pravdepodobne najznámejším prípadom jeho nasadenia. Jedná sa o distribuovanú aplikačnú štruktúru, v rámci ktorej sa klienti prostredníctvom vhodného komunikačného protokolu dopytujú serveru na určité zdroje (súbory, serializované dáta..) a server odpovedá. Hlavná myšlienka distribúcie zodpovedností spočíva v tom, že server a klient môžu (a nemusia) existovať na výrazne rozdielnych zariadeniach, čo ale nemá vplyv na komunikáciu samotnú, keďže táto je zabezpečovaná s využitím unifikovaného rozhrania [6, 7]. Komunikácia nezriedka prebieha asynchrónnou formou – t.j. klient nerealizuje aktívne čakanie na odpoveď serveru, ale spracovanie odpovede môže v momente jej príchodu zabezpečovať k tomu určená funkcia - tzv. *callback handler*.

Niektoré zdroje [9, 10] naviac rozlišujú architektonické varianty *Client - Host* a *Server - Host*. Rozdiel spočíva v tom, že pri klasickej notácii Klient - Server sa pomenovanie vzťahuje k programom bežiacim na ľubovoľných zariadeniach (vrátane prípadu spustenia na jednom a tom istom stroji). Notácia *Client - Host* a *Server - Host* sa neodkazuje na programy / služby plniace rolu klienta resp. serveru, ale adresuje samotné fyzické prístroje vzhľadom k ich úlohám v danej interakcii. V ďalších častiach práce bude v súlade s pojmami architektúry Klient - Server referovaná notácia klasická, t.j. s využitím koncepcie dvoch komunikujúcich služieb, bez uváženia povahy a vzájomného vzťahu zúčastnených zariadení.

Vhodnosť komunikačného modelu Klient - Server je narušovaná snáď len niekoľkými nedokonalosťami bezpečnostnej povahy – prevažne sa však jedná o komplikácie buď odstrániteľné, alebo principiálne neriešiteľné [viď. sekcia *„Nevýhody modelu Klient - Server“* nižšie].

[11] Výhody modelu Klient – Server :

- **Centralizácia dát:**
  Spracovanie požiadaviek netrpí zbytočnou latenciou vplyvom možnosti centralizácie dát v rámci serveru (všetky dáta na rovnakom mieste). Tento prístup naviac pomáha zabezpečiť ochranu dát a správu autorizačných / autentifikačných mechanizmov.
- **Dobrá škálovateľnosť**:
  Časti infraštruktúry ako napr. dodatočné servery, počítače či sieťové prvky môžu byť pridané, vylepšené, vymenené alebo premiestnené bez výrazného narušenia doterajšej činnosti. Jednotlivé uzly sú nezávislé a obecne k dátam pristupuje výhradne server.
- **Variabilná vzdialenosť:**
  Možnosť prístupu k dátam bez nutnosti fyzickej blízkosti serveru a klientov.
- **Implementačná jednoduchosť:**
  Možnosť údržby / vývoja funkčných celkov zodpovedajúcich serveru a klientovi nezávisle na sebe.
- **Zabezpečenie:**
  Rozsiahle možnosti zabezpečenia, mnohokrát schopnosť komunikáciu aj bez nutnosti dôverovať dátam zasielaných klientom.

- **Jednotnosť komunikácie:**
  Transfer dát je sprostredkovaný prenosovými protokolmi, ktoré sú agnostické ohľadom konkrétnych platforiem prepojovaných zariadení.

[12] Nevýhody modelu Klient - Server:

- **Bezpečnostná rizikovosť serveru:**
  Pokiaľ je server úspešne napadnutý, stáva sa perfektným distribútorom malwaru schopného šírenia vírov, trójskych koní či červov ako súčastí bežnej komunikácie s klientmi.
- **Možnosť napadnutia serveru:**
  servery viditeľné zo siete Internet sú náchylné k útokom typu *DoS / DDoS (Denial of Service / Distributed Denial of Service).*
- **Odpočúvanie komunikácie:**
  Existuje riziko odpočúvania komunikácie medzi serverom a klientom (napr. formou monitorovania prenosu dátových paketov v sieti).
- **Napadnutie komunikácie:**
  Prenos je kvôli nutnosti prenosu po sieti (a súvisiacou nutnosťou nadviazania počiatočného spojenia) náchylný na útoky typu *Man In The Middle.*

Implementačná štruktúra a celková podoba programovej zložky klientov a serveru sa môže výrazne líšiť – využívajú sa rôzne technológie, programovacie jazyky, návrhové a architektonické vzory, rôzne prístupy k interpretácii, perzistencii a spracovaniu dát.

1.2.1 Klient

Klient je v architektúre Klient - Server vnímaný ako žiadateľ o informácie alebo zdroje držané a poskytované serverom [13]. Klientom je vlastne počítačový software (alebo hardware, v závislosti na notácii a kontexte). Kontext webových technológií zvyčajne ako klienta vníma buď webový prehliadač, alebo zariadenie z ktorého sa užívateľ snaží pristupovať k funkcionalite systému, najčastejšie prostredníctvom zobrazeného *GUI (Graphical User Interface)*.

[14] Rozlišujeme rôzne typy klientov:

- **Hrubý klient:**

  Výpočtovo intenzívny variant klienta, server plní iba konzultačnú úlohu. Väčšina dátového spracovania prebieha priamo na zariadení / v prehliadači klienta. Umožňuje znížiť záťaž serveru, ale hrozí nekonzistentný užívateľský zážitok kvôli možnej výkonnostnej variabilite hardwaru zariadení klientov a rozdielmi medzi webovými prehliadačmi.

- **Tenký klient:**

  Tenký klient je opakom hrubého klienta. Výrazne sa spolieha na výpočtovú silu serveru, ktorému prenecháva všetky náročné úlohy spracovania dát. Na strane klienta sú vykonávané výhradne jednoduché operácie - ako je napr. odosielanie požiadaviek serveru či vizualizácia odpovedí. Záťaž odňatá zo zariadení na ktorých bežia klientské časti aplikácie je presunutá na server – z toho dôvodu je nutné zabezpečiť, aby mal server dostatočný výkon pre poskytovanie bezproblémového behu služby.

- **Hybridný klient:**

  Hybridný klient je kombináciou tenkého a hrubého klienta – spolieha sa na server najmä v otázkach konzistencie a perzistencie spracovávaných dát, je ale schopný rozsiahleho lokálneho spracovania a výpočtových operácií na strane klienta.

1.2.2 Server

Server je v zmysle architektúry Klient - Server vnímaný ako poskytovateľ informácií, dát či zdrojov, na ktoré sa klienti spytujú. Podobne ako u klienta sa jedná o počítačový software (alebo hardware, v závislosti na notácii a kontexte) – v súvise s využitím v rámci webových technológií ako server označujeme buď službu ktorá prijíma a obsluhuje požiadavky prichádzajúce formou sieťovej komunikácie - alebo alternatívne fyzické zariadenie, na ktorom je táto služba aktívna [15].

[16] Rozlišujeme rôzne typy serverov:

- **Aplikačný server:**

  Aplikačný server hostí webové aplikácie v rámci siete a zabezpečuje, že užívatelia nemusia disponovať vlastnou kópiou aplikácie na to, aby mali prístup k jej funkcionalite. Spadajú sem napr. služby typu *SaaS (Software as a Service)*.

- **Výpočtový server:**
  Server úzko previazaný s myšlienkou konceptu nazývaného *Cloud Computing*, t.j. sieťového prístupu k enormnej výpočtovej kapacite výkonných strojov, ktorá nie je bežne dostupná v štandardných PC.
- **Databázový server:**
  Databázový server zodpovedá za údržbu, trvanlivosť, organizáciu a zdieľanie databázových údajov pre klientské programy, ktoré s nimi pracujú.
- **Webový server:**
  Umožňuje prevádzku dynamických webových stránok a tým prakticky existenciu súčasného Internetu tak, ako ho poznáme. Kombinuje črty a prístupy vyššie uvedených typov serverov v závislosti na potrebách a povahe služieb, ktoré prostredníctvom webu poskytuje.

1.3 Webové stránky

Webové stránky sú základným stavebným kameňom súčasného užívateľsky prístupného obsahu v rámci siete Internet. Pre konzistenciu pojmov je najskôr potrebné definovať, čo je pojmom „webová stránka“ vlastne myslené. V angličtine sa rozlišujú dva lexikologicky podobné, ale funkčne výrazne odlišné názvy – *Web Page* a *Website* [17, 18].

- **Web Page:**
  Štruktúrovaný dokument, ktorý môže byť zobrazený prostredníctvom webového prehliadača ako je napr. *Google Chrome*, *Safari*... Často je používaný skrátený názov „stránka“.
- **Website:**
  Kolekcia webových stránok, ktoré sú spolu štrukturálne či logicky previazané. V prípade systému schopného dynamickej tvorby obsahu zahŕňa aj dynamicky vytvárané stránky.

Nasledujúce odstavce nahliadajú na webový systém ako celok – t.j. slovným spojením „dynamické stránky“ nie je myslená iba jedna stránka ako dokument, ale všetok súvisiaci obsah vrátane vytvárajúcich mechanizmov (*Website*). Webové stránky môžeme deliť do dvoch základných kategórií – na statické a dynamické [18].

1.3.1 Statické

Pôvodný typ webových stránok, existoval už od prvopočiatku technológie *WWW (World Wide Web)*. V súčasnosti ním rozumieme stránky pozostávajúce výhradne z dokumentov typu *.HTML* a *.CSS*, ďalej multimediálnych súborov a potenciálne skriptov implementovaných v programovacích jazykoch spustiteľných z prostredia webového prehliadača (napr. *JavaScript*). Statické stránky neumožňujú perzistentné modifikácie zobrazovaných dát za behu a táto funkcionalita nie je prístupná ani užívateľovi. Zakladajú sa výhradne na prenose a prezentácii vyššie spomenutých nemenných dokumentov z perzistentného úložiska serveru do webového prehliadača klienta [19].

Statickosť stránky však neznamená absolútnu absenciu akejkoľvek interakcie s užívateľom – za statickú stránku je možné považovať napr. program interaktívnej kalkulačky spracúvaný výhradne v prehliadači užívateľa.

1.3.2 Dynamické

Dynamické webstránky pozostávajú z dokumentov ukladaných vo formáte, ktorý nie je priamo vykresliteľný webovým prehliadačom. Tieto dokumenty sú používané ako šablóny, prostredníctvom ktorých server so znalosťou perzistentných dát (v závislosti na požiadavke vytvorenej klientom) formuje odpoveď, ktorá už webovým prehliadačom zobraziteľná je. Dynamické webstránky sú teda webstránky implementujúce mechanizmy, ktoré umožňujú skladať stránky s variabilným obsahom v závislosti na určitom kontexte [5].

Proces zostavenia prebieha na serveri s využitím vhodných programovacích jazykov (napr. *PHP*, *Node.js*, *ASP.NET*...) a nástrojov, ktoré tento proces uľahčujú (napr. rôzne šablónovacie systémy - *Razor*, *Blade*…) [5, 20].

Dynamické webové stránky sprístupňujú nové možnosti interakcie užívateľa s obsahom stránky a stavom systému ako takého. Užívateľ vďaka nim môže svojou činnosťou zasahovať do stavu uložených dát a prenesene teda aj do obsahu zobrazovaného ostatným užívateľom. Dynamickými stránkami sú (vyplývajúc zo svojej povahy) napríklad všetky internetové obchody, rezervačné systémy či diskusné fóra.

1.4 Komunikačné protokoly

Prenos informácie prostredníctvom sieťovej komunikácie tak, aby nedochádzalo k prílišným pochybeniam či nekonzistenciám obsahu prenosu medzi jednotlivými účastníkmi sa musí riadiť určitými pravidlami – súbory týchto pravidiel definujúce špecifiká transferu určitého typu obsahu medzi dvoma alebo viacerými entitami či komunikačnými subsystémami nazývame komunikačnými protokolmi. Protokoly definujú pravidlá, syntax, sémantiku a mechanizmy pre synchronizáciu komunikácie či zotavenie z potenciálnych chýb v priebehu prenosu [21].

Definícia, vývoj a údržba komunikačných protokolov je - podobne ako u mnohých ďalších technológií v rámci Internetu, ktoré vyžadujú štandardizovaný prístup a exaktnú definíciu v teoretickej rovine – zastrešovaná organizáciou *IETF* (*Internet Engineering Task Force*).

Webové aplikácie aplikujú prenos prostredníctvom veľkého množstva komunikačných protokolov rôznych určení – niektoré sú vhodné na prenos dát, súborov, multimediálneho obsahu, implementáciu zabezpečovacích mechanizmov a pod. Použitie týchto protokolov je mnohokrát ukryté pod primitívami či abstrakciami poskytovanými frameworkom alebo programovacím jazykom, v ktorom bol daný systém vytvorený [22].

Najčastejšie využívanými protokolmi v rámci komunikácie v sieti Internet sú protokoly *HTTP* a *HTTPS*.

1.4.1 HTTP

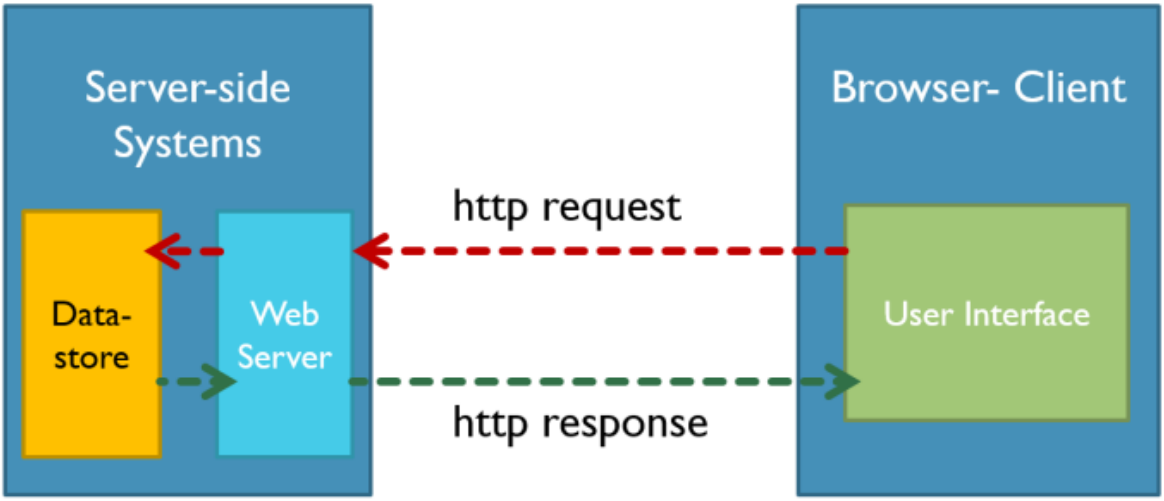
HTTP (Hyper Text Transfer Protocol) Prenosový protokol existujúci už od doby, kedy sa využitie Internetu začalo prenášať do verejnej / komerčnej sféry. Vyvinul ho Tim Berners-Lee v *CERN*-e v rokoch 1989-1991 a až do súčasnosti je základným komunikačným protokolom v rámci prostredia *WWW*.

[23] Rozoznávame štyri hlavné verzie HTTP protokolu:

- **HTTP / 0.9:**
    - Pôvodná verzia *HTTP* protokolu.
    - Jednoriadková povaha požiadaviek (*HTTP* metóda + cesta k dokumentu).
    - Podporovaná metóda iba *GET*.
    - Podporovaná odpoveď iba hypertext.
    - Spojenie ukončené okamžite po odoslaní odpovede.
    - Bez hlavičiek, bez stavových / chybových kódov.

- **HTTP / 1.0:**
  - Podporuje spoluprácu s internetovým prehliadačom.
  - Uvádza polia hlavičiek umožňujúce prenos metadát ohľadom požiadavky či odpovede (typ obsahu, stavový kód, *HTTP* verzia...).
  - Odpoveď serveru schopná prenášať súbory iné ako čisté *HTML* – napr. skripty, súbory kaskádových štýlov či médiá.
  - Podporované metódy: *GET, HEAD, POST.*
- **HTTP / 1.1:**
  - Aktuálne najčastejšie využívaná verzia.
  - Množstvo optimalizácií a vylepšení, ako napr. perzistentné spojenia, kompresia / dekompresia dát či využitie cache.
  - Podporované metódy: *GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS.*
- **HTTP / 2.0:**
  - Vynútenie zabezpečeného spojenia (webové prehliadače podporujú *HTTP* / 2.0 iba cez zabezpečené spojenie).
  - Server môže preventívne odosielať zdroje, ktorých využitie je v blízkej budúcnosti pravdepodobné do cache klienta ešte predtým, ako sú vyžiadané – redukuje zbytočné požiadavky.
  - Podporuje binárne protokoly umožňujúce menšie zahltenie prenosovej kapacity, lepšiu rýchlosť dátového spracovania a kvalitnejšie ošetrenie špeciálnych znakov v rámci prenášaného obsahu.
  - Podporuje multiplexing – klient má možnosť spustiť viacero požiadaviek súbežne cez jediné TCP spojenie bez nutnosti čakať na odpoveď z predchádzajúcich volaní.

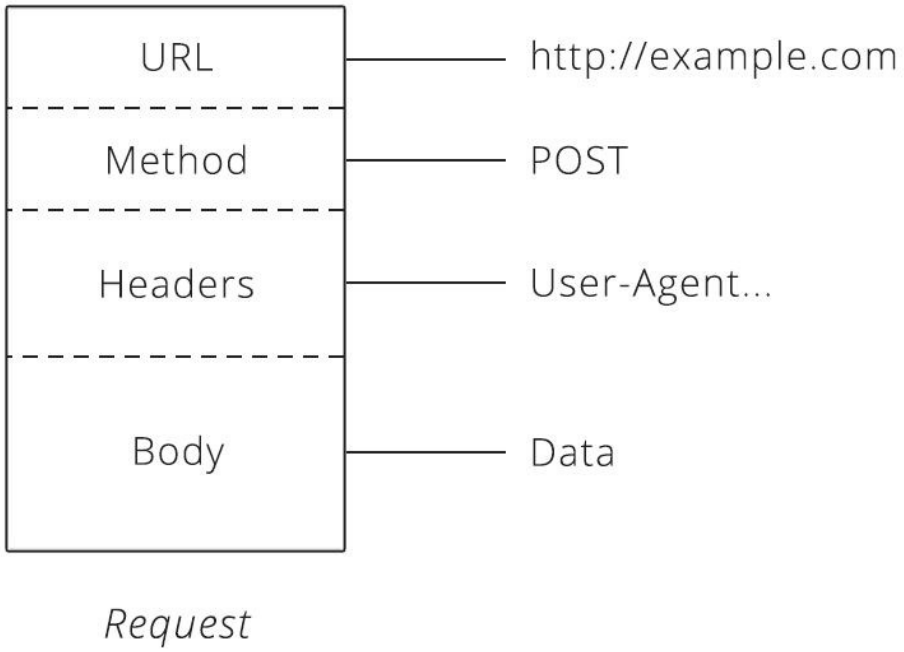
HTTP protokol je komunikačným protokolom založeným na modeli *Request - Response*, a teda výmene požiadaviek a odpovedí v dohodnutom formáte medzi klientom a serverom.



Obrázok 2. Princíp HTTP spojenia medzi klientom a serverom založeného na výmene požiadaviek a odpovedí [24]

1.4.1.1 Požiadavka

Požiadavka je abstraktným konštruktom *HTTP* protokolu (často sa v IT žargóne používa anglický názov *Request*), ktorý je odosielaný od klienta k serveru. Cieľom požiadavky je buď pristúpiť k nejakému zdroju držanému týmto serverom, spýtať sa na informáciu (dáta) potrebné k ďalšej aktivite klienta alebo prinútiť server vykonať nejakú činnosť či postupnosť činností [25, 26, 27].



Obrázok 3. Základná štruktúra HTTP požiadavky [28]

***HTTP* požiadavka je určená niekoľkými informáciami:**

- **URL:**

  URL (Uniform Resource Locator) je unikátny identifikátor udržiavajúci informáciu nutnú k prístupu na požadovaný zdroj (v kontexte populárnej architektúry webových služieb *MVC* [viď. kapitola *3.1 - „Model – View - Controller"*] sa jedná o cestu (*Route*) k metóde komponenty nazývanej *Controller*. Cesta obsahuje prípadné argumenty, ktoré táto metóda vyžaduje (v prípade systému pre spracovanie objednávok môže byť takýmto parametrom napr. ID modifikovanej objednávky).

- **HTTP Metóda:**

  Metóda podporovaná *HTTP* štandardom. Jedná sa o vlastne o jednoslovné upresnenie určené pre konzumáciu serverom, ktoré by správne malo určovať akým spôsobom si klient s daným zdrojom praje manipulovať.

  [29, 30] Najčastejšie využívané metódy sú:

  - **GET:**

    GET metóda si od serveru vyžiada zdroj alebo dáta špecifikované prostredníctvom *URL* požiadavky. *GET* metódy by v žiadnom prípade nemali umožňovať modifikáciu zdrojov na serveri (toto správanie môže byť cielene zneužiteľné malwarom či omylom spustené rôznymi automatizovanými nástrojmi určenými napr. k indexácii obsahu na Internete). Zvyčajne neobsahuje telo.

  - **HEAD:**

    HEAD metóda poskytuje identickú odpoveď ako ekvivalentná *GET* požiadavka, ale nevracia telo odpovede (v prípade, že je žiadúce získať iba metadáta ako napr. stavový kód bez nutnosti práce so samotným navráteným obsahom). Zvyčajne neobsahuje telo.

  - **POST:**

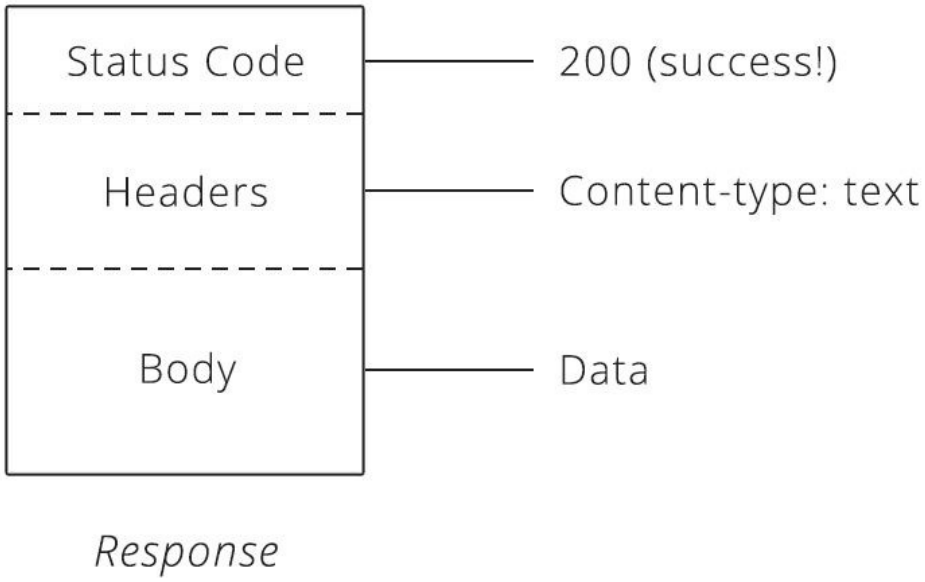
    POST metóda posiela vo svojom tele na server dáta reprezentujúce určitú entitu - v *OOP (Object – Oriented Programming)* sa môže jednať napr. o vhodným spôsobom serializovaný objekt, štruktúru či pole. Primárnym zámerom metódy *POST* je zvyčajne rozsiahla zmena stavu dát na serveri či spustenie určitej akcie – napríklad uloženia.

- **PUT:**
  PUT metóda je určená ako prostriedok rozsiahleho nahradenia už existujúceho cieľového zdroja novou reprezentáciou obsiahnutou v tele odosielanej *PUT* požiadavky.
- **PATCH:**
  PATCH metóda je podobná metóde *PUT*, ale správne by mala byť používaná k menším, parciálnym modifikáciám cieľového zdroja. Tieto zmeny sú obsiahnuté v tele odosielanej *PUT* požiadavky.
- **DELETE:**
  DELETE metóda slúži k zmazaniu špecifikovaného zdroja. Zdroj zvykne byť určený prostredníctvom *URL* alebo hlavičky *DELETE* požiadavky. Zvyčajne neobsahuje telo.

- **Hlavičky:**
  Požiadavka môže špecifikovať variabilné množstvo špeciálnych polí, tzv. hlavičiek (*Headers*). Hlavičky prenášajú cenné metadáta potrebné k bezproblémovému spracovaniu požiadavky – typickým obsahom hlavičky je napr. typ dát prenášaných v tele požiadavky [31, 32], alebo dodatočné príznaky využiteľné napr. pri obmedzovaní neautorizovaného prístupu (*API* kľúče, tokeny).
- **Telo:**
  Telo správy (*Body*) prenáša obsah správy a využíva sa najmä pri metódach, ktoré nejakým spôsobom požadujú po serveri zásah do perzistentných dát aplikácie (*POST, PUT, PATCH*). Obsah tela zvykne obsahovať vhodnú reprezentáciu dát, ku ktorým sa požiadavka vzťahuje [31, 32] – t.j. pri snahe uložiť objednávku budú obsahom tela dáta objednávky serializované do vhodného formátu (*JSON, XML*). Formát obsahu tela správy je možné rozlíšiť napr. nastavením hlavičky s kľúčom *Content-Type*.

1.4.1.2 Odpoveď

Odpoveď (*Response*) je reakciou serveru na požiadavku klienta a nesie informáciu o jej dokončení / nedokončení, metadáta špecifikujúce detaily priebehu spracovania a prípadnú informáciu o novom stave modifikovaných zdrojov, resp. o chybách v priebehu výkonu operácie. Odpoveď klientovi poskytuje spätnú väzbu jeho akcií [32, 33], na ktorú môže v prípade potreby reagovať. Štruktúra *HTTP* odpovede je značne podobná štruktúre *HTTP* požiadavky.



Obrázok 4. Základná štruktúra HTTP odpovede [29]

HTTP odpoveď je tvorená niekoľkými časťami:

- **Stavový riadok:**

  Stavový riadok (*Status Line*) je kombináciou stavového kódu (*Status Code*) a textového odôvodnenia odpovede v ľudsky čitateľnej forme (*Reason Phrase*), umožňuje jednoduchú identifikáciu akcie, ktorá bola serverom vykonaná (príp. dôvodu, prečo vykonaná nebola).

  [34] Stavové kódy sa kategorizujú do viacerých skupín:

  - **100-199**:

    Kódy riadiace komunikáciu medzi klientom a serverom.

    - 100 – *Continue*
    - 101 – *Switching Protocols*

  - **200-299:**

    Kódy signalizujúce úspešné vykonanie požiadavky.

    - 200 – *OK*
    - 204 – *No Content*

  - **300-399:**

    Kódy značiace, že na požadovanú *URL* nie je možné pristúpiť kvôli presunu súvisiacich zdrojov.

    - 301 – *Moved Permanently*
    - 307 – *Temporary Redirect*

- **400-499:**

  Kódy indikujúce chybu vplyvom klienta (nesprávna požiadavka)

  - 400 – *Bad Request*
  - 404 – *Not Found*

- **500+:**

  Kódy značiace neschopnosť serveru vykonať požiadavku alebo chybu v priebehu spracovania

  - 500 – *Internal Server Error*

- **Hlavičky:**

  Odpoveď, podobne ako požiadavka, môže špecifikovať viacero hlavičkových polí. Význam hlavičiek v odpovedi je identický s hlavičkovými políčkami v kontexte požiadavky [viď. kapitola *1.4.1.1* - „*Požiadavka*“]. Hlavičky držia metadáta potenciálne nápomocné pre voľbu vhodnej formy spracovania odpovede klientom.

- **Telo**

  Telo správy (*Body*), podobne ako u požiadavky, prenáša dátový obsah odpovede. Využíva sa ako prostriedok určený k doprave väčšieho objemu dát od serveru ku klientovi. V rámci *HTTP* odpovede sa telo používa takmer vždy (s výnimkou špeciálnych prípadov ako je napr. reakcia na *HEAD* požiadavku). Obsah tela – pokiaľ bol odoslaný v očakávanom formáte – je prakticky ľubovoľný. Môže obsahovať odôvodnenie neúspešnej akcie, v rámci *OOP* serializované dáta upravenej entity či dokonca dáta multimediálneho súboru.

1.4.2 HTTPS

Protokol *HTTPS* (*Hyper Text Transfer Protocol Secure)* je mutáciou *HTTP* protokolu, ktorá adresuje jednu z jeho základných bezpečnostných slabín – nešifrovanú formu komunikácie. *HTTP* prevádzka je z tohto dôvodu extrémne jednoducho napadnuteľná útočníkom načúvajúcim na sieti ktorou dátové pakety prechádzajú. Zabezpečenie v *HTTPS* je obojsmerné a zostavené prostredníctvom protokolu *SSL / TLS* [35, 36] [viď. kapitola *1.4.2.1 – “SSL / TLS”*].

Tabuľka 1. Porovnanie vlastností protokolov HTTP a HTTPS [37]

	HTTP	**HTTPS**
Zabezpečenie	Bez zabezpečenia	*SSL / TLS* digitálny certifikát vydaný nezávislou autoritou za účelom zabezpečenia komunikácie medzi klientom a serverom
Vrstva OSI / ISO	Aplikačná vrstva	Transportná vrstva
Port	Port č. 80	Port č. 443
Formát prenášaných dát	Nešifrované dáta (*plaintext*)	Šifrované dáta
Rýchlosť komunikácie	Rýchla komunikácia bez réžie naviac	Pomalšia komunikácia kvôli dodatočnej výpočtovej záťaži spôsobenej šifrovaním odosielaných dát
Rýchlosť spojenia	Rýchle naviazanie spojenia	Pomalšie naviazanie spojenia kvôli nutnosti vykonania *handshake* procesu [viď. kapitola *1.4.2.1 – „SSL / TLS“*]

1.4.2.1 SSL / TLS

SSL / TLS (Secure Socket Layer / Transport Layer Security) protokoly sú zvyčajne integrované do sieťovej prevádzky za účelom ochrany dát posielaných medzi klientom a serverom. Bezpečnosť a dôveryhodnosti sú zaručované digitálnym certifikátom, ktorý je pred začiatkom samotnej komunikácie odosielaný klientovi. Server aj klient následne na základe obsahu certifikátu vytvárajú tzv. *Session Keys*, teda symetrické kľúče určené k šifrovaniu prenášaných dát v rámci jedného neprerušeného komunikačného bloku (*Session*) [35, 36, 38].

Tento počiatočný kontakt medzi klientom a serverom na spôsobe zabezpečenia komunikácie, ktorý končí úspešným získaním všetkých prostriedkov pre realizáciu samotného šifrovania, sa nazýva *Handshake*.

[39] Typy bežne vydávaných *SSL / TLS* certifikátov:

- ***DV (Domain Validated):***
  Najnižšia úroveň validácie – verifikuje, že žiadateľ o certifikát je skutočne držiteľom domény, ktorej má byť tento certifikát určený. Overenie prebieha typicky na základe doménového názvu alebo umiestnením verifikačného súboru na ochraňované webové stránky.
- ***OV (Organization Validated):***
  OV certifikáty sú využívané korporáciami, vládami a inými entitami, ktoré chcú poskytnúť dodatočnú vrstvu dôvery svojim návštevníkom. Mimo využitie v rámci *SSL / TLS* zabezpečenia môžu byť použité aj pre podpisovanie kódu a dokumentov.
- ***IV (Individual Validated):***
  IV certifikáty slúžia k rovnakému účelu ako certifikáty typu OV [viď. sekcia *„OV (Organization Validated)*“], ale sú určené pre jednotlivcov – zvyčajne vývojárov, za účelom podpisovania kódu.
- ***EV (Extended Validation):***
  Certifikáty rozšírenej validácie poskytujú maximálnu mieru dôvery návštevníkom a zároveň sú najnáročnejšie získateľným typom certifikátu. Podmienky pre získanie *EV* certifikátu sú nastavené mimoriadne striktne a vyžadujú poskytnutie dodatočnej dokumentácie o subjekte, ktorému má byť certifikát vydaný. Certifikáty typu *EV* smú byť vydané iba firmám a iným registrovaným organizáciám, nie jednotlivcom.

2 TECHNOLÓGIE

Podobne ako v ľubovoľnej oblasti softwarového vývoja, aj k webovému prostrediu sa viaže mnoho rôznych technológií a nástrojov určených k návrhu, vývoju a optimalizácii s cieľom konštrukcie rýchlych, spoľahlivých a užívateľsky prívetivých systémov.

Kapitoly vyššie už načrtli, že aplikácie využívajúce Internet ako propagačné a komunikačné médium dodržiavajú z nutnosti rozdelenia aplikačnej logiky na časť klienta a časť serveru špecifický typ architektúry – samotná aplikácia je tak už pri implementácii rozdelená na dva logické celky, ktoré sa výrazne odlišujú celkovou infraštruktúrou, použitými návrhovými vzormi, prostredím behu a v neposlednom rade technológiami, programovacími jazykmi a frameworkami ktoré sú vhodné k ich implementácii. Tieto dva logické celky nazývame *Frontend* a *Backend*.

2.1 Frontend

Frontend je časťou aplikácie, ktorá umožňuje užívateľovi interagovať s aplikáciou a spravidla sa jedná o časť kódu, za ktorej spracúvanie je zodpovedný webový prehliadač konzumenta aplikácie. *Frontend* zodpovedá za zobrazenie obsahu v podobe webových stránok [40]. Dopad kvalitného (resp. nekvalitného) spracovania *Frontend*-u na celkovú funkcionalitu a užívateľský zážitok z prehliadania býva pritom mnohokrát podceňovaný – je nutné si uvedomiť, že sa jedná o prvý (a potenciálne posledný) styčný bod užívateľa s aplikáciou. Zabezpečiť, že *Frontend* bude intuitívny, responzívny, štrukturálne zmysluplný a vizuálne atraktívny je výzvou v každom projekte webového vývoja.

Pri vývoji *Frontend*-u je potrebné venovať pozornosť trom hlavným logickým celkom:

- Štruktúre stránok a ovládacích prvkov,
- Štýlom a dizajnu,
- Interakčnej logike.

Každá z týchto významne odlišných súčastí webu má vlastné špecifiká, avšak ani jedna nie je zanedbateľná a v symbióze s ostatnými vedie k vytvoreniu takého *GUI,* ktoré užívateľovi úspešne sprostredkúva interné funkcionality za zachovania čo najvyššej miery komfortu pri práci so systémom ako takým.

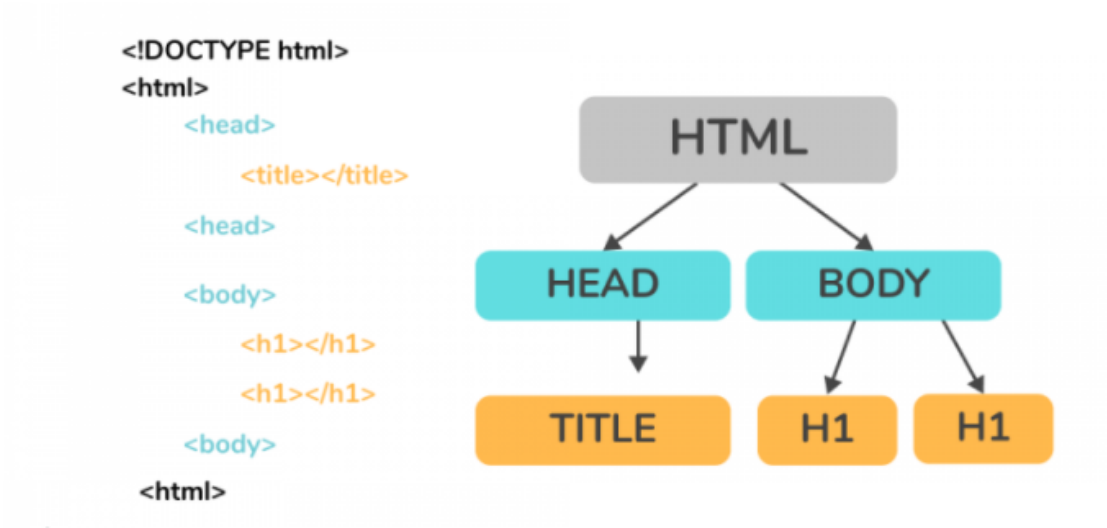
2.1.1 Štruktúra a štýly

Štruktúra a štýly webovej stránky propagujú prvý vizuálny dojem, ktorý stránka v užívateľovi zanechá. Navzdory tomu, že z hľadiska technického spracovania sú štruktúra a štýly dva oddelené celky, ich vzájomná súvislosť je natoľko tesná, že v kontexte obsahu webových stránok nemá význam skúmať ich separátne. V súčasnosti tieto koncepty na elementárnej úrovni prakticky kompletne pokrývajú dve technológie – *HTML* a *CSS*.

2.1.1.1 HTML

Hypertextový Značkovací Jazyk, alebo *HTML* (*Hyper Text Markup Language*) je dlhodobo základným staveným blokom štruktúry webového obsahu. Názov správne napovedá, že ide o značkovací / popisný typ jazyka – teda o jazyk, ktorý špecifikuje akým spôsobom má byť obsah zobrazený, resp. čo daný obsah vlastne znamená [41] – Umožňuje obohatenie textu o dodatočné informácie ohľadom jeho významu a štruktúry prostredníctvom možnosti využitia tzv. značiek (*markup tags*). Štrukturálny prvok tvorený značkou, jej obsahom a atribútmi (ako napr. naviazané štýly) nazývame *HTML* elementom. Z týchto elementov je v rámci prehliadača konštruovaný tzv. *DOM* (*Document Object Model*) [42]. *DOM* je možné si predstaviť ako viacúrovňovú stromovú štruktúru odrážajúcu organizáciu elementov v rámci webovej stránky za účelom stanovenia jednoznačnej hierarchie umožnenia programového prístupu k jednotlivým prvkom *HTML* dokumentu, napríklad prostredníctvom jazyka *JavaScript*.

HTML – keďže sa jedná o hypertextový jazyk - umožňuje prepojenie webových stránok medzi sebou navzájom, a to ako v rámci jednej stránky, tak medzi rôznymi webstránkami.



Obrázok 5. HTML štruktúra jednoduchej webovej stránky [43]

Existuje veľké množstvo alternatív spadajúcich do rodiny značkovacích jazykov (*SGML, KML, XML*). Mnohé z týchto jazykov dokonca podporujú odkazovanie naprieč dokumentmi či ich časťami - najviac sa pre vývoj webových aplikácií zaužíval však práve *HTML* (Alternatívne je použiteľný hybridný jazyk *XHTML* kombinujúci výhody rozšíriteľnosti a syntaktickej kontroly jazyka *XML* so základnými stavebnými blokmi využívanými v rámci *HTML*) [44]. Samozrejme existuje celý rad nástrojov a prostriedkov umožňujúcich tvoriť obsah webových stránok prostredníctvom iných značkovacích jazykov – zvyčajne je však nutná konverzia finálneho produktu do niektorého z jazykov podporovaných v rámci webových prehliadačov.

[45, 46] Jednotlivé verzie HTML prinášali rozmanitú škálu nových funkcionalít:

- **HTML 1 (1993):**
  Základná verzia *HTML*. Podporuje základné kontrolky a obrázky, disponuje však iba obmedzenými možnosťami aplikácie štýlov, bez podpory tabuliek či fontov.
- **HTML 2 (1995):**
  HTML 2 odráža aplikáciu prvých regulácií a pravidiel platných naprieč rôznymi prehliadačmi stanových organizáciou *W3C* (*World Wide Web Consortium*), medzinárodnej komunity, ktorá sa angažuje v procese štandardizácie webových technológií a dohliada na ich spracovanie webovými prehliadačmi za účelom poskytnutia čo najvyššej miery interakčnej konzistencie. *HTML 2* zavádza mnoho nových značiek, vrátane tabuliek a formulárov.
- **HTML 3.2 (1997):**
  Uvedenie mnohých nových formulárových elementov a prvotná podpora pre interakciu s *CSS* (*Cascading Style Sheets*), ktoré umožňovali aplikovať na *HTML* elementy štýly a tým výraznejšie ovplyvňovať, akým spôsobom bude obsah zobrazovaný webovými prehliadačmi.
- **HTML 4.01 (1999):**
  Umocnenie podpory *CSS* – vzniká koncept externých *CSS* hárkov obmedzujúcich duplikáciu *CSS* kódu (v predchádzajúcej verzii muselo byť *CSS* súčasťou *HTML* stránky, vo verzii *HTML 4.01* je možné hárok obsahujúci štýly do stránok linkovať, a teda zdieľať jeho obsah medzi viacero rôznych stránok) [47].

- **HTML 5 (2014+):**
  Súčasná verzia *HTML*. Okrem nových elementov podporujúcich pokročilú formulárovú funkcionalitu (*Email*, *Password*), mediálny obsah (*Audio*, *Video*) a dodatočné možnosti členenia stránky (*Section*, *Aside*) uviedla množstvo aplikačných rozhraní pre podporu moderných webových možností *(Geolokácia, Drag & Drop, Cache*…) [48].

Existencia potenciálnych ďalších verzií *HTML* je v súčasnosti otázna - verzia *HTML 5* je koncipovaná ako tzv. „Živý štandard“ [49, 50]. Fakticky ide o experimentálnejší prístup k štandardizácii, ktorý favorizuje dynamický vývoj, špecifikáciu a zavádzanie nových schopností technológie *HTML* na základe potrieb a požiadaviek užívateľov či užívateľských skupín.

2.1.1.2 CSS

CSS (*Cascading Style Sheets*) je jazykom pre popis prezentácie webových stránok, resp. dekorácie *HTML* elementov, ktorými sú tvorené – zahŕňa možnosti manipulovať s farbami, rozložením, fontami a v novších verziách obsahuje aj podporu pre tvorbu prechodov, animácií či transformácií. Okrem zabezpečenia vizuálnej atraktivity stránok umožňuje adaptovať prezentáciu s ohľadom na rôzne typy zariadení, disponujúce výrazne odlišnými veľkosťami zobrazovacej plochy [51, 52].

CSS je ako funkčný celok nezávislé od *HTML* a môže byť využité v spojení s rôznymi inými značkovacími jazykmi (napr. *XML*). Moderný prístup k vývoju webových stránok odporúča separáciu štruktúry obsahu od prezentácie – t.j. oddelenie zápisu *CSS* štýlov od dokumentu reprezentujúceho obsah *HTML* stránky, čím sa zjednoduší údržba stránok a sprístupní možnosť zdieľania jedného *CSS* hárku naprieč viacerými *HTML* dokumentmi - alebo naopak dokonalejšie oddelenie prezentácie jednotlivých stránok v prípade potreby dôkladnej individuálnej personalizácie.

CSS operuje nad elementami *HTML* dokumentu prostredníctvom tzv. selektorov [53]. Selektor v kontexte *CSS* je prvou časťou *CSS* pravidla – ide vlastne o reťazec pomenovaní elementov a pomocných výrazov, ktoré prehliadaču špecifikujú, ktoré *HTML* elementy sú daným pravidlom ovplyvnené. *CSS* selektory môžu byť rozmanitej povahy – od obecných selektorov cez názvy elementov, pomocné atribúty elementov (ako napr. *HTML* atribút „*class*“ či „*id*“) až po komplikované hierarchické štruktúry určujúce subjekt daného pravidla prostredníctvom jeho predkov / potomkov vzhľadom na *DOM* webstránky [53, 54].

Obrázok 6. Korektný formát zápisu CSS pravidla [55]

Podobne ako HTML, aj CSS má za sebou dlhú evolúciu:

- **CSS 1 (1996):**

  Sprostredkúva možnosti primárne pre pridávanie či úpravu vlastností textu (ako napríklad typ, font, dôraz, farba popredia či pozadia). Taktiež uviedlo podporu pre ďalšie prevažne textovo-orientované atribúty – veľkosť medzier medzi slovami, písmenkami či riadkami textu. Webové prehliadače však v čase vytvorenia prvej verzie *CSS* neboli na tieto zmeny okamžite pripravené a verzia *CSS 1* sa príliš nerozšírila.

- **CSS 2.1 (1998):**

  Špecifikácia bola vyvinutá organizáciou *W3C* a publikovaná ako odporúčanie v roku 1998. *CSS 2.1* uvádza množstvo vlastností určených k úprave pozície elementov na stránke, čím sa ukotvil význam *CSS* ako nástroja pre tvorbu a moduláciu obecného rozloženia webovej stránky. *CSS 2.1* taktiež uviedlo nové typy selektorov pre možnosti sofistikovanejšieho výberu elementov v rámci stránky.

- **CSS 3 (1999 +):**

  Posledný platný *CSS* štandard, uvádza podporu viacerých farebných schém (*HSL*, *RGBA*), priehľadnosti, tieňovania, zaoblených rohov elementov či animácií bez nutnosti využitia *JavaScript*-u. Ďalej sprostredkúva využitie atribútových selektorov a tzv. *„media queries"* na určenie veľkosti zobrazovacej plochy a dynamické prispôsobenie obsahu potrebám konkrétneho zariadenia [56, 57].

Navzdory značnému veku aktuálnej verzie nebola nikdy nebola oficiálne štandardizovaná verzia *CSS 4*. Dôvodom je to, že v súvislosti s významným nárastom rozsahu CSS špecifikácie a divergenciou progresu vývoja jednotlivých modulov sa miesto verzovania kompletnej CSS špecifikácie stalo efektívnejším vyvíjať a zverejňovať odporúčania pre jednotlivé funkčné moduly separátne [58].

2.1.1.3 Frameworky

Podobne ako u väčšiny technológií, aj v oblasti tvorby obsahu a štýlov webových stránok vzniklo veľké množstvo nádstavieb a frameworkov.

[59, 60] Frameworky si kladú niekoľko cieľov, ktoré vo väčšej či menšej miere napĺňajú:

- **Časová úspora:**
  Framework umožňuje vyhnúť sa situácii, kedy vývoj *Frontend*-u začína od nuly. Namiesto vytvárania elementárnych komponent sa vývojár môže zamerať priamo na skladanie predpripravených elementov do požadovaného tvaru.
- **Štandardizácia a výkon:**
  Stránka vyvinutá s využitím frameworku môže byť efektívnejšia a rýchlejšia, keďže populárne frameworky zvyknú byť optimalizované pre výkon v rámci viacerých webových prehliadačov.
- **Škálovateľnosť a aktuálnosť:**
  Voľba aktuálneho a kontinuálne vyvíjaného frameworku poskytuje záruku prístupu k vylepšeniam a funkcionalitám uvedeným jeho novšími verziami, čo napomáha s udržiavaním aktuality a konzistencie stránok v súlade s najnovšími trendami webového vývoja.
- **Spoľahlivosť:**
  Vzhľadom k zvyčajne bezplatnej povahe frameworkov pre vývoj *Frontend*-u a ohromnou mierou vyžitia v rámci vývojárskej komunity je takmer isté, že prípadné chyby predchádzajúcich verzií frameworku už boli nahlásené a opravené, pričom neustále prebieha práca na zlepšení pretrvávajúcich kvalitatívnych nedostatkov.

Navzdory nespočetnému množstvu dostupných frameworkov si dlhodobo najväčšiu prezenciu na trhu udržiavajú frameworky *Bootstrap* a *Foundation* [59, 60].

Obrázok 7. Porovnanie miery rozšírenia najpopulárnejších HTML / CSS framework-ov v priebehu posledných rokov (percentuálne hodnoty zodpovedajú podielu vývojárov ktorí uviedli, že danú technológiu už použili k celkovému počtu respondentov)[61]

Bootstrap

Bootstrap je pravdepodobne najznámejším a najvyužívanejším frameworkom pre tvorbu obsahu webstránok (bez zohľadnenia frameworkov zabezpečujúcich interakčnú logiku). Vyvinutý firmou *Twitter* v roku 2010 okamžite po uvedení na trh zaznamenal mimoriadne rýchly nárast obľúbenosti naprieč vývojárskou komunitou [62].

Bootstrap využíva špecifickú formu tvorby rozloženia stránky – stránku vníma ako responzívnu mriežku s 12-timi stĺpcami schopnými prispôsobovať svoje správanie a rozloženie s ohľadom na štyri rôzne veľkosti zobrazovacích plôch v závislosti na rozmeroch displeja konkrétneho zariadenia.

Bootstrap ponúka preddefinované štýly pre všetky štandardné *HTML* elementy a dokonca implementuje vlastné moduly v jazyku *JavaScript* určené pre manipuláciu s dynamickými prvkami (drop-down zoznamy, interaktívne navigačné menu..) [60, 62, 63].

Ďalším z veľkých lákadiel frameworku *Bootstrap* je aj masívny ekosystém zahŕňajúci detailnú dokumentáciu vrátane ohromnej kvantity šablónového kódu pre použitie v rámci vlastnej implementácie - témy, UI elementy a iné „hotové" riešenia významne zjednodušujú vývoj obsahu webstránok – na druhú stranu môže ich extenzívne využitie viesť k znechuteniu niektorých vývojárov z rozsiahlych podobností medzi stránkami, ktoré tieto *Bootstrap* šablóny implementujú [64].

Foundation

Framework *Foundation* bol vytvorený firmou *ZURB* v roku 2008. Zakladá sa na využití technológií *HTML*, *CSS* a *jQuery*. *Foundation* disponuje vlastným *CLI* (*Command Line Interface*) s podporou škály inštalačných, konfiguračných, kontrolných a ladiacich príkazov. Podobne ako konkurenčný *Bootstrap* využíva tzv. „Mobile-First" prístup k vývoju, t.j. vývoj s dôrazom na responzivitu a korektné podanie obsahu stránok na variabilných veľkostiach zobrazovacích plôch rôznorodých zariadení [63].

Foundation využíva pre formu *CSS* označovanú ako *SASS (Syntactically Awesome Style Sheets*). Ide o rozšírenie jazyka *CSS* ktoré umožňuje lepšie kontrolovať a ovládať vzájomné hierarchické závislosti *HTML* elementov ovplyvnených pravidlami štýlov formou zápisu prostredníctvom vnorených štruktúr stromovitého charakteru – tento prístup vedie k nižšej duplicite kódu a lepšej čitateľnosti / modifikovateľnosti jednotlivých *CSS* pravidiel [65].

Využitie frameworku *Foundation* je bezplatné a podobne ako *Bootstrap* sa môže pýšiť aktívnou a rozsiahlou užívateľskou základňou.

2.1.2 Interakčná logika

Koncept dynamických webových stránok so sebou priniesol viacero výziev – mimo tému zabezpečenia komunikačnej prevádzky prevažne problematiku toho, ako korektne doručiť dáta nielen zo serveru smerom ku klientovi, ale aj opačným smerom (a následne vhodným, užívateľovi zrozumiteľným spôsobom reagovať na odpoveď). Bolo zrejmé, že vzniká nutnosť konštrukcie ľahkého a všestranne užitočného nástroja či jazyka pracujúceho v rámci webových prehliadačov, ktorý je schopný asynchrónneho spracovania komunikácie prebiehajúcej medzi prehliadačom užívateľa a vzdialeným serverom.

2.1.2.1 JavaScript

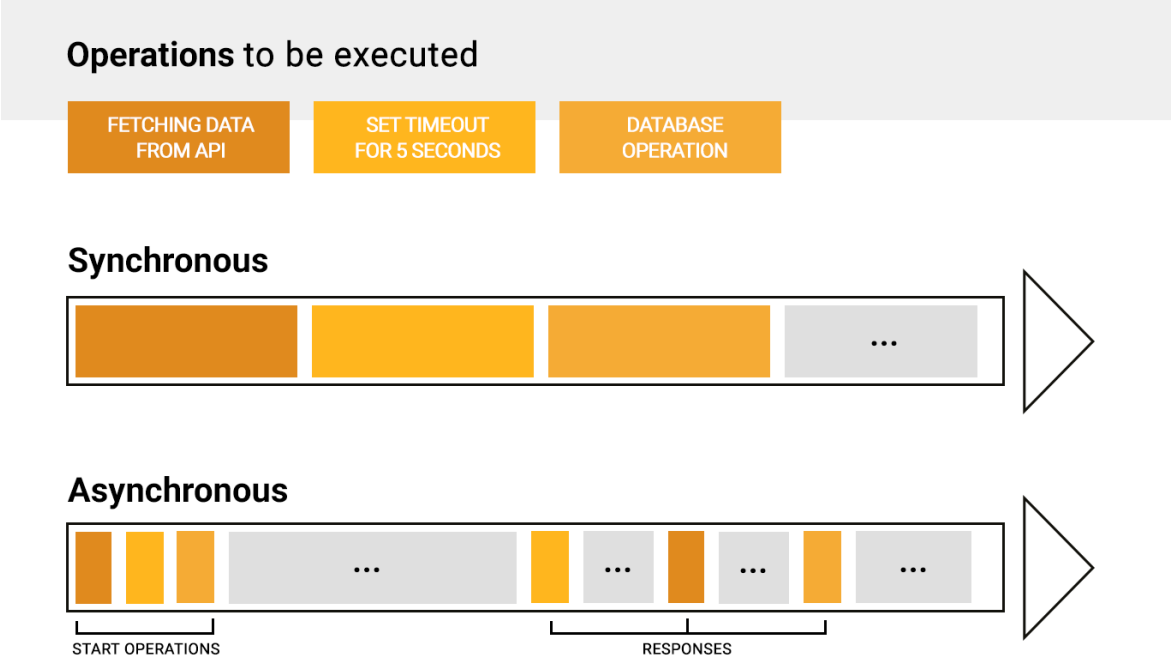
Navzdory verbálnej podobnosti názvov jazyky *JavaScript* a *Java* nemajú spoločné (okrem objektovo orientovaného prístupu a viacerých syntaktických podobností) vlastne prakticky nič. V súčasnosti je mnohými uznávanou hypotézou to, že bizarná zhoda pomenovaní vznikla kvôli snahe pôvodného *JavaScript*-u priživiť sa na popularite taktiež vznikajúceho jazyka *Java*, ktorý mal v danej dobe výhodu výnimočne efektívneho marketingu a kladnej povesti [66].

Súčasný *JavaScript* je ľahkým, jednovláknovým, interpretovaným, dynamickým, objektovo orientovaným skriptovacím jazykom s mimoriadne silnou podporou asynchrónnosti v rámci spracovania udalostí. Mimo objektovo orientovaného programovacieho štýlu je taktiež schopný podporovať imperatívny a deklaratívny prístup [67, 68].

Syntakticky sa zaraďuje JavaScript do syntaktickej rodiny spolu s ostatnými jazykmi, ktorých elementárne syntaktické pravidlá vychádzajú z prototypového jazyka Algol. Z toho dôvodu je kód tvorený v jazyku *JavaScript* vizuálne podobný napr. kódu tvorenému jazykmi ako napr. *Pascal*, *C*, *Java* či *C#.*

[68, 69] *JavaScript* rozlišuje podľa synchrónnosti operácie na dva hlavné typy:

- **Synchrónne:**
  Synchrónne operácie bývajú zvyčajne časovo nenáročné akcie – výpis do konzoly, jednoduchá aritmetická operácia, zmena textu elementu v rámci *DOM* webovej stránky, typová konverzia a pod. Skutočnosť, že tieto operácie prebiehajú v hlavnom vlákne nemá kvôli rýchlosti ich spracovania prakticky žiadny vplyv na odozvu užívateľského rozhrania webu. Pri korektnej implementácii by nemali byť synchrónne realizované dlhšie výpočty či komunikácia po sieti.
- **Asynchrónne:**
  Najčastejšími zástupcami asynchrónneho typu operácií v jazyku *JavaScript* sú napríklad *HTTP* požiadavky, databázové operácie a funkcie ktoré sú podmienené udalosťami generovanými nejakou implementáciou časovačov.



Obrázok 8. Rozdiely spracovania identického sledu operácií vykonávaných synchrónnym a asynchrónnym prístupom [70]

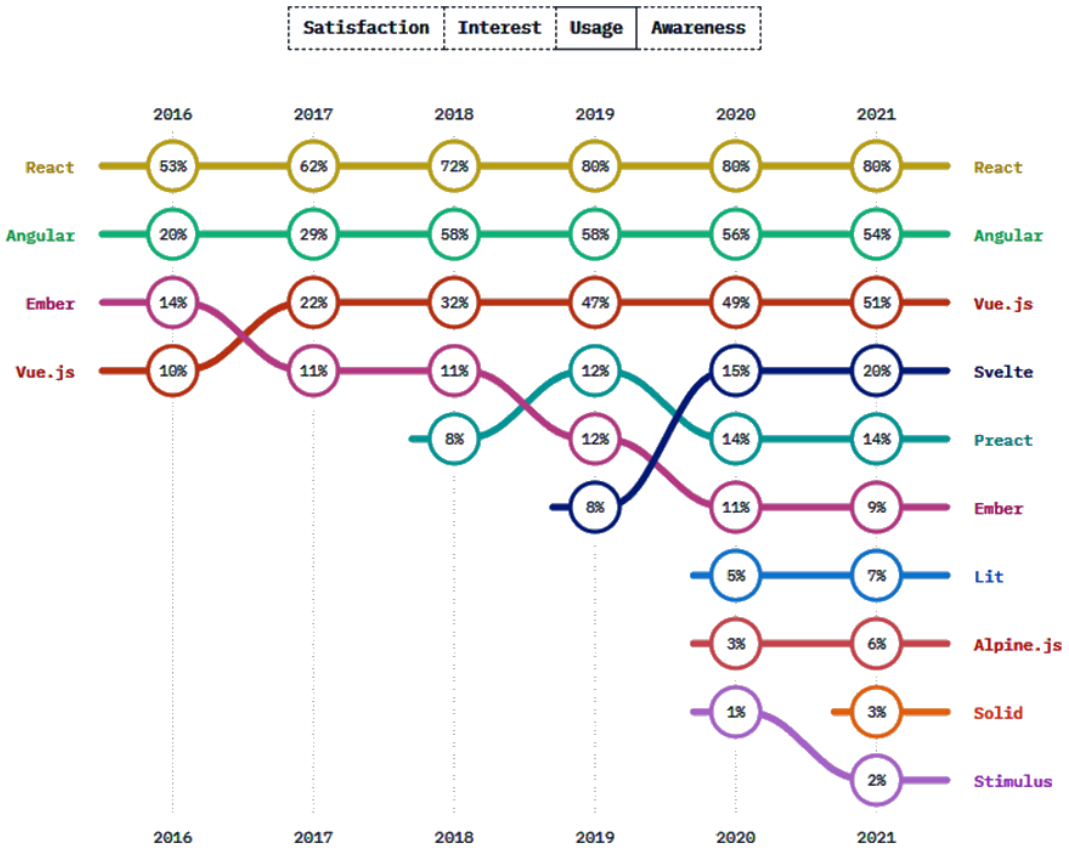
Asynchrónne možnosti jazyka *JavaScript* vychádzajú z nutnosti komunikovať z prostredia klientovho webového prehliadača so vzdialeným serverom (čo môže byť v závislosti od konkrétnej operácie požadovanej po serveri časovo značne náročná činnosť). Aby sa predišlo zamrznutiu užívateľského rozhrania celej webovej aplikácie počas dlhotrvajúcich činností (resp. aktívneho očakávania odpovede), implementuje *JavaScript* spracovanie odpovede prostredníctvom *Callback* funkcií [69, 71].

JavaScript vo vzťahu k funkciám volí tzv. *First-Class* prístup – to znamená, že v rámci umožňuje manipulovať s funkciou ako s ľubovoľnou inou premennou, vrátane schopnosti poslať referenciu na túto funkciu do volania inej funkcie formou argumentu. Tento argument (v prípade, že ho v „hosťovskej“ funkcii neskôr skutočne využijeme) nazývame *Callback*. Využitie *Callback* funkcií v rámci asynchrónneho spracúvania požiadaviek je mimoriadne výhodné – poskytujú prostriedky pre lepšiu kontrolu toku spracovania programu a schopnosti reagovať na dokončenie činností aj bez nutnosti aktívneho čakania [71]. *JavaScript* má okrem zabezpečovania webovej komunikácie na strane klienta aj mnoho iných typov využitia – vďaka objektovému prístupu k uzlom *DOM* umožňuje efektívne manipulovať s jednotlivými elementami webovej stránky a ich atribútmi.

Okrem už spomenutého *JavaScript* poskytuje aj rozsiahle možnosti skvalitnenia vizuálneho aspektu stránok schopnosťou implementovať sofistikované animácie, fascinujúce efekty (mnohokrát v miere zložitosti na ktorú bežné, statické *CSS* nepostačuje), ale aj komunikovať zmeny stavu aplikácie prostredníctvom modifikácie vzhľadu vybraných elementov (napr. na základe predchádzajúcich užívateľských akcií).

2.1.2.2 Frameworky

Podobne ako u každej populárnej technológie, aj JavaScript položil základy pre množstvo frameworkov s odlišnými motiváciami – či už ide o snahu odstrániť / zamaskovať nevýhody čistého *JavaScript*-u, rozšíriť jeho možností a schopnosti alebo zaručiť lepšiu komunikáciu s ostatnými často používanými (alebo naopak, menej známymi) technológiami a nástrojmi, frameworky založené na jazyku *JavaScript* tvoria početnú skupinu – a veľké množstvo z nich je v súčasnosti významnou súčasťou sveta moderného webového vývoja [72, 73].



Obrázok 9. Porovnanie miery rozšírenia najpopulárnejších JavaScript framework-ov a knižníc v priebehu posledných rokov (percentuálne hodnoty zodpovedajú podielu vývojárov ktorí uviedli, že danú technológiu už použili k celkovému počtu respondentov) [74]

Vue.js

Vue.js sa v posledných rokoch stáva jedným z najpopulárnejších *JavaScript* frameworkov vďaka svojej flexibilite a dobrou symbiózou s rôznymi ďalšími skupinami technológií (napr. *PHP* frameworkom *Laravel*). Oficiálna dokumentácia uvádza *Vue.js* ako progresívny framework určený k stavbe užívateľských rozhraní, ktorý je na rozdiel od monolitických frameworkov koncipovaný modulárne, s ohľadom na inkrementálne nasadenie [75].

Hlavnými výhodami frameworku *Vue.js* je intuitívnosť a možnosť konštrukcie *GUI* prostredníctvom tzv. komponent – systém komponent poskytuje abstrakciu nad elementárnymi časťami užívateľského rozhrania s cieľom podporiť čitateľnosť, modularitu a opakovanú použiteľnosť kódu prostredníctvom jeho rozloženia na malé, uzatvorené celky.

Vue.js ďalej definuje sadu direktív, odomykajúcich možnosti pokročilej manipulácie s jednotlivými komponentami (ako napr. direktíva *v-bind*, ktorá umožňuje do komponent posielať parametre, čím vlastne podnecuje tvorbu komponent v tvare generických šablón) [75, 76].

Angular

Angular je webový aplikačný framework vyvinutý firmou *Google* a prvýkrát sprístupnený v roku 2016. Je súčasťou skupiny technológií určenej k budovaniu dynamických webových stránok (tzv. *MEAN Stack*) [77], ktorá okrem frameworku *Angular* samotného zahŕňa dokumentovú databázu *MongoDB*, *Express.js* framework pre tvorbu aplikačných serverov a *Node.js* ako exekučné prostredie serveru.

Motiváciou za vznikom frameworku *Angular* bolo rozšírenie možností *HTML* s cieľom umožniť dátové preväzovanie (*data binding*) zobrazenia na vlastnosti a polia podkladových modelov skrz špeciálne direktívy. Tieto direktívy môžu byť vnímané ako špeciálny spôsob značenia elementov v rámci *DOM* modelu stránky (prostredníctvom atribútov, vyhradených názvov alebo tried), ktoré signalizujú *HTML* kompilátoru integrovanému v rámci *Angular* nutnosť pripojiť k takto označeným elementom špecifické správanie [78] (a prípadne ku nim naviazať prvok nazývaný *Event Listener*, zodpovedný za propagáciu prípadných zmien do detských elementov tohto prvku). Tento účel plnia špeciálne direktívy začínajúce prefixom „*ng*“ – napr. *ng-app*, *ng-repeat*, *ng-include*, *ng-click*...

Aplikácie postavené na frameworku *Angular* implementujú *MV** (*Model – View - Whatever*) architektonický model za využitia zabudovaného mechanizmu *Dependency Injection*. *MV** model je podobný architektonickému vzoru *MVC* [viď. kapitola *3.1 - „Model – View - Controller*] s tým rozdielom, že nekladie dôraz na spôsob implementácie previazanie dátového modelu a zobrazenia – podstatné je iba to, aby sa zmeny modelu premietli do zobrazenia a naopak. Prakticky tak mnohé aplikácie napísané prostredníctvom frameworku *Angular* využívajú architektúru kombinujúcu prvky *MVC* s prvkami *MVVM* [72, 77].

2.1.2.3 Knižnice

Enormné využitie technológie *JavaScript* v rámci webového vývoja podnietilo nielen vývoj aplikačných frameworkov, ale aj nepreberného množstva knižníc značného rozsahu, ktoré si vybudovali podporu vývojárskej komunity. Niektoré z týchto knižníc sú z pohľadu ponúkanej funkcionality a schopností tak rozsiahle, že sú mylne zamieňané za frameworky.

JQuery

JQuery je v súčasnosti už zrelou technológiou, ktorá sa na dejisku vývojových nástrojov objavila prvýkrát v roku 2006 (autorom je vývojár John Resig). Hlavnou úlohou *jQuery*, najmä v dobe vzniku tejto knižnice, bolo zjednodušiť činnosti, pre ktoré čistý *JavaScript* buď neposkytuje efektívnu a stručnú formu zápisu, alebo bez využitia dodatočnej vrstvy abstrakcie nedisponujú príliš vysokou mierou opakovanej použiteľnosti [79, 80]. Pre činnosti ako priechod objektovým modelom dokumentu, selekcia prvkov, spracovanie udalostí, animácie a v neposlednom rade vylepšená syntax pre *AJAX* (*Asynchronous JavaScript and XML*) volania bol príchod *jQuery* prirodzenou evolúciou.

Dlhodobo najpopulárnejšie rozšírenie jazyka *JavaScript*, je v súčasnosti stále mimoriadne používané – aj keď je už mnoho neduhov čistého *JavaScript*-u, ktoré táto knižnica v prvopočiatkoch svojej existencie adresovala už prepracovaných do použiteľnejšej podoby. Kvôli príchodu modernejších frameworkov a nástrojov začína adopcia v *jQuery* v rámci nových projektov stagnovať až mierne upadať [81] – to ale nič nemení na skutočnosti, že sa stále jedná o schopnú a ľahkú technológiu hojne zastúpenú v prostredí Internetu – a to nepochybne aj kvôli nezanedbateľnému množstvu dlhodobo vyvíjaných masívnych informačných systémov, ktoré sú na nej založené.

React JS

React je rozsiahlou knižnicou postavenou na jazyku *JavaScript* (nezriedka mylne označovanou za kompletný framework). Bol sprístupnený firmou *Facebook* v roku 2013 a konceptuálne je značne podobný frameworku *Angular JS* – taktiež sa snaží o zvýšenie modularity kódu prostredníctvom menších logických celkov zvaných komponenty [82]. Knižnica *React* však nepotrebuje zasahovať do inej logiky než do samotného vykresľovania *UI*, čím poskytuje väčšiu implementačnú voľnosť v smere voľby ďalších vývojových technológií a pootvára dvere jednoduchšiemu inkrementálnemu nasádzaniu do už zavedených projektov. Populárnymi aplikáciami vybudovanými s využitím knižnice *React JS* sú napr. *Instagram* či *Yahoo Mail* [83].

Tvorba *UI* prostredníctvom knižnice *React* využíva virtuálny *DOM* – z pohľadu knižnice *React* je každý komponent súhrnom *HTML* a *JavaScript*-u, pričom implicitne implementuje logiku potrebnú k vykresleniu (rendering) sekcie *UI* reprezentovanej daným komponentom v rámci stromovej štruktúry *DOM*. Jednotlivé komponenty majú vlastný stav – *React* zabezpečuje naviazanie tohto stavu na príslušný obslužný tzv. E*vent Handler*, ktorý je vlastne funkciou reagujúcou na zmenu stavu komponentu. Táto interakcia efektívne prepája zobrazenie komponentu (*View*) s podkladovým modelom poskytujúcim dáta, ktoré komponent zobrazuje.

Kontrola efektov na naviazané dáta v rámci knižnice *React* prebieha pomocou tzv. „rozdielového algoritmu“ (*Diffing Algorithm*). Pokiaľ dôjde k zmene vlastností či stavu komponenty, *React* implicitne rozhodne, či je potrebná aktualizácia *DOM* prostredníctvom porovnania nového elementu s elementom doteraz vykresleným. Pokiaľ je aktualizácia nutná, dochádza k prekresleniu reálneho *DOM* modelu stránky a v dôsledku k prekresleniu aplikácie [82, 83].

2.2 Backend

Backend nazývame časť webovej aplikácie, ktorá je prevádzkovaná serverom a jeho hlavnou úlohou (na rozdiel od *Frontend*-u) nie je logika vizualizačného charakteru, ale spracovanie dát, zabezpečenie prístupu k dátovým zdrojom a v neposlednej rade predspracovanie týchto dát do tvaru akceptovaného klientom. *Backend* naďalej spracúva procesy na pozadí, ktoré môžu (a nemusia) byť iniciované interakciami užívateľa s aplikáciou. Užívateľ s logikou *Backend*-u spravidla neprichádza do priameho kontaktu, ale pozoruje jej dopady - vníma ju ako tzv. „Čiernu Skrinku“ (*Black Box*).

Uvedený popis značí, že pod pojem *Backend* v určitom zmysle slova spadajú aj knižnice, systémové komponenty či celé systémy bez užívateľského rozhrania [84].

Podobne ako u *Frontend*-u, aj v prípade vývoja *Backend*-u v priebehu rokov vykryštalizovala rada technológií, ktoré sa ustálili ako overené a vhodné nástroje pre riešenie daného typu problému. Najpopulárnejšie z nich sú popísané v kapitolách nižšie.

2.2.1 Business logika

Neodmysliteľnou súčasťou *Backend*-u je súhrn pravidiel a algoritmov spoločne nazývaných Business logika. Business logika – ako názov napovedá - zahŕňa všetky časti kódu aplikácie adresujúce problémy „definované obchodom“ – t.j. problémy, ktorých efektívne riešenie je primárnou úlohou daného softwaru [85, 86]. Business logika stanovuje akým spôsobom môžu byť dáta vytvárané, ukladané či modifikované a akou formou spolu objekty interagujú – business logika by v rámci aplikácie mala byť v ideálnom prípade čo najviac oddelená od logiky infraštruktúrnej. Jej pravidlá sú ovplyvňované pravidlami skutočného sveta a špecifikami oblastí záujmu, pre ktoré je aplikácia určená – tieto pravidlá definujú množinu operácií, definícií a obmedzení kladených povahou procesov, ktorých riešenie má systém sprostredkúvať.

2.2.1.1 Programovacie jazyky

Základné vlastnosti, syntax a princípy fungovania populárnych programovacích jazykov využívaných k tvorbe *Backend*-u webových aplikácií sa nezriedka výrazne líšia – aj tak je možné pozorovať niektoré spoločné znaky. Väčšina v súčasnosti využívaných programovacích jazykov pre vývoj *Backend*-u sa vyznačuje dobrými rýchlostnými parametrami, škálovateľnosťou, podporou paralelizmu (resp. schopnosťou obsluhy veľkého množstva požiadaviek zároveň) a v neposlednej rade integráciou populárnych frameworkov ponúkajúcich sadu nástrojov a funkcií za účelom zvýšenia celkovej efektivity vývoja. Kapitoly nižšie popisujú výber z najrozšírenejších programovacích jazykov vhodných pre vývoj *Backend*-u. Jazyk *C#* (a s ním súvisiaci framework *ASP.NET Blazor*), na ktorých sa zakladá implementačná zložka praktickej časti tejto práce je následne predstavený v samostatnej kapitole [viď. kapitola 4 - *„ASP.NET Core Blazor*].

PHP

PHP je už dlhodobo nesporne najrozšírenejším programovacím jazykom určeným pre vývoj *Backend*-u [87]. Jedná sa o interpretovaný, skriptovací, rýchly a flexibilný jazyk mimoriadne vhodný pre webový vývoj. Súčasný stav webových technológií a rozvoj nástrojov / frameworkov stavajúcich na alternatívach už ponúka konkurencieschopné technológie / jazyky, navzdory tomu však *PHP* svoj gigantický náskok- čo do miery súhrnného využitia v rámci Internetu – svoj podiel stráca iba veľmi pomaly [88], pričom absolútny počet webstránok ktoré nejakým spôsobom implementujú *PHP* stále pozvoľna narastá. Populárnymi webovými stránkami / portálmi, ktoré sú založené na PHP sú mimo iné giganty ako *Facebook*, *Wikipedia*, *Tumblr*, *Slack* a *Etsy* [89]. Veľkým dielom k rozšíreniu *PHP* určite prispieva aj najznámejší systém pre správu obsahu (*CMS - Content Management System*) a síce *WordPress* [89], ktorý sprístupňuje tvorbu webových stránok aj pre osoby bez programátorských znalostí s využitím šablónovacích mechanizmov ovládaných prostredníctvom grafického rozhrania.

[90, 91] Charakteristické znaky jazyka PHP:

- **Bezplatnosť:**
  PHP je open-source projekt. Vďaka tomu je možné nahliadnuť do jeho zdrojového kódu, vrátane *PHP* interpreteru (napísaného v programovacom jazyku *C*). To samozrejme mimo bezplatnosti znamená aj to, že sa rozsiahla *PHP* komunita môže aktívne spolupodieľať na optimalizáciách, odstraňovaní chýb a dlhodobom zlepšovaní jazyka.
- **Multiplatformnosť:**
  PHP je prirodzene multiplatformným jazykom – nezáleží na tom, či webový server využíva ako svoj operačný systém *Windows, Mac OS* alebo *Linux*.
- **Rozšírenie:**
  V Internetovom prostredí je *PHP* masívne rozšíreným programovacím jazykom, ktorý sa teší rovnako rozsiahlej komunite a vysokej využiteľnosti na trhu práce (či už v dlhodobo prevádzkovaných alebo nových projektoch).

- **Určenie pre Web:**
  Jazyk *PHP* bol špecificky vytvorený za účelom tvorby dynamických webových stránok. Vďaka tomu už vo svojom základe implementuje mechanizmy pre adresovanie programátorských problémov typických pre tento druh vývoja – napríklad jednoduchý prístup k databáze, spracovanie údajov z webových formulárov či správu cookies.

Java

Java je objektovo orientovaným programovacím jazykom všeobecného zamerania [92]. Dlhodobo sa teší mimoriadnej popularite a stala sa ustálenou technológiou prakticky akejkoľvek vysokoúrovňovej formy vývoja. V kontexte webového vývoja sa *Java* v minulosti využívala primárne ako nástroj na tvorbu appletov – miniaplikácií, ktoré bežali v prostredí webovej stránky a obsluhovali užívateľské interakcie, či už s asistenciou webového serveru slúžiaceho ako *Backend* aplikácie, alebo výhradne v priestore klienta.

Jednou z najvýznamnejších výhod jazyka *Java* je jej snaha o dodržiavanie princípu *WORA* (*Write Once Run Anywhere*) – skompilovaný kód vytvorený v jazyku *Java* je spustiteľný na ľubovoľnej platforme či operačnom systéme, ktorý podporuje *Java* behové prostredie bez potreby rekompilácie. Táto vlastnosť je dosiahnutá vďaka skutočnosti, že *Java* je kompilovaná do formátu *bytecode* a spracúvaná jednotne prostredníctvom *JVM* (*Java Virtual Machine*), bez ohľadu na architektúru hosťujúceho zariadenia [92, 93].

Technológia *Java* implementuje pomerne nezvyčajný vývojový model – disponuje rôznymi, samostatne vyvíjanými platformami ktoré podporujú programovací jazyk *Java*.

Všetky z nižšie uvedených vývojových platforiem *Java* pozostávajú z *JVM* a rozhrania pre programovanie aplikácií (*API – Application Programming Interface*). *JVM* zabezpečuje unifikované správanie programu na všetkých kompatibilných systémoch, pričom platformou špecifikované *API* poskytuje funkcie a softwarové komponenty nutné k samotnej tvorbe aplikácií [94].

[94] Súčasne používané platformy programovacieho jazyka Java:

- **Java SE (Standard Edition):**
  Poskytuje základné funkcionality, ktorými programovací jazyk *Java* disponuje. Definuje základné typy, objekty a pred-implementované triedy používané pre kľúčové programátorské úlohy ako je napr. sieťová komunikácia, prístup k databáze či zabezpečenie.
- **Java EE (Enterprise Edition):**
  Java EE sa zakladá na verzii *Java SE* a naviac rozširuje *API* a behové prostredie o prostriedky pre zaručenie kvalitnejšej škálovateľnosti, bezpečnosti a spoľahlivosti sieťových aplikácií veľkého rozsahu.
- **Java ME (Micro Edition):**
  ME verzia programovacej platformy *Java* obsahuje funkcionálne zúženú, hardwarovo nenáročnú implementáciu *API* a *JVM*, pričom jej primárnym účelom je poskytovať optimálne vlastnosti pre vývoj v prostredí, kde hrá rolu obmedzený výpočtový výkon a iné limitácie (napr. energetická náročnosť u mobilných zariadení).
- **JavaFX:**
  JavaFX je verzia platformy *Java* určená pre tvorbu ľahkých internetových aplikácií. Užívateľské rozhranie je zostavené prostredníctvom *Java API*. Uvádza mnohé zaujímavé prvky - hardwarovú akceleráciu, rozšírené mediálne schopnosti a efektívne využitie možností klientov disponujúcich vyšším výpočtovým výkonom pre sprostredkovanie moderného a plynulého zážitku z používania webovej aplikácie.

Python

Python je moderný programovací jazyk, prvýkrát sprístupnený v roku 1991 a navrhnutý tak, aby bol intuitívny, syntakticky nenáročný, expresívny a jednoduchý na naučenie a použitie. Implementuje zjednodušenú syntax podobnú anglickému jazyku bez zbytočných špeciálnych symbolov sťažujúcich čitateľnosť kódu [95]. Vďaka svojej nenáročnosti, prístupnosti a aktívnemu kontinuálnemu vývoju je dlhodobo jedným z najpoužívanejších programovacích jazykov.

Vlastnosti jazyka *Python* tvoria pomerne unikátny mix – jedná sa o *Open-Source*, interpretovaný a multiplatformný jazyk disponujúci podporou viacerých programovacích paradigiem vrátane procedurálneho, funkcionálneho, s dôrazom na objektovo orientované.

Python v kontexte vývoja webových aplikácií býva nasadený prevažne na serveri [96] a nezriedka je možné ho nájsť ako hnací motor pomocných aktivít v rámci webu – ako formu riadenia pracovných tokov či modulov externých funkcionalít ku ktorým aplikácia pristupuje, ako napr. strojového učenia či analýzy / spracovania dát.

[97] Jazyk Python so sebou nesie mnoho benefitov:

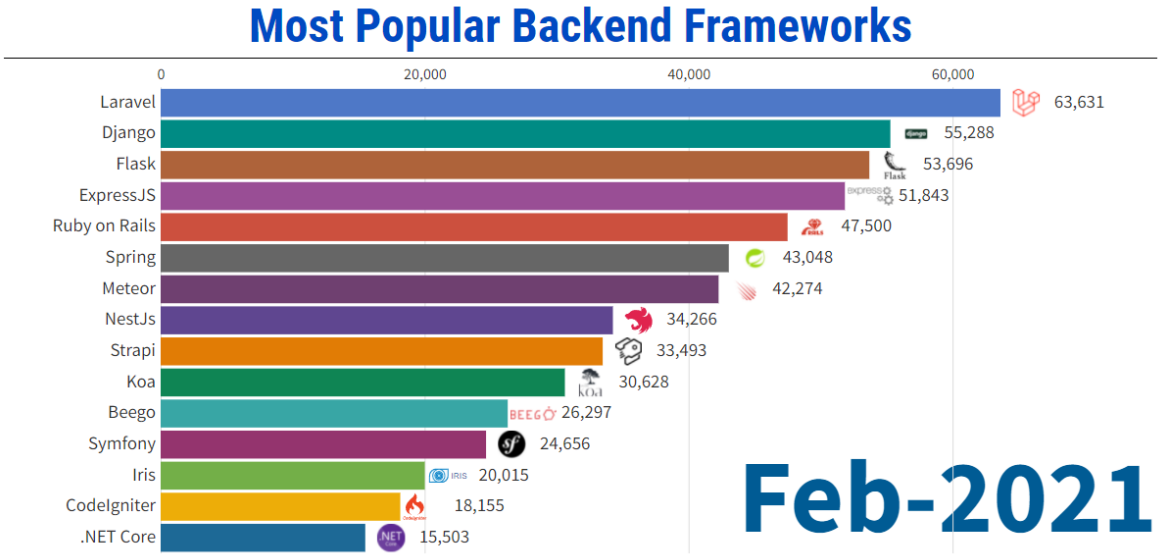
- **Jednoduchosť:**
  Python sa snaží riešiť problémy s využitím menšieho objemu expresívneho kódu a čo najviac minimalizuje využitie syntaktických prvkov ako sú napr. zátvorky či iné špeciálne znaky – efektívne kondenzuje a zjednodušuje príkazy a operácie, ktoré by vo väčšine iných programovacích jazykov vyžadovali značne komplexnejší zápis.
- **Strojové učenie / Dátová analýza:**
  Súčasnosť ponúka stále rozšírenejšie spektrum aplikácií a záujmových oblastí, v ktorých inteligentná analýza a spracovanie dát so zámerom ich zmysluplného využitia hrajú významnú úlohu. *Python* disponuje bohatým výberom knižníc pokrývajúcich túto oblasť a je celkovo najobľúbenejším jazykom pre tvorbu a nasádzanie postupov strojového učenia - čo dokazuje aj skutočnosť, že ho pre dané účely adoptujú aj známe mená ako napr. *Google*.
- **Open-Source:**
  Python je vyvíjaný pod *Open-Source* licenciou a teda jeho použitie a distribúcia nie je nijakým spôsobom spoplatnená – a to ani pre komerčné využitie. Zároveň (podobne ako u *Open-Source* jazykov a technológií popísaných v predchádzajúcich kapitolách) sa vďaka tomuto prístupu otvárajú možnosti včasného ladenia, odstraňovania chýb a optimalizácií plynúcich z možnosti komunity nahliadnuť do zdrojového kódu.
- **Multiplatformnosť:**
  Interpreter jazyka *Python* je multiplatformný a kompatibilný so všetkými väčšími operačnými systémami ako *Windows*, *Linux* či *Mac OS*. Za predpokladu, že v rámci *Python* programu nebudú použité systémovo závislé funkcie či premenné je teda kód plne prenositeľný.

- **Veľké množstvo knižníc:**

  Python integruje mnoho štandardných knižníc zastrešujúcich funkcionality rôznych programátorských smerov a zameraní ako napr. *Soft Computing*, webový vývoj či skriptovanie. V kombinácii s *Open-Source* prístupom tieto knižnice umožňujú tvoriť riešenia (aj komerčných) problémov rýchlo a s minimálnymi nákladmi.

2.2.1.2 Frameworky

Hlavným zmyslom takmer každého frameworku je zbaviť programátora nutnosti implementácie typických základných funkcionalít prostredníctvom poskytnutia určitých „stavebných kameňov“ a / alebo podkladovej štruktúry aplikácie. Výnimkou z tejto voľnej charakteristiky nie sú ani frameworky určené k tvorbe dynamicky generovaných webstránok, ktorých hlavnou úlohou je optimalizovať vývojársky čas poskytnutím hotových riešení typických problémov danej oblasti – v tomto zmysle hovoríme najčastejšie o sieťovom prenose, smerovacej funkcionalite, prístupu k databáze a súvisiacim *ORM* (*Object - Relational Mapping*) riešeniam, či šablónovacím systémom pre dynamickú tvorbu *HTML* dokumentov.



Obrázok 10. Najpopulárnejšie frameworky pre vývoj Backend-u webových aplikácií, dáta prieskumu zo začiatku roku 2021. Bodové hodnotenie je vypočítané na základe kvantity repozitárov vedených v rámci systému správy verzií GitHub [98]

Laravel

Laravel je v súčasnosti najpopulárnejším *PHP* frameworkom. Prvýkrát sprístupnený v júni roku 2011 je stále kontinuálne vyvíjaný a aktualizovaný – najčastejšie v nadväznosti na sprístupnenie novej verzie štandardu jazyka *PHP*. *Laravel* láka predovšetkým možnosťami efektívneho spravovania infraštruktúry aplikácie prostredníctvom architektonického modelu *MVC*, predpokladanú redukciu ceny vývoja a údržby software a zvýšenie efektivity programátora. Tieto očakávania napĺňa prostredníctvom čistej a expresívnej syntaxe a hotovými, pred-implementovanými celkami funkcionality štandardne využívanej v rámci webových stránok – ako je napr. prihlasovanie, správa užívateľov či možnosti testovania *API* [99]. Okrem možnosti implementácie jadrovej logiky aplikácie prostredníctvom predpripravených aplikačných rozhraní obsahuje niekoľko ďalších zaujímavých funkčných celkov.

[100] Významné funkčné komponenty v rámci frameworku Laravel:

- **Eloquent:**

  Eloquent je systém objektovo-relačného mapovania (*ORM*) špecifický pre framework *Laravel*. Umožňuje interagovať s akoukoľvek podporovanou databázou relačnej povahy prostredníctvom objektovo orientovaného prístupu priamo z jazyka *PHP*, bez nutnosti využívať databázovo špecifické technológie a jazyky ako napr. *SQL*. Okrem *CRUD* (*Create, Read, Update, Delete*) funkcionality zastrešuje tiež obšírne možnosti konfigurácie a správy vzťahov medzi entitami (resp. tabuľkami), vystavovanie databázových zmien s využitím mechanizmu migrácií a taktiež pokročilú funkcionalitu užitočnú vo webovom prostredí - ako napr. stránkovanie.

- **Blade:**

  Blade je šablónovací systém pre tvorbu dynamického webového obsahu. Zachováva možnosť využívať čistý *PHP* kód v šablónach a naviac prináša rozšírenie o ďalšie schopnosti prístupné prostredníctvom špeciálnych direktív. Okrem klasických *PHP* funkcií využívaných pri dynamickom generovaní *HTML* dokumentov (kľúčové slová *if-else, for, while*..) umožňuje syntakticky elegantným spôsobom pristupovať k hlbšej logike aplikácie – sprostredkúva napr. prístup k rodine autentifikačných direktív, ktoré podmieňujú vytvorenie blokov v rámci šablóny stavom prihlásenia či udelenými oprávneniami užívateľa, pre ktorého je obsah určený.

- **Artisan:**
  Artisan CLI (Command Line Interface) disponuje príkazmi určenými pre zvýšenie svižnosti vývoja a spríjemnenie práce programátora s frameworkom *Laravel*. Umožňuje spúšťanie migrácií, testov či plánovanie časovaných operácií. Modulárny spôsob implementácie rozhrania poskytuje možnosti definície vlastných príkazov v prípade, že by sa už poskytované rozhranie ukázalo ako nedostačujúce.

Ruby on Rails

Ruby on Rails – založený na jazyku *Ruby* – je framework určený k vývoju webových aplikácií využívajúci architektonický vzor *MVC*. Prvýkrát bol sprístupnený v januári roku 2012 ako prostriedok k urýchleniu vývoja. Pôvodná implementácia zahŕňala smerovací systém *(Journey Engine*), podporu databázového tázania a logovaciu funkcionalitu. Súčasný stav frameworku *Ruby* on *Rails* s kontinuálnym vývojom samozrejme adaptuje nové možnosti a schopnosti. Aktuálne kľúčové funkcionality, z ktorých sa *Ruby on Rails* skladá sú rozdelené do viacerých balíčkov (nazývaných niekedy aj sub-frameworky) - počet a zodpovednosti týchto sub-frameworkov sa v priebehu vývojového cyklu *Ruby on Rails* v minulosti menilo [101].

Stavebné kamene implementácie *Ruby on Rails* sú navrhnuté tak, aby mali symbiotický efekt, pričom ako nosné myšlienky sa uvádzajú „*Convention over Configuration*“ a „*Don´t Repeat Yourself*“ [101, 102].

- **Convention Over Configuration (CoC):**
  Ruby on Rails sa od mnohých konkurenčných vývojových frameworkov (frameworky jazyku *Java,* rôzne mutácie frameworku *.NET*) odlišuje tým, že nevyžaduje rozsiahly konfiguračný kód a pri automatizovaných činnostiach sa riadi primárne na základe odporúčaných menných konvencií. Zásah do konfiguračných štruktúr je nutný iba v prípade, že sa vývojár chce od konvencie nejakým spôsobom odchýliť.
- **Don´t Repeat Yourself (DRY):**
  Každá časť znalosti o systéme je stanovená unikátnou, definitívnou a relevantnou reprezentáciou, pričom znalosťou o systéme môžu byť myslené dáta alebo metadáta, logika, funkcionalita či algoritmus – *Ruby on Rails* podporuje opakovanú použiteľnosť singulárnych deklarácií.

> Dobrým demonštračným príkladom je implementácia prístupu k databáze prostredníctvom tzv. *Active Record* [viď. odstavce nižšie], pri ktorom programátor nemusí špecifikovať konkrétne názvy databázových stĺpčekov v objektoch na ktoré sa mapujú – framework túto znalosť jednoducho odvodí z databázového prostredia prostredníctvom automatizovaného implicitného mapovania pomenovaní existujúcich tried na relačné tabuľky.

Podobne ako u konkurenčných frameworkov, aj *Ruby on Rails* implementuje formu objektového prístupu k databázovým údajom – v tomto prípade realizovanú implementáciou návrhového vzoru *Active Record* [103]. Skratkovite sa jedná o systém pre *ORM* medzi objektami typu *Record* a databázovými tabuľkami, ktorý okrem *CRUD* funkcionality priamo z prostredia jazyka *Ruby* pokrýva aj možnosti konfigurácie a alternácie vzťahov medzi databázovými entitami, dodatočného dynamického načítania dát a dokonca aj zdieľanie autorizačných mechanizmov a validačných obmedzení medzi aplikáciou a databázovým prostredím.

Kompatibilitu s architektonickým vzorom *MVC* reprezentuje zložka pohľadu na dáta, t.j. zvyčajne stránka v *HTML* formáte. *Ruby on Rails* disponuje šablónovacími možnosťami založenými na formátoch *.RHTML* a *.RXML*. *Ruby on Rails* zabezpečuje sadu už implementovaných pomocných metód pre využitie v rámci tvorby týchto dynamických šablón – prevažne za účelom bežných úloh ako je konverzia formátov, správa chybových hlásení a navigácia. Úlohu kontrolérov potom – analogicky ako väčšiny konkurenčných webových frameworkov založených na vzore *MVC* - následne zastrešujú komponenty označované v kontexte *RoR* ako *Action Controller* [104].

Django

Zrod webového frameworku *Django* v roku 2003 značil značné posilnenie programovacieho jazyka *Python* v sfére webového vývoja – v čase, kedy paradoxne prevládal názor, že existuje príliš mnoho frameworkov pre webový vývoj v jazyku *Python*. *Django* je zrelý webový framework, ktorý bol v roku 2008 registrovaný pod *BSD* (*Berkeley Software Distribution*) licenciou a v roku 2013 sa dočkal prelomovej verzie *Django 1.5* zaisťujúcej kompatibilitu s práve prichádzajúcou novou verziou ním využívaného programovacieho jazyka – *Python 3* [105]. Môže sa pýšiť pôsobivou adopciou poprednými technologickými organizáciami – na frameworku *Django* je postavený portál *Mozilla.org*, *Openstack.org* či dokonca mobilná verzia webstránok spoločnosti *Instagram*.

Django je aj v dnešnej dobe stále aktívne vyvíjaný – bezpečnostné aktualizácie či drobné opravy sa objavujú až každý mesiac, pričom verzie prinášajúce väčšie zmeny vychádzajú zhruba raz za 8 mesiacov [106].

Analogicky k ostatným frameworkom popisovaným v tejto kapitole, aj *Django* adaptuje architektúru *MVC* – zvláštnosťou frameworku *Django* je poopravenie tohto architektonického vzoru do formátu *MVT* (*Model – View – Template*), kde šablóny (*Templates*) sú tvorené značkovacím jazykom *HTML,* pričom vizualizačná / dátová logika obsiahnutá v častiach typu *Model* a *View* je obsluhovaná programovacím jazykom *Python*. Zložka *View* v tomto špecifickom prípade zastupuje *Controller* a zabezpečuje prístup k navigácii, *CRUD* funkcionalite (prostredníctvom integrácie systému *Django ORM*) a v neposlednej rade plnenie a poskytovanie šablón vyžadovaných webovými klientmi [107].

Klasický proces spracovania požiadavky vo frameworku *Django* teda spočíva v predspracovaní požiadavky vrstvou middleware. Následne prebieha smerovanie (routing) na *View* definované vzormi vypozorovanými v *URL* požiadavky. Väčšina business logiky prebieha v kontexte daného *View*, ktoré pristupuje k modelom aj šablónam za účelom formovania odpovede [107]. Odpoveď je nakoniec odosielaná – predtým ako opustí server môže prebehnúť prípadné dodatočné post-spracovanie prostredníctvom ďalšej vrstvy middleware logiky.

Okrem už popísaných charakteristických vlastností a modulov *Django* disponuje mnohými ďalšími nástrojmi často vídanými v sfére frameworkov webového vývoja [105] - sprostredkúvajú pripojenie k databáze, *URL* smerovanie, podporu šablónovacej funkcionality, autentifikačné či autorizačné mechanizmy so správou užívateľov a podporu automatizovaného testovania.

2.2.2 Databáza

Neodmysliteľnou súčasťou drvivej väčšiny moderných aplikácií je určitá forma perzistentného dátového úložiska. Databázy umožňujú ukladanie veľkého množstva záznamov pokiaľ možno priestorovo efektívnym spôsobom s dôrazom na čo najvyššiu rýchlosť prístupu k dátam a rýchlosť operácií nad uloženou množinou dát – efektivita jednotlivých operácií realizovaných nad dátami je výrazne ovplyvňovaná štruktúrou, povahou a zameraním konkrétnej databázovej implementácie. Obecne delíme databázy podľa spôsobu tázania do dvoch hlavných skupín – *SQL* a *NoSQL* [108, 109].

Aplikácie nasadené v prostredí webového charakteru k databáze pristupujú zvyčajne z prostredia *Backend*-u – či je zariadenie prevádzkujúce *Backend* server zároveň zariadením, na ktorom je umiestnená adresovaná databáza (alebo databázy) je už plne závislé na zvolenej fyzickej architektúre systému.

2.2.2.1 SQL databázy

SQL databázy – nazývané aj databázy relačné – sú pomenované po dopytovacom jazyku *SQL* (*Structured Query Language*). Rôzne firmy a ich proprietárne databázové systémy disponujú rôznymi variantami jazyka *SQL* (*MySQL* vs. *MSSQL*) – vo väčšine prípadov sa však jedná iba o minimálne odchýlky.

SQL databázy ukladajú dáta v tabuľkách s pevne definovanou stavbou – táto prísne dodržiavaná štrukturálna vlastnosť so sebou nesie výhody aj nevýhody [110]. Relačné databázové systémy sú dlhodobo najvyužívanejším formátom perzistentného úložiska – aj keď sa v poslednej dobe pre špecializované prípady rozširuje vplyv a popularita *NoSQL* databáz, pre business-orientované systémy konvenčného rozsahu sú vďaka svojim benefitom vhodnou voľbou.

[111] Najvýznamnejším znakom relačných dátových úložísk je ACID teoréma:

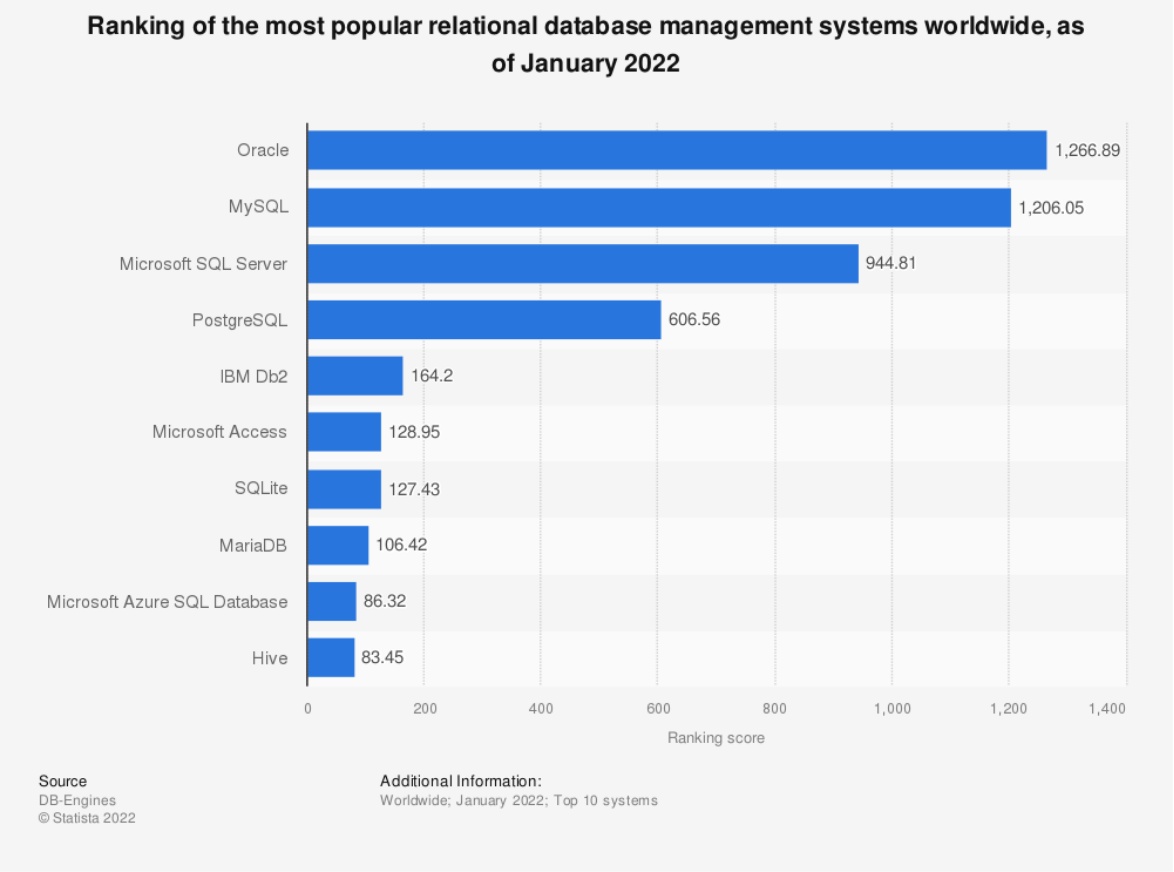
- **Atomickosť (Atomicity - A):**
  Databázová operácia je ďalej nedeliteľná – atomická. Všetky zmeny v dátach sú vykonávané tak, ako keby sa jednalo o jedinú operáciu – t.j. sú buď vykonané všetky bez výnimky, alebo žiadna.
- **Konzistencia (Consistency – C):**
  Dáta sú vždy v konzistentnom stave v bode pred začatím a po ukončení transakcie, pričom transakcia môže byť vnímaná ako prevod dát z jedného konzistentného stavu na druhý.
- **Izolovanosť (Isolation - I):**
  Okamžitý stav transakcie je pre ostatné transakcie neviditeľný – v dôsledku toho sa súbežne vykonávané transakcie zvonku javia, ako keby boli vykonávané sériovo. Pokiaľ dôjde k nutnosti zrušenia zmien vykonaných transakciou, musia byť reťazovo zvrátené aj zmeny vykonané transakciami touto zmenou zasiahnutými.

- **Trvalosť (Durability - D):**

  Po úspešnom dokončení transakcie sú zmeny v dátach permanentne uložené v databáze, kde v tomto stave pretrvajú a nemalo by dochádzať k ich spontánnej zmene – a to ani v prípade systémového zlyhania.

Relačné databázy okrem benefitov vyplývajúcich z dodržiavania *ACID* vlastností disponujú aj určitými charakteristickými limitáciami – najmä problematickou údržbou, implementačnou cenou, náročnosťou na kapacity fyzických ukladacích médií a nedostatočnou škálovateľnosťou [110]. Väčšina týchto limitácií (výmenou za istoty zaručované *ACID* vlastnosťami) určitým spôsobom adresujú *NoSQL* databázy [viď. kapitola *2.2.2.2 - „NoSQL databázy“*].

Typickými zástupcami skupiny relačných databázových riešení sú *Oracle DB*, *MySQL*, *PostgreSQL* a *Microsoft SQL Server* (použitý v rámci implementačnej zložky praktickej časti tejto práce).

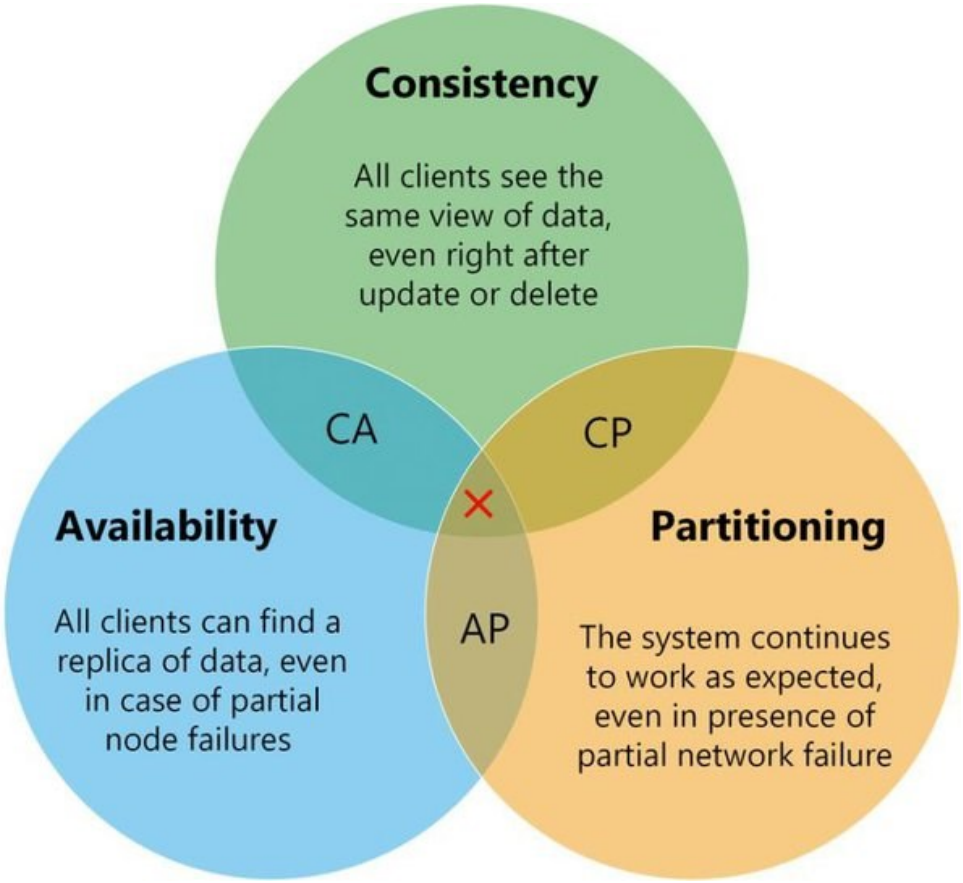


Obrázok 11. Porovnanie popularity najrozšírenejších relačných databázových systémov, prieskum zo začiatku roku 2022 [112]

2.2.2.2 NoSQL databázy

NoSQL (*Not only SQL*) databázy sú formou perzistentného dátového úložiska nezávislého na relačnom modeli – nemajú štruktúrované schéma, sú navrhnuté pre prevádzku vo veľkých databázových zhlukoch s pokročilými možnosťami dátovej distribúcie [113]. Nasadenie špecializovanej *NoSQL* databázy vhodnej štruktúry môže priniesť znateľné benefity najmä v prípadoch narážajúcich na obmedzenia úložísk relačného typu – teda napríklad pri spracovaní mimoriadne veľkého množstva dát (tzv. *Big Data*) alebo nutnosti popisovať / udržiavať objekty variabilnej štruktúry.

V kontexte *NoSQL* databáz existuje aj ekvivalent *ACID* teorémy popisujúcej špecifické vlastnosti distribuovaných *NoSQL* systémov – túto teorému nazývame *CAP* [113]. Na rozdiel od teorémy *ACID* (ktorá platnosť všetkých vlastností v rámci relačnej databázy zaručuje), *CAP* teoréma iba popisuje vlastnosti distribuovaného databázového úložiskového systému, pričom samotná databázová implementácia musí voliť vhodný kompromis medzi dostupnosťou a konzistenciou (čím vyššia dostupnosť tým nižšia konzistencia a naopak).



Obrázok 12. Grafické znázornenie CAP teorému, jeho vlastností a kompromisov, ktoré popisuje [114]

[114] CAP teoréma je definovaná tromi hlavnými zložkami:

- **Konzistencia (Consistency – C):**
  Konzistencia v rámci teorémy *CAP* popisuje schopnosť systému vždy vracať správne dáta – t.j. zaručiť, že po ľubovoľnej aktualizácii budú všetkým čítajúcim entitám poskytnuté rovnaké informácie.
- **Dostupnosť (Availability – A):**
  Dostupnosť stanovuje permanentnú dostupnosť dát, resp. schopnosť databázového systému obslúžiť všetky požiadavky bez ohľadu na okamžité vyťaženie systému.
- **Odolnosť voči rozpadu siete (Partition Tolerance - P):**
  Odolnosť voči rozpadu siete značí schopnosť databázy plniť svoju úlohu za predpokladu, že došlo k parciálnemu rozpadu siete a / alebo izolácii jedného či viacerých serverov.

Databázové riešenia založené na *NoSQL* princípe mnohokrát adresujú problémy či scenáre, v ktorých konvenčné relačné prístupy narážajú na svoje limitácie. Medzi ich hlavné výhody v porovnaní s relačnými databázami patrí najmä cena - Štrukturálna voľnosť v drvivej väčšine *NoSQL* riešení spôsobuje, že zmeny vo formáte dát mnohokrát nemajú dopad na spôsob, akým sú ukladané a / alebo spracovávané, čo výrazne pomáha znižovať mieru réžie a potrebný vývojový čas. Tento faktor zvykne byť tým významnejší, čím väčší je rozsah databázového systému. Ďalším významným kladom je horizontálna škálovateľnosť (t.j. rozloženie databázového riešenia na viacero strojov). To umožňuje efektívnejšie vyťaženie hardware, rovnomernejšie využitie úložiskového priestoru, masívny nárast možností paralelného spracovania a zvýšenie priepustnosti dátových operácií [113, 114].

[115] NoSQL databázy je možné deliť podľa spôsobu ukladania dát na rôzne podtypy:

- **Kľúč – Hodnota:** *Redis, Memcached*
- **Dokumentové:** *MongoDB, Couchbase, Firebase Realtime DB*
- **Stĺpcovo orientované:** *Cassandra, HBase, Hypertable*
- **Grafové:** *Neo4J, ArangoDB, OrientDB*

Podobne ako každá technológia, aj *NoSQL* databázy majú svoje nedostatky – v tomto prípade je však kvôli vyššie zmienenému deleniu do viacerých výrazne odlišných podkategórií s rozdielnym zameraním, štruktúrou a modelom použitia pomerne náročné určiť univerzálne platné negatíva.

Všeobecne sa dá povedať, že väčšina problémov plynie z nedostatočnej štandardizácie *NoSQL* technológií, náročnosti zachovávania konzistencie a synchronizácie dát, nedostatku užívateľskej / vývojárskej podpory a prípadného chybného nasadenia takého typu databázy, ktorá je s povahou adresovaného problému nekompatibilná a nevedie k riešeniu považovanému za kvalitatívne uspokojivé.

3 NÁVRHOVÉ A ARCHITEKTONICKÉ VZORY

Návrhové vzory sú odporúčané postupy riešenia často sa vyskytujúcich úloh. Predstavujú osvedčený mechanizmus či šablónu, ktorá umožňuje spoľahlivo, jednoducho a optimálne riešiť určitý typ vývojového problému, pričom zvyčajne zohľadňujú rozšíriteľnosť či modifikovateľnosť – ich nasadenie v rámci aplikácie vedie k zníženiu chybovosti a zlepšeniu dlhodobej údržby a prehľadnosti kódu aplikácie. Návrhové vzory sú prostriedkom, ktorý zaobaľuje časom overené úspešné riešenia identifikované predchádzajúcimi návrhármi a vývojármi a poskytuje tieto znalosti vývojárom riešiacim analogické problémy [116] – ide teda o súbor statických a dynamických štruktúr, ktoré vznikli ako riešenia typických problémov konštrukcie aplikácií v určitej doméne (ale bez viazanosti na konkrétny programovací jazyk).

Návrhové vzory sú koncipované genericky – nie sú univerzálnymi riešeniami konkrétnych systémov. Pre plné zakomponovanie návrhového vzoru pre potreby aplikácie je stále nutné mapovanie návrhového vzoru na štruktúry, ktoré majú tento vzor implementovať - vrátane prípadných modifikácií či kompromisov vychádzajúcich z povahy súčasnej stavby aplikácie a ďalších obmedzení [117].

Architektonické vzory sú následne veľmi podobné návrhovým vzorom – nezameriavajú sa však na šablóny odporúčaných riešení typických problémov, ale na popis štruktúry a logického členenia systému ako celku, nielen vybranej množiny subsystémov. Architektonické vzory definujú spôsob prevádzky vzájomnej komunikácie medzi nimi definovanými funkčnými časťami. Umožňujú členenie aplikácie s ohľadom na opakovanú použiteľnosť, pričom podľa potrieb konkrétneho riešenia je možné architektonické vzory v rozumnej miere kombinovať [117].

[118] Medzi často používané architektonické vzory radíme napr. :

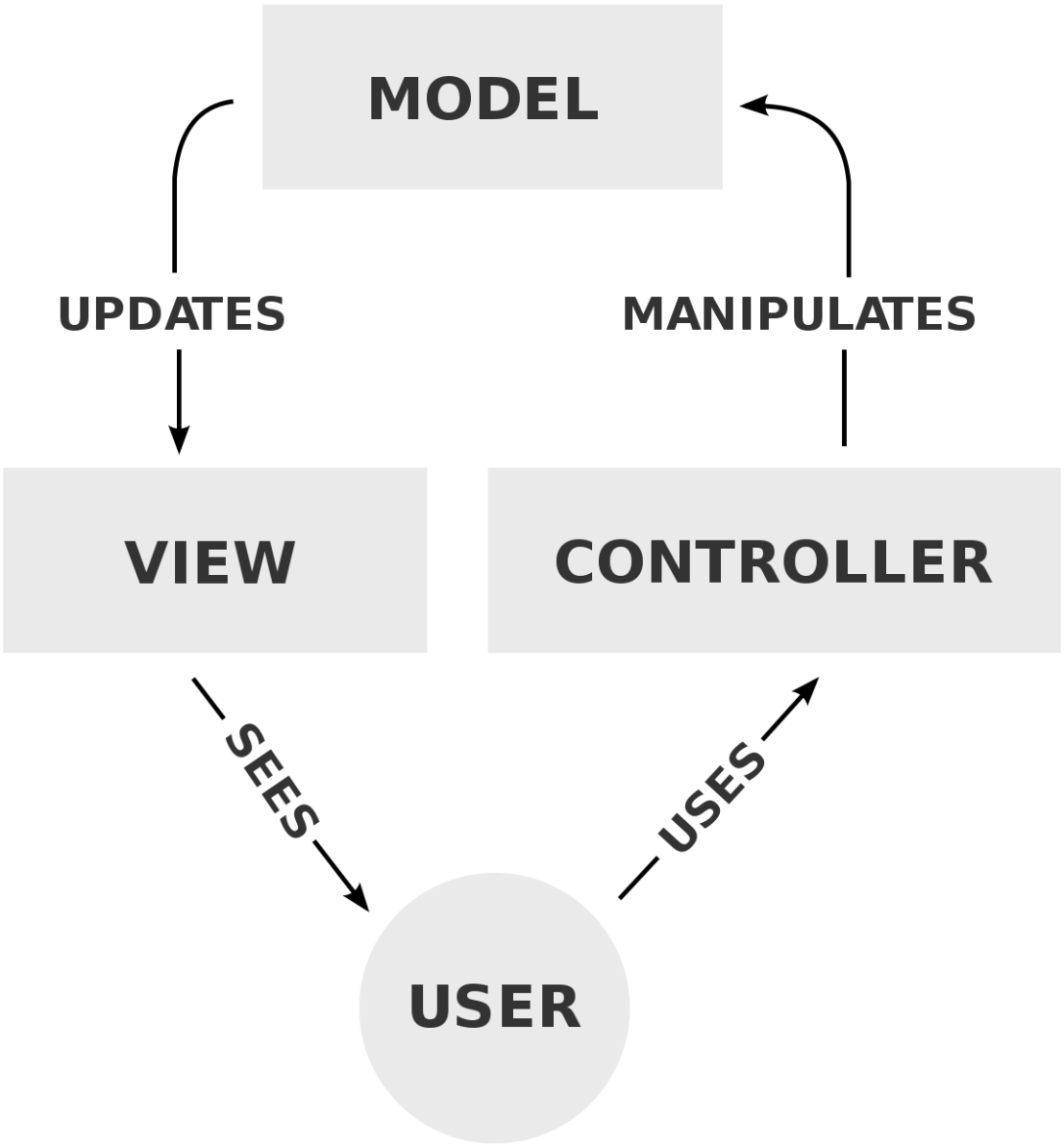
- **Distribuovaná architektúra:**
  Distribuovaná architektúra popisuje software tvorený oddelenými, vzájomne komunikujúcimi časťami. Využíva sa pri paralelnom spracovaní, kedy jednotlivé časti systému bežia v rámci rôznych procesov, potenciálne na rôznych procesoroch.
- **Vrstvená architektúra:**
  Často nazývaná aj N-vrstvová architektúra, rozdeľuje systém do niekoľkých vrstiev na základe klasifikácie častí software na základe ich zodpovedností [viď. kapitola *3.2 – N-vrstvová architektúra*].

- **MVC:**
  Architektúra typu *Model – View – Controller*, rozdeľuje aplikácie na časti reprezentujúce vizualizačnú logiku, business logiku a kontrolér sprostredkúvajúci ich interakcie.
- **Blackboard:**
  Architektonický vzor *Blackboard* popisuje komunikáciu častí aplikácie (nezávislých kooperujúcich prvkov) prostredníctvom zdieľaného média – tzv. tabule.

Konkrétne potreby vývoja nevynucujú ostať iba pri jednom architektonickom vzore – pokiaľ kontext či povaha software volá po použití viacerých, je ich kombinácia (za zachovania vlastností ktoré ich definujú) vhodná, ba priam odporúčaná.

3.1 Model – View – Controller

Neustále sa zvyšujúca potreba flexibility softwaru kladie zvýšené nároky na náročnosť vývoja rozhrania určenému ku komunikácii *Frontend* a *Backend* zložky webovej aplikácie – bez ucelenej architektúry by sa mohlo stať nevyhnutným pri každej zmene business logiky softwaru upraviť ako *Backend*, tak aj *Frontend* aplikácie. Modifikácia a zásahy do implementácie na viacerých miestach (najmä u väčších softwarových ekosystémov) nezriedka vedú k ťažko spozorovateľným chybám [119]. Architektúra *Model – View - Controller* sa snaží adresovať tento problém oddelením dátovej, vizualizačnej a business logiky softwaru [120] – umožňuje separátny vývoj jednotlivých funkčných celkov aplikácie a pri dostatočnej granulite umocňuje ich opakovanú použiteľnosť naprieč celým softwarovým riešením. Návrhový vzor *Model – View - Controller* sa prvýkrát aplikoval v objektovo orientovanom jazyku *Smalltalk* v roku 1980 [121], ale navrhnutý a predstavený bol ešte skôr – v roku 1979 na Univerzite v Oslo. Postupným nasádzaním a voľnou adaptáciou do povedomia programátorov sa *Model – View - Controller*, s ohľadom na jeho mimoriadne dobrú využiteľnosť pri tvorbe aplikácií využívajúcich Internet ako svoju hlavnú platformu stal v súčasnosti najrozšírenejším architektonickým vzorom pre vývoj software orientovaného do webového prostredia.



Obrázok 13. Všeobecný interakčný model realizovaný implementáciou architektonického vzoru Model – View – Controller[122]

3.1.1 Model

Model je ústrednou časťou architektonického vzoru *MVC*. Všeobecne môžeme povedať, že model obsahuje ako predpisy formátov objektov s ktorými aplikácia pracuje (v rámci *OOP*), tak jadrovú funkcionalitu aplikácie - z toho dôvodu je nutné rozlišovať, či sa jedná o *Model* dátovej povahy, alebo o model udržiavajúci logiku aplikácie [116, 121, 123, 124].

- **Dátové modely:**

  Dátový *Model* je abstraktnou reprezentáciou dát v počítačovom systéme. V rámci *OOP* opisujú prvky alebo javy reálneho sveta tak, aby s nimi bolo možné manipulovať v rámci aplikačnej logiky – zjednodušene povedané, ide o súhrn dát popisujúcich skutočný svet spolu s metódami nutnými na ich spracovanie.

  Dátové *Modely* udržiavajú stav aplikácie a v ideálnom prípade by nemali vedieť vôbec nič o spôsobe, akým s nimi aplikácia manipuluje. Komplexnejšie aplikácie usilujúce o ešte kvalitnejšie rozdelenie zodpovedností jednotlivých komponent programu môžu ďalej deliť dátové modely na rôzne špecializované podtypy:

  - **View modely [119]:**

    Vypožičané z architektúry *MVVM* (*Model – View – View Model*), *View Model* nachádzajú svoje uplatnenie aj pri vývoji webových aplikácií z nasledujúcich dôvodov:

    - Zasielanie kompletných dát doménového modelu na *Frontend* nie je vždy žiadúce. Mapovaním do *View Model*-u s ktorým pracuje *Frontend* sa môžeme vyhnúť potenciálnym bezpečnostným rizikám z nepozornosti a zároveň zmenšujeme objem dátového toku.
    - Pokiaľ *Controller* aj *Frontend* pracujú so zhodným *View Model* prvkom, validačná logika môže byť zdieľaná (okrem špeciálnych prípadov odhaliteľných až v hlbších vrstvách programu) a obmedzená výhradne na *View Model*, čo znižuje neprehľadnosť kódu vzniknuvšiu následnom miešania validačnej a business logiky na jedinú modelovú triedu.

  View Model určený ku komunikácii medzi *Frontend* a *Backend* časťou aplikácie (v tomto prípade bez ohľadu na to, či obsahuje alebo neobsahuje nejakú logiku určenú výhradne pre vizualizačné účely) sa môže nazývať aj *DTO* (*Data Transfer Object*).

  - **Databázové modely [125]:**

    Databázové modely (alebo entity, či doménové - dátové *Modely*), ako názov napovedá, reprezentujú objektovú reprezentáciu dát uložených v rámci databázového riešenia aplikácie. Konkrétny formát informácií ukladaných do databázy je pochopiteľne závislý na zvolenom type databáze [viď. kapitola *2.2.2 - „Databáza"*], pričom populárne webové frameworky zvyknú podporovať viacero rôznych typov databáz.

Pokiaľ je používaná databáza založená na relačnom princípe, je nutné pre zavedenie databázových *Modelov* nasadenie *ORM* mechanizmu [viď. kapitola *3.2.3.1 - „ORM"*],. Tento typ *Modelov* by mal čo najdôvernejšie kopírovať dáta obsiahnuté v databázových štruktúrach, prípadne súvisiace obmedzenia kladené na jednotlivé položky či definované vzťahy medzi nimi (primárne / cudzie kľúče a pod.).

- **Doménové modely [126]:**

  Nazývané aj *Modely* business logiky, sú triedou modelov zastrešujúcich všeobecne najväčšiu zložku webovej aplikácie. Doménové *Modely* využívajú iné *Modely* dátového typu s ktorými manipulujú takým spôsobom, aby zaistili naplnenie podstaty business procesov ktoré má aplikácia za úlohu riešiť. Pre rôzne vrstvy *Modelov* business logiky môžeme nájsť v rámci N-vrstvovej architektúry rôzne názvy (aplikačné služby, služby, repozitáre...). Doménové *Modely* môžu interagovať s inými doménovými *Modelmi* prostredníctvom programátorom definovaného rozhrania - Podobne ako u dátových *Modelov*, ani u *Modelov* business logiky správne nemalo existovať prílišné povedomie o kontexte, z ktorého sú ich možnosti používané.

3.1.2 View

View je súhrnom častí aplikácie zodpovedných za prezentovanie grafického užívateľského rozhrania, zobrazovanie informácií užívateľovi a definovania aplikačnej logiky v rámci *Frontend*-u aplikácie, možností interakcie užívateľa s podkladovými *Modelmi*, a previazania týchto logických celkov na mechanizmus tvorby požiadaviek odosielaných na server - a následného spracovania či vizualizácie odpovedí zasielaných serverom [116, 127]. *View* je najčastejšie (najmä v kontexte webového prostredia) vnímané ako grafické užívateľské rozhranie, avšak miera jeho vizuálnej kvality je otázna – môže ísť taktiež o *View* typu *CLI*, alebo o *View* vo forme *API* určeného napr. pre konzumáciu strojovými klientmi.

Štrukturálne sa *View* skladá zo súborov písaných prostredníctvom *DSL* (*Domain Specific Languages*), ktoré sú interpretované prostredníctvom zobrazovacieho engine do formátu vykresliteľného internetovým prehliadačom (ako napr. formát *HTML*). Veľké množstvo webových frameworkov k možnosti efektívnejšieho rozdelenia logiky v rámci *View* (napr. prostredníctvom rozčlenenia do komponent) využíva vlastné šablónovacie systémy.

Webové šablónovacie systémy umožňujú programový zápis *View* ako kombinácie statického obsahu (*HTML, CSS*) a dynamického obsahu [128], za ktorého tvorbu zodpovedá *Backend* - štandardne na základe podkladového *Modelu* (alebo *View Model*-u), ktorý bol šablóne poskytnutý pri generovaní zobrazenia, pričom komunikácia *View* s hlbšími časťami aplikácie prebieha v čistej *MVC* architektúre prostredníctvom zasielania *Modelov* výhradne cez prvky typu *Controller*. Čo sa týka *Modelov* samotných, *View* pristupuje k ich štruktúre, ktorú v rámci implementácie formulárov dokáže modifikovať na elementárnej úrovni a prípadne pristupovať k ďalším funkcionalitám relevantným pre interakciu užívateľa s podkladovými *Modelmi* (ako napr. validačné obmedzenia). Implementácia rozsiahlej business logiky v rámci *View* sa neodporúča.

Výhodou odčlenenia vizualizačnej logiky prostredníctvom *View* je, že pokiaľ nedošlo k výraznej zmene podkladových *Modelov*, tak je možné prakticky kompletne zmeniť dizajn aplikácie bez nutnosti realizovať akékoľvek zmeny v business logike. Rozšírené možnosti abstrakcie nadobudnuté prostredníctvom oddelenia zobrazenia od business logiky a vhodne implementovaného členenia komponent sprístupňujú ďalšie výhody – ako napr. zdieľanie rozloženia štruktúry stránok (*Layout*) medzi jednotlivými zobrazeniami či opakovanú použiteľnosť *Modelov* v rámci rozdielnych *Views* [116, 123].

3.1.3 Controller

Controller v *MVC* architektúre zaručuje komunikačný kanál medzi užívateľom pracujúcim s aplikáciou prostredníctvom *View* a business logikou aplikácie zastúpenou množinou dátových či aplikačných *Modelov* [116, 127]. Zaručenie obšírnej logiky spracovania dát nie je úlohou *Controller*-u – je vhodné tieto úlohy prenechať príslušným špecializovaným *Modelom*, s ktorými *Controller* je schopný manipulovať napríklad prostredníctvom referencie obdržanej skrz mechanizmus *Dependency Injection* [viď. kapitola *3.3.1 – "Dependency Injection"*].

Zodpovednosti komponenty *Controller* zahŕňajú prijatie a mapovanie dát ktoré zasiela *Frontend* aplikácie a zaistenie dokumentácie rozhrania, ktorým disponuje *Backend* (*API Contract*) vrátane stanovenia formátov a / alebo *HTTP* kódov možných odpovedí serveru, realizácia validácie nad prijatými požiadavkami, kontroly prostredníctvom autorizačných mechanizmov aplikácie a formulácia odpovedí odosielaných klientom prostredníctvom modelov [124].

Controller je jedinou zložkou aplikácie, ktorá má aktívne povedomie a možnosť priamej manipulácie s oboma zvyšnými štrukturálnymi celkami – ako s *Views*, tak aj s *Modelmi*.

Rozdelenie webovej aplikácie na *Frontend* a *Backend* značí, že *Controller* zastáva ďalšiu dôležitú úlohu – sú časťou aplikácie, ktorá spravuje a poskytuje (či už priamo alebo prostredníctvom vhodného middleware) informácie o metadátach a prostredí systému, na ktorom *Backend* beží a ktoré by mohli mať vplyv na špecifiká či formát požiadaviek / odpovedí – medzi tieto údaje patria napr. informácie o operačnom systéme, verzii použitého frameworku či podporované formáty prenosu dát. Ako vstupný bod do funkcionalít *Backend*-u aplikácie sú taktiež lokalitou, v ktorej sa prejavuje smerovanie (tzv. *Routing*) – t.j. mapovanie kombinácie *URL* identifikátoru a *HTTP* metódy na určité vstupné body konkrétneho C*ontroller*-u, ktorý bude ďalej zodpovedný za obslúženie tejto požiadavky [124].

3.2 N-vrstvová architektúra

Vrstvenie softwaru je jedným z najvhodnejších spôsobov na rozdelenie komplikovaného alebo funkcionálne bohatého systému. Myšlienka vrstvenia je prítomná v ktorejkoľvek dostatočne komplexnej oblasti počítačového sveta – či už sa jedná o zvyšujúcu sa mieru abstrakcie smerom od fyzikálnych javov na hardwarovej úrovni až k rozhraniam sprostredkovaným operačným systémom, alebo o rovnaký koncept aplikovaný na modely sieťovej komunikácie (*OSI / ISO*), vrstvenie je konceptom dobre známym a dlhodobo používaným.

Vrstvenie v rámci architektúry webových aplikácií (najmä tých, kde si obšírnosť business logiky žiada dodatočné členenie) je mimoriadne populárnym prístupom. Nasledujúce podkapitoly sa budú venovať základnému modelu – tzv. *3-Tier* (trojvrstvovej) architektúre, pričom v realite je podľa potreby dodatočného rozdelenia funkcionálnych celkov aplikácie možné použiť vrstiev samozrejme viac – všeobecne hovoríme o N-vrstvovej architektúre [119, 129]. U N-vrstvovej architektúry je základný komunikačný mechanizmus založený na princípe, že vyššie vrstvy môžu volať funkcionalitu a služby vrstiev nižších (a obdržať odpovede a výsledky svojich volaní), avšak nemali by mať povedomie o existencii prípadných vrstiev spočívajúcich nad nimi.

[130] Rozlišuje sa architektúra s otvorenými a uzatvorenými vrstvami:

- **Otvorené vrstvy**:
  Každá vrstva môže volať ktorúkoľvek z vrstiev nachádzajúcich sa hierarchicky pod ňou.
- **Uzatvorené vrstvy:**
  Každá vrstva môže volať iba jednu pevne stanovenú, „podriadenú“ nižšiu vrstvu.

Praktická časť tejto práce z uvedených podtypov N-vrstvovej architektúry používa formát s otvoreným typom vrstiev.

Výhody N-vrstvovej architektúry:

- Dobrá prenositeľnosť a kompatibilita s cloudovým prostredím,
- Jednoduchá učebná krivka,
- Lepšia adaptácia a kompatibilita s heterogénnymi prostrediami (napr. rôzne OS).

Nevýhody N-vrstvovej architektúry:

- Umožňuje vznik vrstvy, ktorej funkčný prínos je minimálny, pričom jej jediným efektom je zhoršenie celkovej latencie systému,
- Monolitický design aplikácie znemožňuje nezávislé nasadenie nových prvkov,
- Problematické zabezpečenie zabezpečenia v prípade rozsiahleho systému.

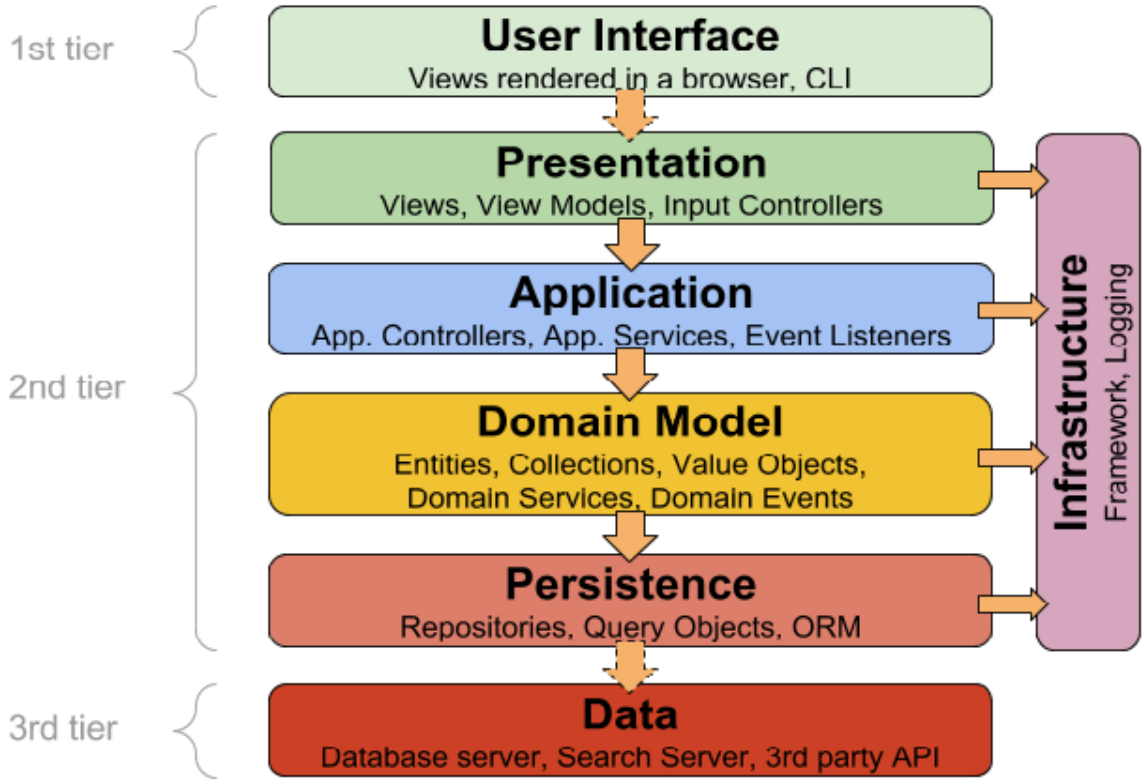
Anglicky písaná literatúra občas v kontexte N-vrstvovej architektúry uvádza rôzny formát slova vrstva – niekedy je význam vyjadrený prostredníctvom slova *Tier*, inokedy *Layer*, pričom nie vždy sa jedná o pojmy zameniteľné – v prípade, že literatúra pracuje s oboma pojmami, existujú medzi nimi určité rozdiely [119, 130, 131]:

- **Tier:**
  Fyzicky separovaná vrstva, nasadená na rozdielnom stroji ako ostatné *Tiers*. S ostatnými *Tiers* komunikuje buď priamym sieťovým spojením alebo prostredníctvom asynchrónnych správ. Fyzické rozdelenie vrstiev zlepšuje škálovateľnosť, ale aplikujú sa naň typické nevýhody distribuovaného spracovania – ako napr. nutnosť synchronizácie či dodatočné nároky na objem a latenciu komunikácie.

- **Layer:**
  Logická vrstva, *Layers* sa od seba odlišujú na základe zodpovedností a obsiahnutej funkcionality, rovnako ako závislostí, ktoré potrebujú ku svojej správnej činnosti. Existuje možnosť v prípade potreby umiestniť viac logických vrstiev (*Layers*) na každú fyzickú vrstvu (*Tier*).

Aplikácia vytvorená v praktickej časti tejto práce vníma N-vrstvovú architektúru adresová v zmysle slova *Layer* – teda logickej vrstvy, keďže sa predpokladá, že vzhľadom na rozsah a relatívnu výpočtovú nenáročnosť bude *Backend* systému sprostredkúvaný jediným fyzickým serverom.

Hlavnou výhodou rozdeľovania logiky do vrstiev je motivácia nenáročnou udržateľnosťou a rozšíriteľnosťou vyvíjaného systému. Mimoriadne známou mutáciou N-vrstvovej architektúry je práve trojvrstvová architektúra disponujúca tromi vrstvami – prezentačnou vrstvou (*Presentation Layer*), vrstvou business logiky (*Business Logic Layer*) a vrstvou dátového prístupu (*Data Access Layer*) [132].



Obrázok 14. Viacvrstvová architektúra popisujúca jedno z populárnych logických členení webovej aplikácie (vrstvy vnímané v zmysle Layers) [133]

3.2.1 Prezentačná vrstva

Prezentačná logická vrstva sa zameriava na vizualizáciu a sprostredkovanie interakcií medzi užívateľom a software. Trojvrstvová architektúra vo svojej základnej forme nekladie nároky na skutočnú podobu prezentačnej vrstvy – môže sa rovnako jednať o jednoduché príkazové rozhranie (*CLI*), ako o vizuálne bohaté *GUI* desktopovej / mobilnej / webovej aplikácie. Okrem samotnej vizualizácie je prezentačná vrstva zodpovedná za prijímanie vstupov užívateľa, ich vyhodnocovanie a spätnú väzbu [134].

Webové aplikácie vnímajú prezentačnú vrstvu najčastejšie analogicky k *Frontend* časti softwarového systému – t.j. ako časť aplikácie prístupnú z webového prehliadača klienta, zvyčajne založenú na technológiách *HTML*, *CSS* a *JavaScript* (alebo ich mutáciách). Ostatné typy aplikácií prezentačnú vrstvu ponímajú analogicky, ale pochopiteľne môže byť definovaná inými typmi technológií [135] – napr. *XML* rozloženie u programov vytvorených prostredníctvom technológie *WPF* (*Windows Presentation Foundation*).

Prezentačná vrstva je (aspoň u vývoja smerovaného na webové prostredie) vrstvou, u ktorej je najčastejšie zameniteľné, či sa jedná o vrstvu fyzickú alebo logickú (*Tier* vs. *Layer*) – toto vyplýva z skutočnosti, že typickými základnými štrukturálnymi časťami webovej aplikácie sú logické celky dané architektúrou Klient - Server, pričom tieto sú len výnimočne zamýšľané pre použitie z jediného fyzického zariadenia.

3.2.2 Vrstva business logiky

Vrstva business logiky (občas nazývaná aj vrstvou doménovou alebo aplikačnou) má za úlohu obsiahnuť jadrovú funkcionalitu systému – t.j. pravidlá, ktoré stanovujú akým spôsobom systém plní úlohy, na ktorých riešenie bol zostrojený. Zároveň táto vrstva zodpovedá za podružné funkcie slúžiace k optimálnemu behu či podporným procesom (ako je napr. infraštruktúra, logovanie, či autorizačné procesy) [119]. Pokiaľ je žiadúce dodatočné rozdelenie zodpovedností v rámci doménovej vrstvy, je možné a vhodné realizovať jej rozpad na ďalšie, vzájomne komunikujúce pod-vrstvy, v ideálnom prípade prepojené formou stromu závislostí [viď. kapitola *3.3.1 – "Dependency Injection"*].

Doménová vrstva môže zahŕňať výpočty na základe požiadaviek odosielaných prezentačnou vrstvou so zohľadnením povahy už uložených / evidovaných dát, prípadné hĺbkové validácie v kontexte rozšírenejšieho povedomia o celkovom stave systému, než aký zvykne byť prístupný vrstve prezentačnej [119, 135].

Vrstva business logiky zodpovedá za určovanie a exekúciu postupu spracovania na základe obdržaných dát a príkazov. Webové aplikácie vrstvu business logiky interpretujú ako *Backend* (resp. *Backend* s výnimkou práce s databázou v prípade súčasnej existencie vrstvy zabezpečujúcej dátový prístup).

3.2.3 Vrstva prístupu k dátam

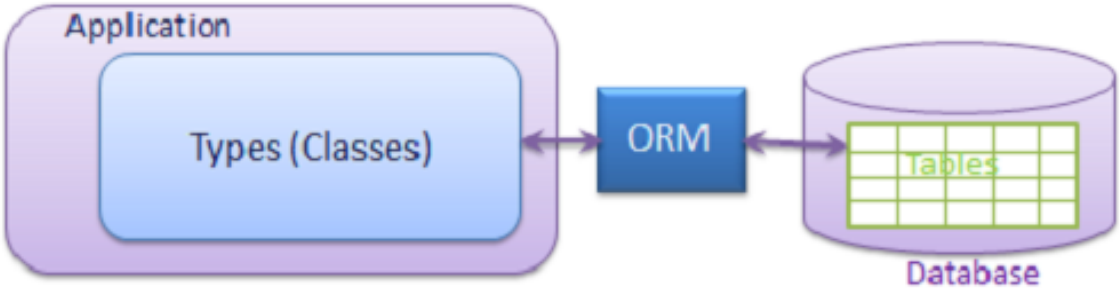
Vrstva dátového prístupu v trojvrstvovej architektúre (nazývaná aj perzistentná vrstva či úložisková vrstva) poskytuje centralizovaný prístup k dátam, s ktorými má aplikácia možnosť pracovať – v prípade konvenčnej webovej aplikácie teda sprostredkúva prístup k perzistentnému úložisku [131]. Zodpovednosti úložiskovej vrstvy nekončia správou databázového systému – môžu do nej spadať aj komunikačné systémy, správcovia transakcií, pravidelne spúšťané rutiny alebo ďalšie balíčky manipulujúce s perzistentnými dátami bez nutnosti využitia bežného komunikačného kanálu prechádzajúceho aplikáciou.

Prístup k tejto vrstve N-vrstvového modelu je umožnený vrstvám business logiky umiestneným hierarchicky vyššie – perzistentná vrstva poskytuje dáta pre operácie, ktoré v nich prebiehajú a zároveň sprostredkúva služby pre aktualizáciu uložených dát vplyvom výsledkov či zmien vzniknuvších v rámci týchto operácií [135]. V rámci aplikácie založenej na trojvrstvovom modeli by vrstva prístupu k dátam nemala priamo komunikovať s vrstvou prezentačnou – prostredníkom v tejto komunikácii musí byť vždy vrstva aplikačná / doménová [119]. Ohľadom stanovenia rozdielu medzi vrstvou fyzickou a vrstvou logickou v tomto prípade nie je možné stanoviť pevné závery – mnoho webových aplikácií disponuje perzistentným úložiskom priamo na tom istom fyzickom zariadení ktoré prevádzkuje doménovú logiku, mnoho ďalších zas preferuje prístup, kedy je databáza vedená bokom od aplikačného *Backend*-u.

Spôsob akým aplikácia manipuluje s vrstvou dátového prístupu je veľmi dôležitý – pri nesprávnom prístupe k dátam alebo nízkej granulite požiadaviek na túto vrstvu existuje šanca, že výkon aplikácie pri operáciách s perzistentným úložiskom bude neuspokojivý.

3.2.3.1 ORM

Objektovo-relačné mapovanie (*ORM – Object - Relational Mapping*) je mechanizmus, ktorý poskytuje vývojárom potrebnú abstrakciu k mapovaniu aplikačného objektovo orientovaného kódu na databázové tabuľky relačného charakteru. *ORM* je mocným nástrojom kvôli jeho masívnemu dopadu na zníženie objemu napísaného kódu a s tým súvisiace zrýchlenie vývoja či zjednodušenie prepojenia aplikačnej logiky na databázové úložiská [136]. Zároveň umožňuje efektívnu manipuláciu s databázou aj menej skúseným vývojárom, ktorí nie sú príliš zruční v písaní a optimalizácii *SQL* kódu. Objekty (vrátane jednotlivých vlastností a obmedzení na nich kladených) ktoré prostredníctvom *ORM* mapujeme na relačné tabuľky by mali týmto tabuľkám čo možno najbližšie korešpondovať - pokiaľ to nie je možné, *ORM* frameworky zvyknú ponúkať možnosti nastavenia konkrétneho formátu mapovania medzi objektom a jeho relačným protipólom. Takéto objekty v kontexte využitia *ORM* nazývame *Entity* [125].



Obrázok 15. Princíp objektovo - relačného mapovania, schéma zachytáva komponent zodpovedný za mapovanie vo formáte čiernej skrinky [137]

ORM, podobne ako ďalšie technológie pridávajúce určitú mieru abstrakcie, uvádza taktiež potenciálne nevýhody – zďaleka najčastejším problémom je optimalizácia databázových otázok. Programátor pristupujúci k dátam prostredníctvom *ORM* si mnohokrát neuvedomí, akým spôsobom bude jeho otázka preložená do *SQL* formátu – vrátane toho, aká náročná daná otázka potenciálne môže byť [138]. Nepomer medzi zložitosťou niektorých štruktúr *OOP* a databázových aktivít tento problém ďalej prehlbuje – napr. načítanie dát v cykle z hľadiska *OOP* je jednoduchou a triviálnou operáciou – pri použití *ORM* už je nutné dbať skutočnosti, že každý prístup pre jediný prvok vytvára nové pripojenie do databázy (namiesto alternatívneho načítania všetkých požadovaných dát v rámci jednej databázovej otázky) už sa však môžeme pozerať na masívnu penalizáciu z hľadiska výkonu.

Vytvárané riešenia využívajúce *ORM* nie vždy majú možnosť byť testované na veľkom objeme dát – čo vedie k tomu, že sa problém podcenenej optimalizácie môže prejaviť až časom [138].

Backend praktickej časti práce nasádza *ORM* v podobe technológie *Entity Framework* od firmy *Microsoft*. Riešenie operuje nad relačnou databázou typu *SQL Server*.

3.2.3.2 Repozitár

Návrhový vzor repozitár (*Repository*) je pevne zviazaný s využitím v rámci vrstvy dátového prístupu. Repozitár je súhrnným označením pre množinu komponent (tried, objektov...) obaľujúcich logiku potrebnú pre ľubovoľný prístup k dátam a dôkladnejšie oddelenie infraštruktúry dátového prístupu od aplikačnej logiky vo vyšších vrstvách [139].

V kombinácii s *ORM* mapovaním a implementáciou podporných princípov (ako napr. *Unit of Work*) zaručuje jasné a jednosmerné rozdelenie zodpovedností v otázkach závislostí medzi pracovnou doménou kde prebieha spracovanie dát a oblasťou dátovej alokácie / mapovania na pomedzí *Backend*-u a databázového riešenia.

Unit of Work

Jednotka práce (*Unit of Work*) je v rámci návrhového vzoru repozitár spôsobom ako pri operáciách, ktoré manipulujú s dátami vo viacerých krokoch zaručíme konzistenciu nie nepodobnú princípu definovanému v rámci *ACID* princípov pre relačné databázy [viď. kapitola *2.2.2.1 – "SQL Databázy"*] [119].

Práca s databázou - v prípade viacerých operácií - vyžaduje udržiavanie povedomia o tom, ktoré entity / tabuľky / dáta boli zmenené – a či už tieto zmeny boli zapísané do perzistentného úložiska. Nežiadúcim správaním je modifikovať dáta po častiach v kombinácii s vykonaním zápisu po každej parciálnej modifikácii / vložení / zmazaní v rámci úložiska – jedným rizikom je opakovaná manipulácia s pomalým perzistentným úložiskom a teda k výraznému zvýšeniu latencie operácií. Druhý problém spočíva v prípade, ak bude postupnosť operácií z nejakého dôvodu prerušená – bez ohľadu na to, či pôjde o fyzické zlyhanie zariadenia alebo výnimku vyvolanú chybou programu. Popísaná situácia môže viesť k nekonzistencii dát v rámci databázy [119, 140] – a následným čítaniam týchto chybných dát v priebehu nasledujúcich operácií, s potenciálom silného lavínového efektu.

Princíp *Unit of Work* (resp. jeho konkrétna implementácia realizovaná buď prostredníctvom vhodného frameworku či vlastnoručne vytvoreného riešenia) sa zaručuje, že transakcia ovplyvňujúca úložisko je nad všetkými zasiahnutými dátami buď vykonaná kompletne, alebo vôbec [139, 140] – a teda aj v prípade chybových stavov úložisko obsahuje dáta, ktoré sú síce nemodifikované, ale zato neustále v konzistentnej forme.

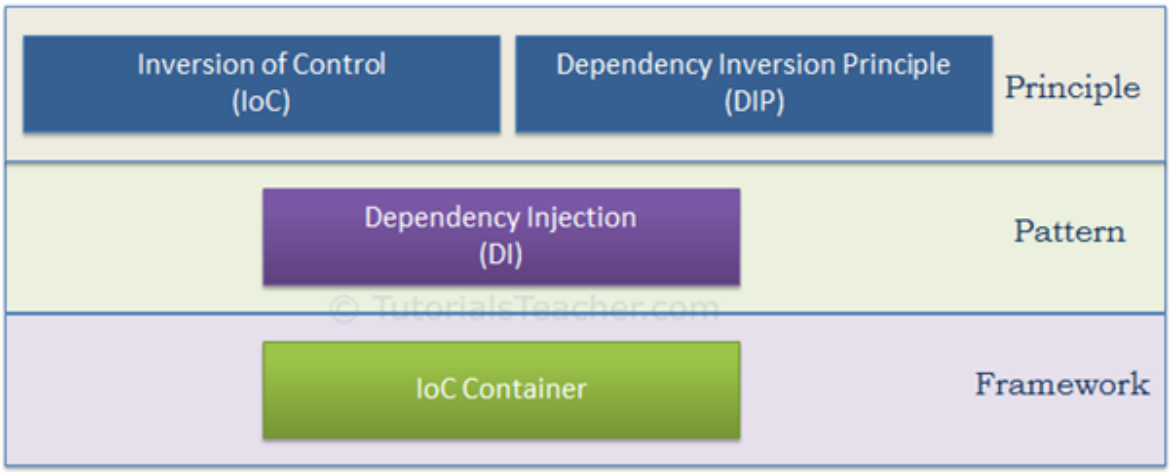
3.3 Inversion of Control

Dizajnový princíp *Inversion of Control* (*IoC*) sa zaoberá myšlienkou, že objekty a služby by nemali byť zodpovedné za závislosti (dependencies), ktoré ku svojej činnosti vyžadujú – táto zodpovednosť by mala byť odovzdané externej autorite, resp. mechanizmu, ktorý sa postará o poskytnutie všetkých potrebných závislostí do konkrétnych objektov, čím sníma nutnosť programátora vykonávať prácny refaktoring v prípade, že sa závislosti akokoľvek zmenia [141]. Konkrétna štruktúra prostredníctvom ktorej je myšlienka *IoC* implementovaná sa zvyčajne nazývaná kontajner – tento kontajner udržiava povedomie o jednotlivých komponentoch aplikácie, ktoré môžu byť závislosťami iných komponentov - ale taktiež môžu byť na ďalších komponentoch samy závislé. Spôsob interpretácie tohto stromu závislostí je pritom zodpovednosťou samotného kontajneru. *IoC* kontajnery často umožňujú mimo registrácie závislostí určiť aj ich životný cyklus – t.j. stanoviť, aký je vzťah a trvanlivosť jednotlivých závislostí [124].

3.3.1 Dependency Injection

Návrhový vzor *Dependency Injection* (*DI*) je implementáciou myšlienky *Inversion* of *Control* [119, 141]. Jedná sa o použitie *IoC* kontajneru konfigurovaného mimo jadro aplikačnej logiky (napr. pri spustení programu) k tomu, aby tento kontajner zastrešoval poskytovanie potrebných závislostí naprieč triedami aplikácie bez nutnosti ich explicitného (a potenciálne hierarchicky komplikovaného) vytvárania samotným vývojárom [141]. Aplikácia *DI* mechanizmu dosiahne ešte vyššej úrovne znovupoužiteľnosti a generiky v prípade, že moduly samotné nemajú k dispozícii (mimo špecifických výnimiek) konkrétne implementácie svojich závislostí – je žiadúce, aby spolu komunikovali prostredníctvom abstrakcií či rozhraní. Riešenia pre aplikáciu DI mechanizmov zvyčajne umožňujú programátorovi vhodným spôsobom (v závislosti od konkrétneho frameworku / technológie – anotácie, *XML* predpisy...) špecifikovať mapovanie konkrétnych tried na rozhrania, ktorými budú pri injektovaní reprezentované.

Uvedené akcie ďalej znižujú pravdepodobnosť nesprávnej manipulácie s objektmi závislostí a zvyšujú testovateľnosť vďaka možnosti jednoduchej zmeny mapovania v rámci konfigurácie *IoC* kontajneru [142].



Obrázok 16. Vzájomné súvislosti názvoslovia uvádzaného vo vzťahu k mechanizmu Dependency Injection [143]

[124, 144] Rozlišujeme tri základné typy Dependency Injection:

- **Constructor Injection**:
  Injektovanie závislosti do konštruktora vytváraného objektu, tento spôsob je najpoužívanejším a najjednoduchšie udržiavateľným.
- **Setter Injection**:
  Injektovanie hodnoty / objektu poskytované *IoC* kontajnerom do *setter*-u niektorej z vlastností objektu.
- **Method Injection**:
  Injektovanie hodnoty / objektu prostredníctvom jej dosadenia formou argumentu do niektorej z metód objektu.

4 ASP.NET CORE BLAZOR

ASP .NET Core Blazor (alebo skrátene *Blazor*) je mladý framework oficiálne sprístupnený v roku 2018. Zakladá sa na skĺbení funkcionality viacerých výnimočných a v rámci trhu dobre zaužívaných technológií. Hlavná sila frameworku *Blazor* spočíva vo využití jazyka *C#* a schopnostiach podkladovej implementácie *.NET Core*, šablónovacieho systému *Razor* a základoch prevzatých z populárneho frameworku pre tvorbu dynamických webových stránok *ASP .NET* [145, 146, 147].

4.1 Charakteristika

Blazor je webový *UI* framework určený k tvorbe klientskej časti aplikačnej logiky za využitia jazyka *C#*, čím sa odlišuje od prístupu drvivej väčšiny konkurenčných frameworkov na vývoj *Frontend*-u (keďže tieto najčastejšie volia prístup realizovať nejakú formu nadstavby nad jazykom *JavaScript*) [148].

[148, 149] Využitie frameworku Blazor prináša mnoho benefitov pre priebeh vývoja:

- **Systém Komponent:**
  Spôsob členenia logiky pre konštrukciu užívateľského rozhrania je inšpirovaný modernými frameworkami vývoja *Frontend*-u a umožňuje vytvárať zložitejšie celky tvorené elementárnymi, parametrizovanými komponentmi s využitím kombinácie jazykov *HTML*, *CSS* a *C#.*
- **JavaScript Interop:**
  Blazor implementuje sadu nástrojov a *API* známych pod názvom *JSInterop* určených na komunikáciu medzi kódom definovaným v jazyku *C#* a kódom v jazyku *JavaScript* a naopak – to znamená, že *Blazor* plne podporuje využitie bohatej škály existujúcich knižníc určených pre *JavaScript*.
- **Open-Source:**
  Blazor je vedený pod *Open-Source* licenciou, čo vedie k lepšiemu pochopeniu interných mechanizmov frameworku komunitou a aktívnejšiemu prístupu vývojárskej spoločnosti k prispievaniu do kódu s úmyslom ladenia chýb či optimalizácie.
  Základ frameworku spočíva v platforme *.NET*, ktorá je kompletne zdarma a bez nutnosti akýchkoľvek licenčných poplatkov za použitie, vrátane využitia komerčnej povahy [150].

- **Zdieľanie kódu:**
  Tvorba *Frontend*-u webovej aplikácie prostredníctvom jazyka *C#* umožňuje vďaka implementácii *.NET* štandardu odkazovať akékoľvek knižnice (či iné projekty), ktoré sú so zvolenou verziou tohto štandardu kompatibilné – to v podstate znamená, že väčšina kódu (mimo špecifické aplikačné rozhrania nepodporované webovým prehliadačom) môže byť zdieľaná medzi klientskou a serverovou časťou aplikácie, čo urýchľuje vývojový cyklus aplikácie, redukuje vývojárske chyby a zabraňuje zbytočnej duplicite kódu.

Užívateľské rozhranie tvorené v rámci frameworku *Blazor* využíva osvedčený šablónovací systém *Razor*, ktorý sprostredkúva funkcionalitu nie nepodobnú rôznym direktívam známym z konkurenčných frameworkov (za zmienku stojí napríklad možnosť realizovať v rámci šablóny dynamicky generovaného *HTML* dokumentu podmienky či cykly) [146]. Prostredníctvom kódu písaného v jazyku *C#* vkladaného priamo do *HTML* dokumentov vznikajú súbory obsahujúce predpis dynamickej webstránky s príponou *.razor* (alternatívne *.cshtml*) [151].

ASP.NET Core Blazor je vo svete technológií ešte stále pomerne novým úkazom – entuziazmus technologického giganta *Microsoft* do jeho ďalšieho vývoja ale prakticky garantuje ďalšie vylepšovanie a rozvoj nových funkcionalít v budúcich rokoch.

4.1.1 Jazyk C#

C# je moderný, kompilovaný, objektovo-orientovaný a typovo bezpečný programovací jazyk s koreňmi v rodine programovacích jazykov vychádzajúcich z jazyka *C* (podobne ako *C++* či *Java*), s ktorými nesie významné syntaktické podobnosti. Softwarový design v kontexte jazyka *C#* sa zameriava na vytváranie zapuzdrených a samo-popisných komponent s presne definovanou a pevne stanovenou funkcionalitou s cieľom odhaľovať ich ďalšiemu použitiu ako programový model založený na vlastnostiach (tzv. *Properties*), metódach (*Methods*) a udalostiach (*Events*) [152].

Cieľom jazyka *C#* je maximalizácia produktivity programátora – tento účel sa snaží dosiahnuť prostredníctvom vhodnej rovnováhy syntaktickej jednoduchosti, expresívnosti a výkonu. Rôzna literatúra uvádza rôzne vlastnosti jazyka *C#* ako jeho kľúčové charakterové črty – Najvýznamnejší dopad na metodiku vývoja software majú vlastnosti, ktorým sú venované nasledujúce podkapitoly.

4.1.1.1 Objektová orientácia

Paradigma objektovej orientácie programovacieho jazyka je v rámci *C#* bohato implementované vo forme mechanizmov ako zapuzdrenie, dedičnosť a polymorfizmus. Tieto princípy spejú pri korektnej implementácii k stavu, kedy každý objekt je znovu použiteľným logickým celkom s presne definovanou zodpovednosťou a dátovou skladbou, ktorá odráža jeho účel, pričom interná logika jednotlivých objektov je od okolitého kódu odtienená a prístupná iba skrz programátorom definované rozhranie [152, 153].

Navzdory orientácii jazyka *C#* primárne na objektovo orientovaný prístup sa stále viac a viac jedná o jazyk viac-paradigmatický – častým dôkazom tohto tvrdenia je značná popularita niektorých funkcionálnych konštruktov postupne integrovaných do *C#* a moderných verzií *.NET* frameworku, ako sú napr. technológie *LINQ* či lambda výrazy [154].

4.1.1.2 Typová bezpečnosť

C# je typovo bezpečný jazyk – s inštanciami všemožných typov je povolené manipulovať iba prostredníctvom protokolov ktoré samy definujú, čím zachovávajú vlastnú internú konzistenciu – príkladom môže byť nemožnosť zavolať funkciu, ktorá akceptuje výhradne celočíselný argument s argumentom iného typu bez toho, aby bol predtým explicitne konvertovaný [152]. Typová bezpečnosť umožňuje zabrániť chybám z nepozornosti. Ako dodatok k typovej kontrole realizovanej za behu *C#* vynucuje statické typovanie – a teda typovú bezpečnosť vo fáze kompilácie. Statické typovanie pomáha zamedziť množstvu chýb ešte pred samotným spustením programu, čo uľahčuje konštrukciu programov väčšieho rozsahu tak, aby boli robustnejšie, predvídateľnejšie a menej náchylné k neočakávaným zlyhaniam.

4.1.1.3 Správa pamäte

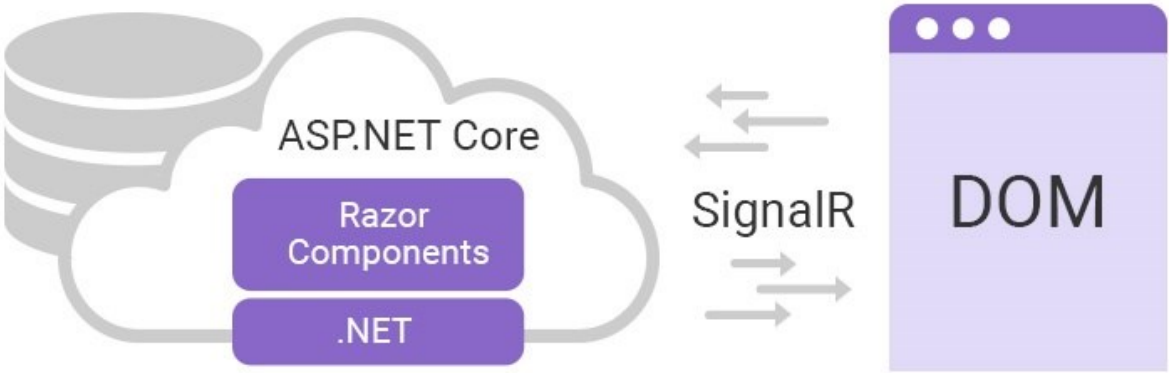
Pamäťový manažment je v jazyku *C#* riadený dynamicky za behu programu – jazyk disponuje modulom bežne označovaným ako zberač odpadu (*Garbage Collector*), ktorý beží spoločne s programom a priebežne uvoľňuje pamäť zaberanú objektmi, ktoré už naďalej nie sú odkazované [154]. Automatizovaná forma správy pamäte uvoľňuje ruky programátora tým, že nemusí osobne riešiť platnosť či neplatnosť existujúcich ukazovateľov a uvoľňovanie pridelenej pamäte, čím šetrí značné množstvo vývojového času. Ukazovatele ako súčasť platnej syntaxe v rámci *C#* síce existujú, ale ich explicitné využitie v rámci väčšiny programátorských úloh nie je nevyhnutné [146].

4.2 Modely nasadenia

Modely nasadenia frameworku *Blazor* (mimo experimentálnych alebo stále vyvíjaných) existujú v súčasnosti dva [155]. Rozdiely medzi nimi nemajú výnimočne výrazný dopad na dynamiku vývoja webovej aplikácie, zato naopak významne vplývajú na spôsob, akým je rozdeľovaná zodpovednosť medzi webovým prehliadačom klienta a serverom, ktorý požiadavky spracúva – v súvislosti s tým sa pochopiteľne významne mení aj komunikačný model spájajúci tieto dva celky.

4.2.1 Blazor Server

Blazor Server je modelom nasadenia, ktorý väčšinu výpočtových úloh realizuje v doméne serveru a hlavnou úlohou klienta je byť vizualizačným prostriedkom a rozhraním pre ovládanie logiky. Celá webová interakcia je spracúvaná na serveri v rámci *ASP.NET Core* aplikácie – vrátane aktualizácií *UI*, obsluhy udalostí či volaní funkcií v jazyku *JavaScript* [155]. Signály, že k týmto akciám došlo sú doručované od klienta k serveru prostredníctvom spojenia *SignalR* (*SignalR* je *open-source* knižnica pre podporu webovej funkcionality v reálnom čase sprístupňujúca serveru možnosť poskytovať obsah klientovi takmer okamžite – samozrejme s ohľadom na obmedzenia komunikačného kanálu) [156].



Obrázok 17. Model nasadenia Blazor Server a ním definovaná forma interakcie užívateľa s grafickým rozhraním aplikácie [157]

Server udržiava stav klientov prostredníctvom špeciálnych štruktúr nazývaných obvody (*Circuits*). Obvod obsahuje obranné mechanizmy umožňujúce pretrvať aj krátkodobé výpadky siete či snahu klienta o obnovenie pripojenia po jeho prerušení [158]. Nevýhodou modelu nasadenia *Blazor Server* je skutočnosť, že každá inštancia aplikácie (napríklad pri otvorení vo viacerých oknách) vyžaduje samostatný obvod k udržiavaniu svojho stavu.

Nutnosť otvárania nových obvodov implikuje alokáciu dodatočných zdrojov na serveri, ktorý danú inštanciu aplikácie obsluhuje. Uzatvorenie okna prehliadača alebo navigácia na externú *URL* adresu spôsobí okamžité uvoľnenie pamäte vyhradenej pre obvod a súvisiace zdroje. Alternatívou korektného uzatvorenia je napr. ukončenie spojenia vplyvom prerušenia siete – v tomto prípade *Blazor Server* udržiava odpojené obvody a ich stav po konfigurovateľne dlhú dobu za účelom umožniť klientovi opätovné pripojenie [158].

Tabuľka 2. Výhody a nevýhody nasadenia frameworku Blazor s využitím modelu Blazor Server [155, 159]

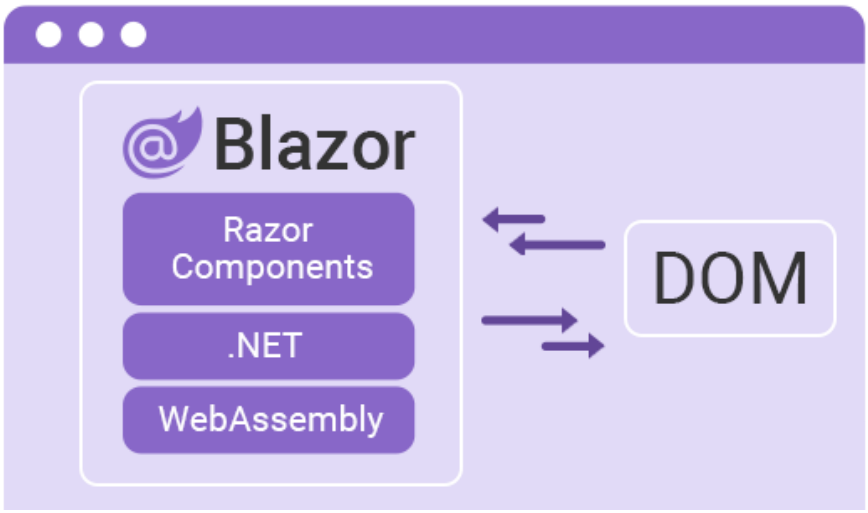
Výhody	Iniciálny objem načítaných dát je oproti *Blazor WebAssembly* výrazne menší, čo garantuje rýchlejšie prvotné načítanie aplikácie.
	Aplikácia z dôvodu kompletného behu v prostredí serveru umožňuje využívať všetky kompatibilné *.NET* knižnice a *API* - dokonca aj tie, ktoré nie sú z prostredia webového prehliadača pri využití *WebAssembly* modelu dostupné.
	Beh aplikácie je spravovaný prostredníctvom *.NET Core* na serveri - to zachováva prístup k pokročilým analytickým možnostiam vrátane plného debuggovacieho režimu.
	Podpora tenkých klientov - *Blazor Server* funguje aj v prehliadačoch, ktoré nepodporujú formát *WebAssembly* a na zariadeniach s limitovanou výpočtovou kapacitou.
	Aplikačný kód tvorený prostredníctvom *.NET / C#* nie je klientom vôbec prístupný (vrátane kódu komponent, s ktorými užívateľ interaguje).
Nevýhody	Operácie majú vyššiu latenciu spôsobenú nutnosťou komunikácie so serverom pri vykonaní ľubovoľnej interakcie s prostredím aplikácie.
	Neexistuje žiadna online funkcionalita – ak zlyhá sieťové prepojenie medzi klientom a serverom, aplikácia prestane fungovať.
	Server je vystavený väčšej záťaži kvôli nutnosti spracovania každej ich akcie a udržiavania priebežného stavu.
	Nemožnosť nasadenia bez serveru napr. s využitím *CDN (Content Delivery Network)*

4.2.2 Blazor WebAssembly

Alternatívnou voľbou k *Blazor Server* je model nasadenia *Blazor WebAssembly*. Princípom spracovania webovým prehliadačom sa *WebAssembly* výrazne líši od modelu *Server* – jedná sa o tzv. *SPA (Single Page Application)* framework [160].

SPA je taká implementácia webovej aplikácie, ktorá načíta iba jediný *HTML* dokument a pokiaľ vznikne potreba aktualizovať jeho zobrazenie, je telo tohto dokumentu nahrádzané informáciami obdržanými prostredníctvom požiadaviek na server. Správna implementácia *SPA* zvyčajne vedie k zlepšeniu výkonu webových stránok, keďže nie je nutné načítať celé stránky, iba dáta ktoré obsahujú [160, 161]. Cenou je vyššia komplexnosť architektonickej a bezpečnostnej stránky pri tvorbe aplikácie, náročnejšie *SEO (Search Engine Optimization)* a nutnosť implementovať zmysluplnú navigáciu.

Blazor WebAssembly operuje s logikou klienta priamo v prehliadači bez nutnosti akýchkoľvek pluginov (na rozdiel od staršej analogickej technológie *Silverlight*) na základe aktuálnych webových štandardov, bez nutnosti rekompilácie kódu do iného jazyka. Beh *.NET* kódu na strane klienta je zabezpečovaný technológiou *WebAssembly* (binárny inštrukčný formát spustiteľný v prostredí webového prehliadača [162]) optimalizovanou pre svižné sťahovanie a rýchlosť spracovania. *WebAssembly* je webovým štandardom podporovaným väčšinou moderných webových prehliadačov vrátane zodpovedajúcich mobilných variant).



Obrázok 18. Model nasadenia Blazor WebAssembly a ním definovaná forma interakcie užívateľa s grafickým rozhraním aplikácie [163]

Tabuľka 3. Výhody a nevýhody nasadenia frameworku Blazor s využitím modelu Blazor WebAssembly [155, 159]

Výhody	Elementárne interakcie s ovládacími prvkami (po úspešnom stiahnutí aplikácie zo serveru) nevyžadujú kontinuálnu komunikácia so serverom. Krátkodobá strata spojenia nemá vplyv - aplikácia ostáva funkčná.
	Odníma výpočtovú náročnosť zo serveru a presúva ju na hardware klienta, lepšie škáluje s väčším množstvom klientov.
	ASP .NET Core webový server nie je nevyhnutný – existujú možnosti produkčného nasadenia bez serveru, vrátane distribúcie aplikácie prostredníctvom *CDN* systému.
Nevýhody	Možnosti klientskej časti aplikácie sú častokrát limitované schopnosťami a obmedzeniami webového prehliadača.
	Kladie nároky na hardware a software využívaný klientom (kvôli spracovaniu časti logiky aplikácie priamo v zariadení klienta. Software musí podporovať minimálne nevyhnutné technológie pre beh aplikácie (napr. *WebAssembly*).
	Iniciálne načítanie je pomalšie v porovnaní s modelom *Blazor Server.*

4.3 Zabezpečenie

Zabezpečenie webových aplikácií môže byť, najmä u rozsiahlych systémov s veľkým množstvom užívateľov v rôznych úlohách a s rôznymi oprávneniami, náročnou úlohou. Implementácia autorizačných / autentifikačných mechanizmov je preto mnohokrát pokrývaná samotným frameworkom – *ASP.NET Core Blazor* v tomto prípade nie je výnimkou. Poskytuje široké spektrum konfigurovateľných nástrojov pre zabezpečenie bezpečnej prevádzky a prenosu dát medzi klientom a serverom, rovnako ako možnosti identifikácie užívateľa a určenie jeho oprávnení v rámci aplikácie [164]. Kontroly a správa zabezpečenia je potom riešená prakticky univerzálne prostredníctvom registrovaného middleware (v kontexte dvojdielnej webovej aplikácie tak označujeme kód, ktorý je vykonávaný pri prijímaní požiadavky ešte pred príchodom spracovania na úroveň *Controller-u* [viď. kapitola *3.1.3 - „Controller“*] – alebo naopak, pri odosielaní odpovede serveru smerom ku klientovi, až po jeho opustení).

4.3.1 Autentifikácia

Autentifikácia je proces overenia identity užívateľa počítačovým programom [165]. Autentifikačný proces môžeme vnímať ako sled akcií, v ktorom užívateľ aplikácii (napr. pomocou znalosti prihlasovacích údajov) preukáže kým v skutočnosti je a aplikácia určí množinu zdrojov, ktorá týmto údajom zodpovedá.

Bežná implementácia autentifikačných postupov na webových stránkach zahŕňa registračnú fázu, kedy osoba vytvára svoj užívateľský účet a prihlasovacie údaje a následné opakované zadávanie vytvorených prihlasovacích údajov do príslušného prihlasovacieho formulára pri pokuse prihlásenia do systému [166, 167]. Alternatívne môže byť z popísaného procesu vypustená registračná fáza, pokiaľ je žiadúce, aby do systému neexistovala možnosť neriadeného prístupu.

Väčšina moderných frameworkov poskytuje už implementované sofistikované mechanizmy na vyriešenie autentifikačnej tematiky. V prípade frameworku *Blazor* autentifikačné procesy zabezpečuje služba nazývaná *IAuthenticationService*. Pred samotným využitím existujúceho *API* pre rozhodovanie autentifikačných problémov je nutné vykonať niekoľko iniciálnych konfigurácií [166]:

1) Registrácia autentifikačného mechanizmu v triede *Startup / Program* webovej aplikácie. Autentifikácia v *.NET Core* podporuje rôzne typy týchto mechanizmov. Miera vhodnosti sa líši v závislosti na povahe a forme nasadenia aplikácie.
   [166] Známe podporované možnosti autentifikácie v .NET Core:
   - **Certifikát**
   - **Cookie**
   - **Jwt Bearer Token**
   - **OAuth**
2) Registrácia služby *IAuthenticationService* v triede *Startup* webovej aplikácie (napr. prostredníctvom začlenenia do metódy *ConfigureServices*). Je možné za účelmi registrácie použiť zvolený kontajner závislostí [viď. kapitola *3.3.1 - „Dependency Injection“*].
3) Pridanie volania funkcie *app.UseAuthentication()* v triede Startup webovej aplikácie (napr. prostredníctvom začlenenia do metódy *Configure*). Táto činnosť vedie k registrácii middleware zodpovedajúceho za autentifikačný proces vždy, keď je spracúvaná požiadavka klienta na server.

Autentifikačný postup a jeho úspech resp. neúspech je po korektnej konfigurácii pravidiel a registrácii potrebných služieb už iba záležitosťou toho, či klient od ktorého plynú požiadavky na server poskytol identifikačné dáta v požadovanej forme, rozsahu a kvalite.

4.3.2 Autorizácia

Autorizácia je proces kontroly užívateľských oprávnení už autentifikovaného užívateľa (musí teda spravidla nastupovať až po fáze autentifikácie) a v rozhodovacích bodoch zahŕňa kontrolu, či daný užívateľ má alebo nemá oprávnenia na zobrazenie / konzumáciu určitého typu obsahu poskytovaného aplikáciou [165].

[168] ASP.NET Core podporuje dva základné prístupy k autorizácii:

- **Role-based:**
  Autorizácia založená na roli užívateľa v rámci systému. Jednoduchší a priamočiary variant, pre väčšinu systémov je táto forma však úplne postačujúca (praktická časť práce taktiež využíva typ autorizácie založený na roliach užívateľov) [169].
- **Policy-based:**
  Autorizácia založená na detailnejších pravidlách (bezpečnostných politikách) kladených na údaje prihláseného autentifikovaného užívateľa [170].

Podobne ako v prípade autentifikácie, registrácia bezpečnostných rolí prebieha v konfiguračnom module *Startup*, pričom umožňuje konfigurovať bezpečnostný middleware aplikácie nasledujúcim spôsobom [169, 170, 171]:

1) Registrácia služby *IAuthorizationService* v triede *Startup* webovej aplikácie (napr. prostredníctvom začlenenia do metódy *ConfigureServices*). Je možné za účelmi registrácie použiť zvolený kontajner závislostí [viď. kapitola *3.3.1 - „Dependency Injection“*].
2) Pridanie volania funkcie *app.UseAuthorization()* v triede Startup webovej aplikácie (napr. prostredníctvom začlenenia do metódy *Configure*). Toto volanie musí byť realizované až po predchádzajúcom *app.UseAuthentication()* – na poradí záleží, keďže pokiaľ užívateľ nie je v prvom rade autentifikovaný, sú akékoľvek autorizačné rutiny zbytočné.

Aplikácia autorizácie je po iniciálnej konfigurácií a nadefinovaní pravidiel mimoriadne jednoduchá a prebieha prostredníctvom dekorovania častí aplikácie špeciálnymi atribútmi [172]. Použitie je principiálne zhodné, názvy atribútov sa mierne líšia vzhľadom na časť kódu, ktorú s nimi označujeme [172, 173].

- **Atribút [Authorize]:**
  Označuje celé *Controller*-y alebo ich endpoint-y. Značí, že pri snahe pristúpiť k danej logike užívateľ skutočne disponuje oprávneniami na výkon danej akcie (napr. prístup do správy užívateľských účtov má iba administrátor). V rámci tvorby *Blazor* užívateľského rozhrania potom umožňuje naviac označovať aj celé stránky (nie však už ich časti / komponenty).
- **Atribút [AuthorizeView]:**
  Atribút určený k ochrane špecifických častí užívateľského rozhrania pred neoprávnenou manipuláciou (resp. manipulácia s generovaným *HTML* dokumentom na úrovni viditeľnosti / neviditeľnosti označených prvkov stránky). Princíp použitia je analogický k vyššie popísanému *Authorize* – rozdielne sú iba cieľové časti kódu, ku ktorým sa vzťahuje.
- **Atribút [AllowAnonymous]:**
  Označuje funkcionalitu, ku ktorej môže pristúpiť ktorýkoľvek užívateľ aplikácie aj bez toho, aby disponoval užívateľským účtom. Dáva zmysel napr. pri akciách ako prihlásenie či registrácia (je zrejmé, že užívateľ ktorý sa prihlasuje / registruje účtom buď nedisponuje, alebo ním momentálne nie je autentifikovaný).

```
1  void ConfigureAuthMechanisms()
2  {
3      app.UseIdentityServer();
4      app.UseAuthentication();
5      app.UseAuthorization();
6  }
```

Zdrojový kód 1. Volanie metód určených k registrácii autentifikačných a autorizačných mechanizmov

4.3.3 Ďalšie možnosti

Okrem mechanizmov na sprostredkovanie vopred nachystaných autorizačných či autentifikačných funkcionalít je súčasťou *.NET Core* ekosystému aj niekoľko takmer hotových riešení, na ktorých je nutné iba vykonať základné úpravy či mierne ich prispôsobiť na mieru vyvíjanej aplikácií. *Microsoft* za účelom riadenia zabezpečovacích mechanizmov odporúča využívať sadu knižníc a rozhraní súhrnne nazývaných *ASP.NET Core Identity* [174].

4.3.3.1 ASP.NET Core Identity

Jedným z oficiálne odporúčaných riešení problematiky zabezpečenia webovej aplikácie založenej na frameworku *ASP.NET Core* (resp. *Blazor*) je implementácia autorizačných / autentifikačných mechanizmov prostredníctvom sady aplikačných rozhraní *ASP.NET Core Identity – ASP.NET Core Identity* je kvôli podobnosti pomenovaní často chybne zamieňaný za výrazne odlišnú technológiu *Microsoft Identity Platform* (ktorá je založená na vývojárskej platforme *Azure Active Directory*). Okrem parciálnej zhody názvov však tieto technológie nemajú príliš spoločného [124, 174].

ASP.NET Core Identity sa dá popísať ako *API*, ktoré ponúka základovú implementáciu *UI* slúžiaceho na registráciu, prihlásenie či odhlásenie užívateľa, vrátane podkladovej logiky pre správu užívateľov, hesiel, profilových údajov, rolí, zabezpečovacích tokenov a kontroly potvrdenia užívateľských účtov prostredníctvom e-mailovej schránky [174]. Údaje pre využitie v rámci funkcionalít poskytovaných rozhraniami *Identity* sú zvyčajne ukladané v databázovom úložisku relačného typu (z dôvodu kompatibility sa v rámci *.NET* ekosystému prevažne využíva úložisko *MS SQL Server*). *Identity* podporuje aj použitie alternatívnych databázových možností - napr. *Azure Table Storage*. *ASP.NET Core Identity* je vedený ako *Open-Source* [175].

Okrem prihlásenia prostredníctvom užívateľského účtu vedeného v rámci aplikácie samotnej umožňuje *ASP.NET Core Identity* aj autentifikáciu s využitím užívateľských profilov tretích strán – medzi tieto patrí *Facebook*, *Google*, *Microsoft* a *Twitter* [174].

Konfigurácia a personalizácia funkcionalít v réžii *ASP.NET Core Identity* je možná, podobne ako u ostatných zabudovaných autorizačných / autentifikačných mechanizmov prostredníctvom konfiguračných metód v programovom module (triede) *Startup / Program*.

```
void SetupAuthMechanisms()
{
    builder.Services.AddIdentityServer()
    .AddApiAuthorization<ApplicationUser, ApplicationDbContext>(options =>
    {
        options.IdentityResources["openid"].UserClaims.Add("name");
        options.ApiResources.Single().UserClaims.Add("name");
        options.IdentityResources["openid"].UserClaims.Add("role");
        options.ApiResources.Single().UserClaims.Add("role");
    });
    JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Remove("role");
    builder.Services.AddAuthentication().AddIdentityServerJwt();

    builder.Services.AddAuthentication().AddGoogle(googleOptions =>
    {
        googleOptions.SignInScheme = IdentityConstants.ExternalScheme;
        googleOptions.ClientId = builder.Configuration["Authentication:Google:ClientId"];
        googleOptions.ClientSecret = builder.Configuration["Authentication:Google:ClientSecret
"];
    });
}
```

Zdrojový kód 2. Možná podoba metódy zodpovednej za konfiguráciu autentifikačných služieb webovej aplikácie s využitím technológie ASP .NET Core Identity.

4.3.3.2 Alternatívy

Existuje samozrejme aj možnosť nepoužiť predpripravené konštrukty poskytované frameworkom *.NET Core* a vytvoriť si vlastné autorizačné a autentifikačné prostriedky – tento prístup ale nie je odporúčaný, najmä z dôvodu mimoriadnych časových výdajov na implementáciu a taktiež kvôli skutočnosti, že riešenie vytvorené vývojárom, resp. menším vývojovým tímom - mnohokrát s nedostatkom skúseností či času - má veľkú šancu kvalitatívne zaostávať za dlhodobo udržiavanými, časom overenými a odladenými riešeniami poskytovanými frameworkom.

II. PRAKTICKÁ ČASŤ

5 SOFTWAROVÝ NÁVRH

Návrh a design softwarových riešení je v súčasnosti neodmysliteľnou súčasťou vývojového cyklu takmer každého softwarového produktu dostatočného rozsahu. Predbežný návrh v nejakej forme využívame u prakticky každého systému, kde je pre osoby činné v rámci projektu náročné udržať v ľubovoľnom momente všetky potrebné informácie o jeho zameraní, účele, požadovanom vzhľade, špecifikách business logiky a pod.

Návrhové metodiky a zaužívané modelačné nástroje popísané v nasledujúcich odsekoch umožňujú stanoviť jasnú úlohu a smer, ktorým sa produkt uberá – vrátane štruktúr, prostriedkov a procesov nevyhnutných k uznaniu implementovaného riešenia zadávateľom za riešenie úplné a uspokojivé.

5.1 Všeobecné informácie

Praktická zložka práce v podobe programu webového systému sa zakladá na predtým realizovanom softwarovom návrhu. Vytvorený návrh je možné vnímať ako určitú formu odporúčania rešpektovaného nasledovnou implementáciou systému. Hlavnou zodpovednosťou tohto návrhu je dosiahnuť, aby boli možnosti vytváraného systému pokiaľ možno čo najkompletnejšie (v zmysle pokrytia požiadaviek na jadrové funkcie), aby boli vzaté do úvahy či už štandardné alebo výnimočné stavy, do ktorých sa môže užívateľ svojou interakciou s aplikáciou dostať a v neposlednom rade úsilie maximalizovať efektivitu v zmysle svižnosti implementačnej časti vývoja. Vývoj software ako celok bol v prípade tejto práce založený na vývojovom modeli označovanom v teórii vývoja software ako model typu *Vodopád* [176].

Vzhľadom na požadované výstupy bol zvolený nasledovný pracovný postup:

1) **Analýza a zber požiadaviek**
2) **Tvorba modelov a diagramov**
    - UC model, jeho parciálne diagramy a scenáre
    - Drôtené modely užívateľského rozhrania
    - Diagram tried (dátový pohľad)
3) **Implementácia systému**

Model typu *Vodopád* je v súčasnosti z trhu aktívne vytláčaný metodikami založenými na agilnom vývoji softwaru [177] – v prípade vytváraného systému pre správu objednávok sa však ponúkal ako prirodzená voľba na základe viacerých kľúčových faktorov:

- Vývojový tím o veľkosti jednej osoby,
- Zvládnuteľný rozsah systému bez nutnosti prírastkového vývoja,
- Časové limitácie vývoja.

Spätne sa ukázalo, že voľba implementačného postupu bola správna – rozsah tvoreného systému dovoľoval realizovať návrh v časovom predstihu bez nadmerného výskytu problémov odhalených až v neskorších fázach vývoja.

5.1.1 Motivácia

Motivácie k vytvoreniu analytickej časti softwarového riešenia sa môžu medzi rôznymi projektami pochopiteľne líšiť. Významným motivačným faktorom býva snaha pridať do riešenia dodatočnú vrstvu abstrakcie medzi rozsiahlou analýzou a implementačnou fázou, snaha predchádzať zbytočným chybám prameniacich z nepochopenia / nedorozumenia medzi zákazníkom a dodávateľom, túžba kvalitnejšie odhadnúť náročnosť riešenia a vo výsledku zredukovať finálne náklady na dodávaný software prostredníctvom dôkladného a uchopiteľného návrhu, či kondenzácia masívnych textových špecifikácii náročných na pochopenie do tvaru spracovateľného programátorom bez predchádzajúceho oboznámenia so všetkými špecifikami implementovanej business logiky [178].

Analytická časť tejto práce popísaná v nasledujúcich kapitolách sa zaoberala hrubým návrhom cieľovej funkcionality v podobe vybraných typov modelov a im príslušných diagramov za účelom získania uceleného konceptu toho, ako bude vyzerať implementačná fáza tvorby systému – aké problémy bude aplikácia prioritne riešiť, ktoré moduly by mala obsahovať, akým spôsobom vzájomne interagujú a akú formu budú nadobúdať elementárne komponenty – od dátových až po tie vizualizačné.

Jednotlivé moduly návrhu priniesli nasledovné benefity:

- **Požiadavky:**

  Spracovanie, analýza a výsledné modelovanie *Požiadaviek* jasne stanovilo, ktoré funkcionality sú pre zachovanie kľúčovej myšlienky a účelu tvoreného software nevyhnutné a taktiež to, či je funkcionalita popisovaná danými *Požiadavkami* nejakým spôsobom logicky prepojená.

- **Model prípadov použitia:**

  Sprostredkúva náhľad do množiny typických *Aktérov* systému a ich zodpovedností a ešte pred samotnou implementáciou predstavuje prístup k cenným informáciám ovplyvňujúcim široké spektrum činností naprieč celým vývojovým procesom. Významné faktory popísané v rámci modelu *Prípadov použitia* sú napr. role *Aktérov*, typické činnosti *Aktérov* v rámci systému a potenciálne závislosti / súvislosti týchto aktivít.

  - **Scenáre:**

    Scenáre zahrnuté do jednotlivých prípadov použitia popisujú typický priechod užívateľa disponujúceho prístupom k tomuto prípadu použitia danou aktivitou. Vytipovanie *Scenárov* (vrátane prípadných alternatívnych či chybových priechodov) vedie k lepšiemu pochopeniu procesov a formuje predstavu o bežných pracovných postupoch užívateľa. Zamyslenie sa nad alternatívnymi priechodmi v rámci *Scenárov* ďalej vnáša potrebné svetlo do krajných prípadov, ktoré je potrebné v rámci softwarového riešenia mať na pamäti a vhodným spôsobom ošetriť.

- **Model tried:**

  Model *Tried* (*Class Model*) a stavba dátových štruktúr v rámci aplikácie, ktorú popisuje poskytuje neoceniteľný prínos pre programovanie business logiky. Prehľadnou formou zachytáva prvky ktoré aplikáciu tvoria, čo významnou mierou zvyšuje rýchlosť práce vývojára, keďže možnosť implementovať funkčné celky podľa vopred pripravenej prehľadnej schémy redukuje povahu implementácie na svižnú prácu mechanického charakteru.

- **Drôtené modely:**

  Drôtené modely sa ukázali byť mimoriadne užitočným prostriedkom pre konceptuálny návrh *GUI* aplikácie. Predpis vo forme týchto návrhov, podobne ako u modelu *Tried*, zvyšuje efektivitu programátorskej práce tým, že pri implementácii vizualizačnej logiky aplikácie nie je nutné zamýšľať sa nad pozíciou / členením / funkcionalitou jednotlivých ovládacích prvkov, iba konštruovať čo najvernejšiu reprezentáciu pripravených náčrtov prostredníctvom zvolenej množiny technológií.

5.1.2 Jazyk UML

Jedným z nástrojov návrhu software využitým pre tvorbu viacerých diagramov je *UML* (*Unified Modeling Language*). *UML* je technológiou prvýkrát uvedenou na trh koncom minulého tisícročia – od tej doby sa dočkal viacerých revízií a vylepšení, pričom v súčasnosti *UML* predstavuje najrozšírenejší a univerzálne uznávaný prostriedok k definovaniu, špecifikovaniu, konštruovaniu, dokumentovaniu a vizualizácii rôznorodých častí softwarových systémov či modelovanie a popis business procesov [179, 180]. *UML* je možné použiť v takmer ľubovoľnej vedomostnej doméne a je kompatibilné s väčšinou komponentovo a objektovo orientovaných metodík vývoja software.

Modelovací jazyk *UML* nie je vývojovým procesom – nešpecifikuje aké úlohy majú byť v rámci vývoja riešené, neumožňuje riadenie aktivít vývojového tímu a nenasádza kritéria na vyhodnocovanie implementačných aktivít [180] – voľbu vývojového procesu teda necháva plne v réžii vývojového tímu. *UML* naviac stanovuje štandardy princípov tvorby rôznych typov diagramov, ktoré sú súčasťou *UML* špecifikácie.

[181] Špecifikácia UML člení diagramy podľa typu na dve hlavné rodiny:

- **Štrukturálne:**
  Diagramy štrukturálneho typu popisujú statickú stavbu systému – t.j. štruktúru ktorá ostáva nemenná, resp. je nezávislá od aktivít a činností prebiehajúcich v prostredí daného systému.
- **Behaviorálne:**
  Diagramy behaviorálneho typu popisujú premenlivé zložky objektov v rámci systému ako postupnosť zmien stavov v závislosti na čase resp. aktivitách a činnostiach, ktoré systém vykonáva.

Modely implementované v rámci praktickej časti práce obsahujú dva typy diagramov zakorenených v *UML* – jedná sa o diagram *Prípadov použitia* a diagram *Tried*. *Drôtené modely* a model *Požiadaviek* (navzdory vizuálnej podobnosti modelu požiadaviek k ostatným diagramom) sú síce neoceniteľným popisným prostriedkom systému, ale nie sú to koncepty popisované jazykom *UML*.

5.1.3 Technológie

Kvalita výstupov každej aktivity spojenej s vývojom technologických riešení je v súčasnosti veľmi úzko previazaná s kvalitou a vhodnosťou nástrojov zvolených za účelom ich tvorby. Výnimkou nie je ani oblasť modelovania a designovania webových systémov – z toho dôvodu boli dôkladne uvážené technológie využité na spracovanie jednotlivých výstupov s dôrazom najmä na jednoduchosť práce s nimi a vizuálnu konzistenciu výstupov.

5.1.3.1 Enterprise Architect

Software Enterprise Architect (*EA*) od firmy *Sparx Systems* je nástrojom na analýzu a modelovanie softwaru. Poskytuje kompatibilitu s technológiami *UML, SysML, BPMN* a mnohými ďalšími, pričom pokrýva všetky fázy vývoja – od zberu a správy *Požiadaviek*, cez analýzu, fázu modelovania, implementácie, testovanie až po nasadenie a priebežnú údržbu. S modelmi umožňuje pracovať prostredníctvom grafického rozhrania a disponuje rozsiahlymi dokumentačnými zdrojmi. Primárnym cieľom software *Enterprise Architect* je zabezpečiť prostriedky pre konštrukciu robustného a udržateľného software [182]. Ďalšou výhodou *EA* sú možnosti exportu výstupov – či už formou podpory pre formátovaný výstup grafickej povahy, alebo v niektorom zo štruktúrovaných textových formátov.

Enterprise Architect je spoplatneným produktom [183]. Licencia použitá pre potrebu tvorby diplomovej práce bola zdieľanou akademickou licenciou a k internej funkcionalite prostredia softwaru teda bolo pristupované s využitím zdieľaného kľúčového úložiska.

5.1.3.2 Figma

Figma je online nástrojom využívaným pre návrh užívateľského rozhranie pýšiaca sa silnými možnosťami kolaborácie v rámci vývojového tímu. *Figma* poskytuje nástroje pre tvorbu ako rastrovej, tak vektorovej grafiky, a zakomponovanie týchto prostriedkov do kontrolných celkov [184]. Naviac disponuje užitočnými pokročilými možnosťami ako napr. združovanie prvkov návrhu do znovu-použiteľných funkčných skupín, prístup viacerých spolupracovníkov do vytváraného prototypu súčasne či možnosťou generácie kódu priamo z vytvorených štýlov.

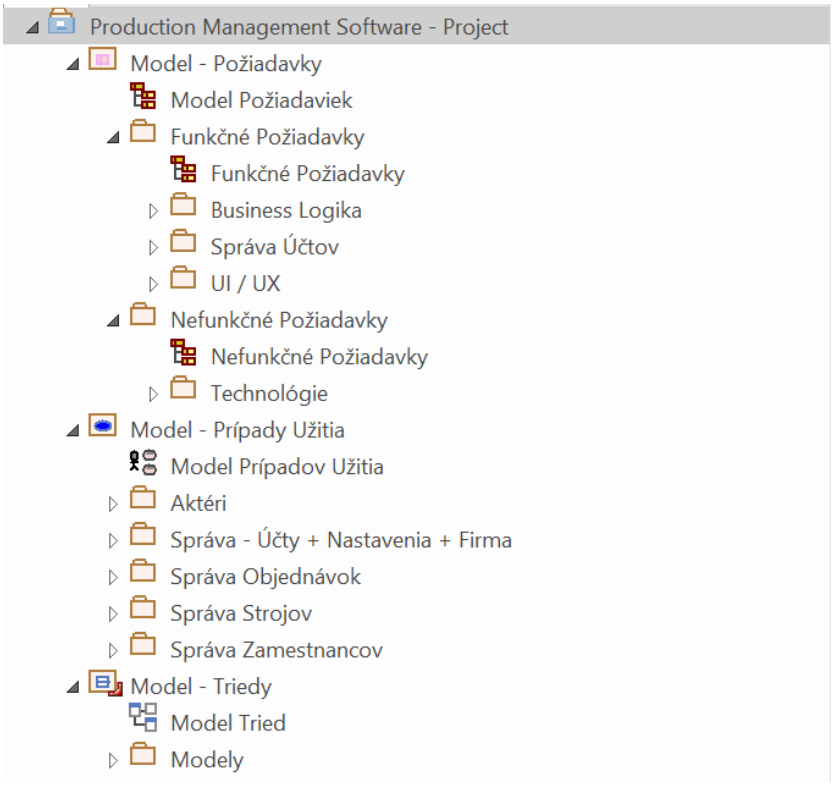
Figma v súlade s potrebami navrhovaného systému pre správu objednávok použitá za účelom tvorbu *Drôtených modelov* (*Wireframes*) na konceptuálnej úrovni – t.j. tak, aby bolo z detailu zrejmé, aké ovládacie prvky sú na stránkach prítomné vrátane ich vzájomného rozloženia, bez prílišného dôrazu na konkrétne štýly či farebnosť.

Väčšina skutočne pokročilej funkcionality softwaru *Figma* však nebola v prípade tvorby prototypov vzhľadom na povahu a rozsah projektu využitá. Schopnosti aplikácie *Figma* boli bezplatne využívané prostredníctvom študentskej licencie získanej prostredníctvom registrácie do edukačného programu priamo na príslušných webových stránkach [185].

5.2 Modelovanie systému

Modelovanie cieľového systému pre správu objednávok využívalo oba hlavné nástroje popísané v predchádzajúcich odstavcoch [viď. kapitola *5.1.3 - „Technológie“*].

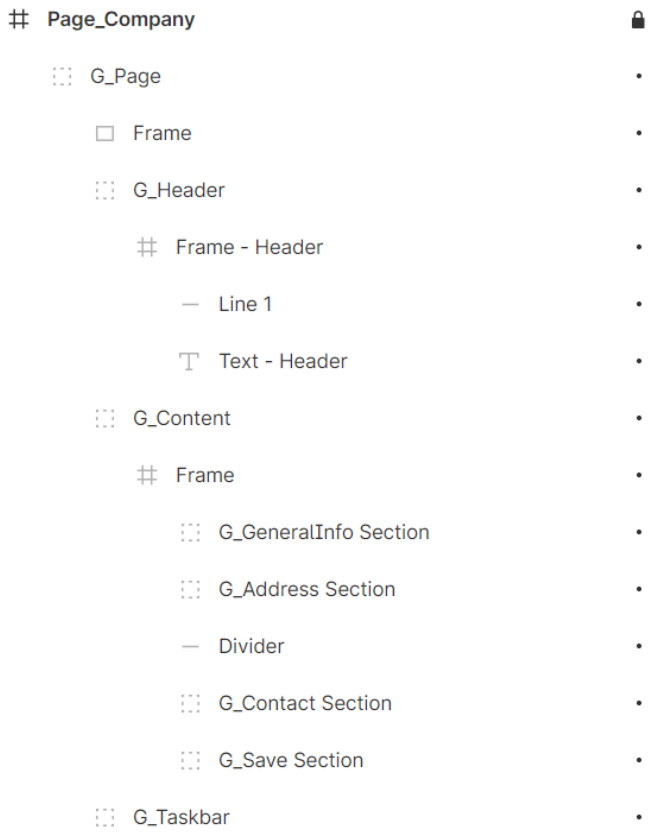
Prostredie *Enterprise Architect* poslúžilo ako nástroj pre vytvorenie modelu *Požiadaviek*, diagramov *Prípadov použitia* a diagramu *Tried*. *Enterprise Architect* umožňuje elegantné organizovanie častí projektu do stromovitej štruktúry nie nepodobnej súborovému systému. Projektová hierarchia teda bola vytvorená na základe spracúvaných celkov a ich ďalšieho delenia závislého od kvázi-separátnych funkčných celkov modelovanej aplikácie.



Obrázok 19. Štruktúra projektu v software Enterprise Architect

Prostredie *Figma* bolo adresované priamo prostredníctvom webovej služby a disponuje taktiež mimoriadne schopnými organizačnými možnosťami. Drôtené modely vypracované v tomto prostredí (keďže ide o *GUI* skladané z jednotlivých komponentov, resp. menších funkčných celkov).

Figma komponenty nie sú rozdeľované do súborov ako takých, navzdory tomu je však možné zoskupenie prvkov tvoriacich jednotlivé stránky do hierarchickej štruktúry nie nepodobnej štruktúre *HTML* dokumentu - s dodatočnou vrstvou abstrakcie spočívajúcej v tom, že jednotlivé komponenty nemusia byť mapované 1:1 na reprezentáciu ich *HTML* prvkov.



Obrázok 20. Štruktúra drôteného modelu stránky vytvorenej s využitím nástroja Figma

Modelovanie funkcií a zodpovedností systému prostredníctvom k tomu určených nástrojov a diagramov v mnohých prípadoch umožňuje znázornenie prepojenia či väzby medzi položkami prostredníctvom vzťahov. Počas tvorby diagramov bol kladený dôraz na zmysluplné zachytenie týchto vzťahov takým spôsobom, aby bola z diagramov zreteľná forma vzájomnej interakcie jednotlivých častí. *UML* medzi tieto vzťahy radí napr. *Agregáciu*, *Kompozíciu*, *Závislosť*, *Asociáciu*, *Generalizáciu* či *Realizáciu*.

5.2.1 Model požiadaviek

Spracovanie problematiky *Požiadaviek* v rámci vytvorenej organizačnej štruktúry a parciálnych diagramov reprezentujúcich jednotlivé kategórie okrem rozmiestnenia samotných položiek *Požiadaviek* do zrozumiteľnej diagramovej podoby pridáva každej *Požiadavke* krátke objasnenie, resp. popis danej funkcionality (Pri prehliadaní v prostredí *.eap* projektu je zobrazenie tohto popisu dostupné kliknutím ľavého tlačidla myši na danú požiadavku).

Okrem samotných objektov *Požiadaviek* a im zodpovedajúcich popisov boli do diagramu prirodzene umiestňované vzťahy medzi jednotlivými *Požiadavkami* (v prípade, že existovala previazanosť medzi nimi popisovanou funkcionalitou). Pokryté funkčné celky umožnili využitie rôznorodých typov vzťahov medzi položkami požiadaviek.

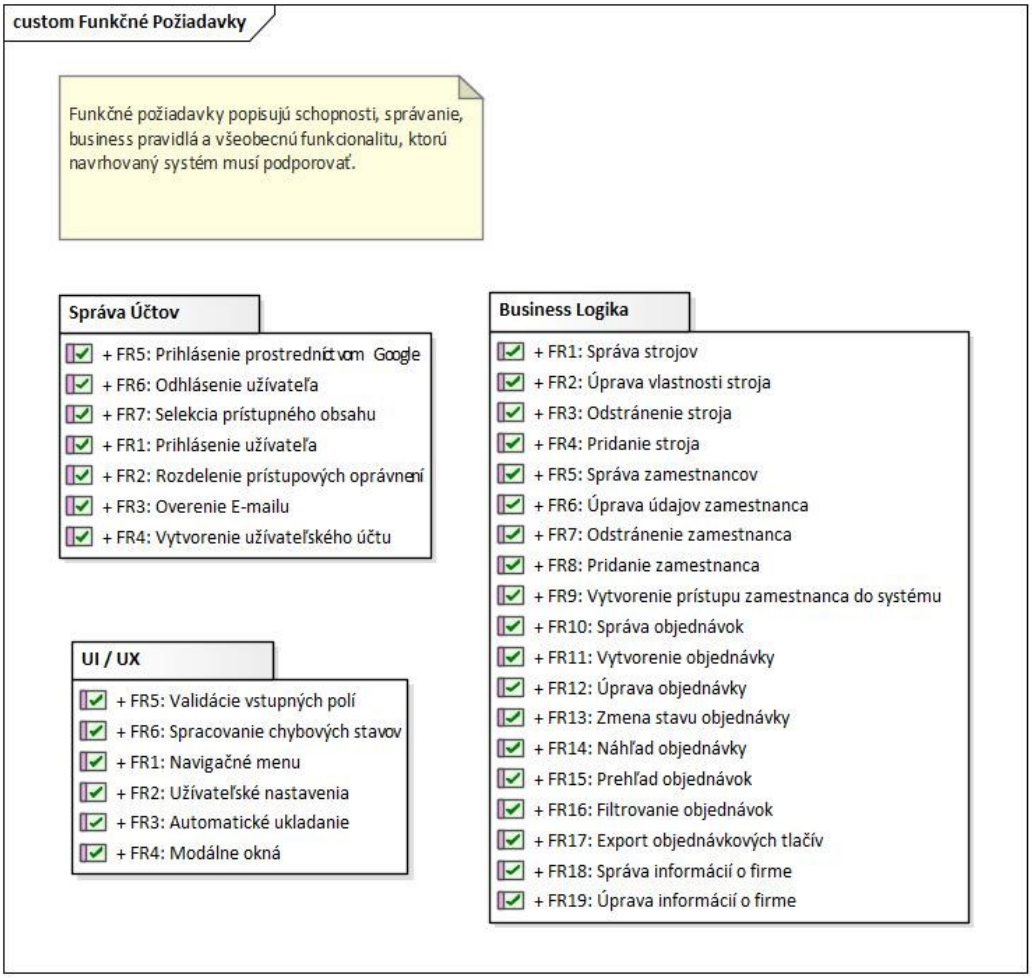
[186, 187, 188] Základné delenie požiadaviek:

- **Funkčné požiadavky:**
  Popisujú základné funkcie ponúkané systémom. Pokiaľ nedôjde k naplneniu funkčných požiadaviek, nie je implementácia systému úplná – jedná sa vlastne o *Požiadavky* kladené užívateľom na funkcionalitu obsiahnutú finálnou implementáciou.
- **Nefunkčné požiadavky:**
  Požiadavky popisujúce všetko, čo funkčné *Požiadavky* nezachytávajú. Nefunkčné *Požiadavky* sú značne rozmanitou kategóriou, keďže môžu obsahovať nároky kladené na rôzne, výrazne odlišné časti systému – napr. odozvu, výkon, zabezpečenie, kompatibilitu, udržateľnosť, škálovateľnosť a pod.

5.2.1.1 Funkčné požiadavky

Funkčné *Požiadavky* reprezentujú behaviorálne požiadavky na aplikáciu – stanovujú, akým spôsobom bude navrhovaný systém spracovávať informácie a určujú jeho schopnosti a pravidlá, ktorých implementácia je nutná k uznaniu funkcionality za úplnú [188].

Funkčné *Požiadavky* sú v rámci analytického spracovania software pre správu objednávok rozčlenené do troch rôznych kategórií podľa ich logického zamerania. Tieto kategórie sú v rámci vytvoreného *.eap* projektu reprezentované formou viacerých balíčkov.

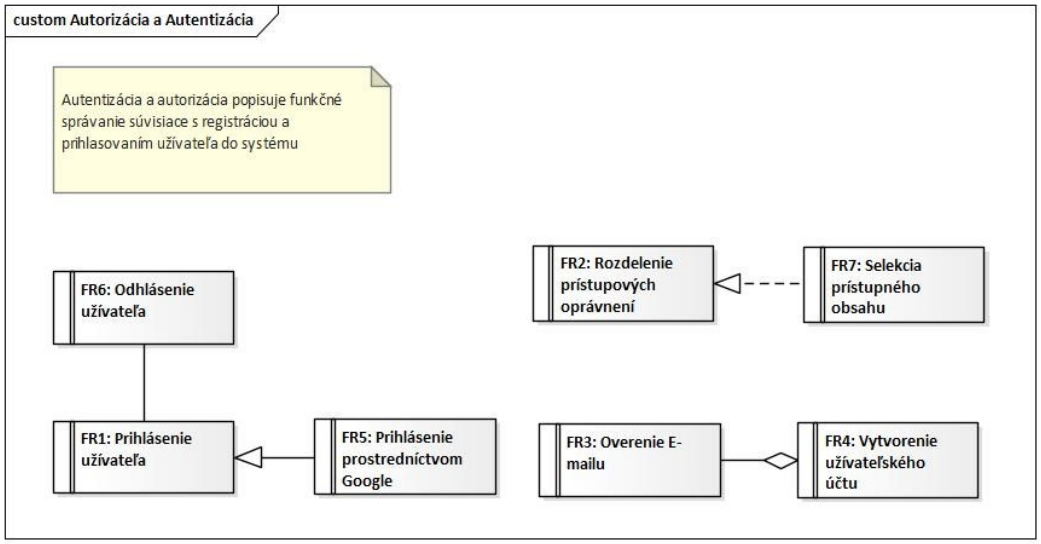


Obrázok 21. Prehľad a členenie funkčných požiadaviek v rámci navrhovaného systému

Hlavné balíčky funkčných požiadaviek:

- **Správa účtov:**

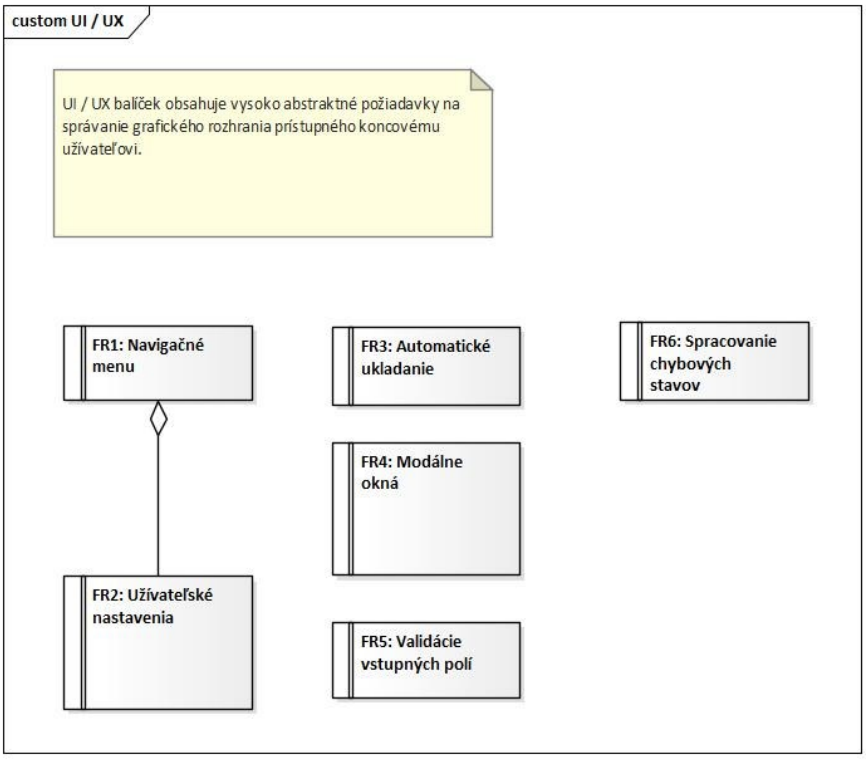
  Balíček zahŕňajúci *Požiadavky* správy účtov definuje funkcionality, ktorými systém musí disponovať k zabezpečeniu uspokojivej separácie od voľne prístupnej infraštruktúry Internetu, k očakávanej prevádzke autorizačných či autentifikačných mechanizmov a procesným úkonom už špecificky odvíjaným od zamýšľaného použitia systému ako takého.



Obrázok 22. Funkčné požiadavky, modul „Správa účtov“

- **UI / UX:**

  Množina funkčných *Požiadaviek* zaoberajúcich sa UI / UX stanovuje schopnosti systému v zmysle interakcie s užívateľom, zaručenia dostupnosti požadovanej funkcionality a reakcie na neočakávané situácie a zadávania užívateľských nastavení za účelom personalizácie určitých typov funkcionalít, ktoré systém zohľadňuje naprieč širokým spektrom úkonov v rámci internej business logiky.



Obrázok 23. Funkčné požiadavky, modul "UI / UX"

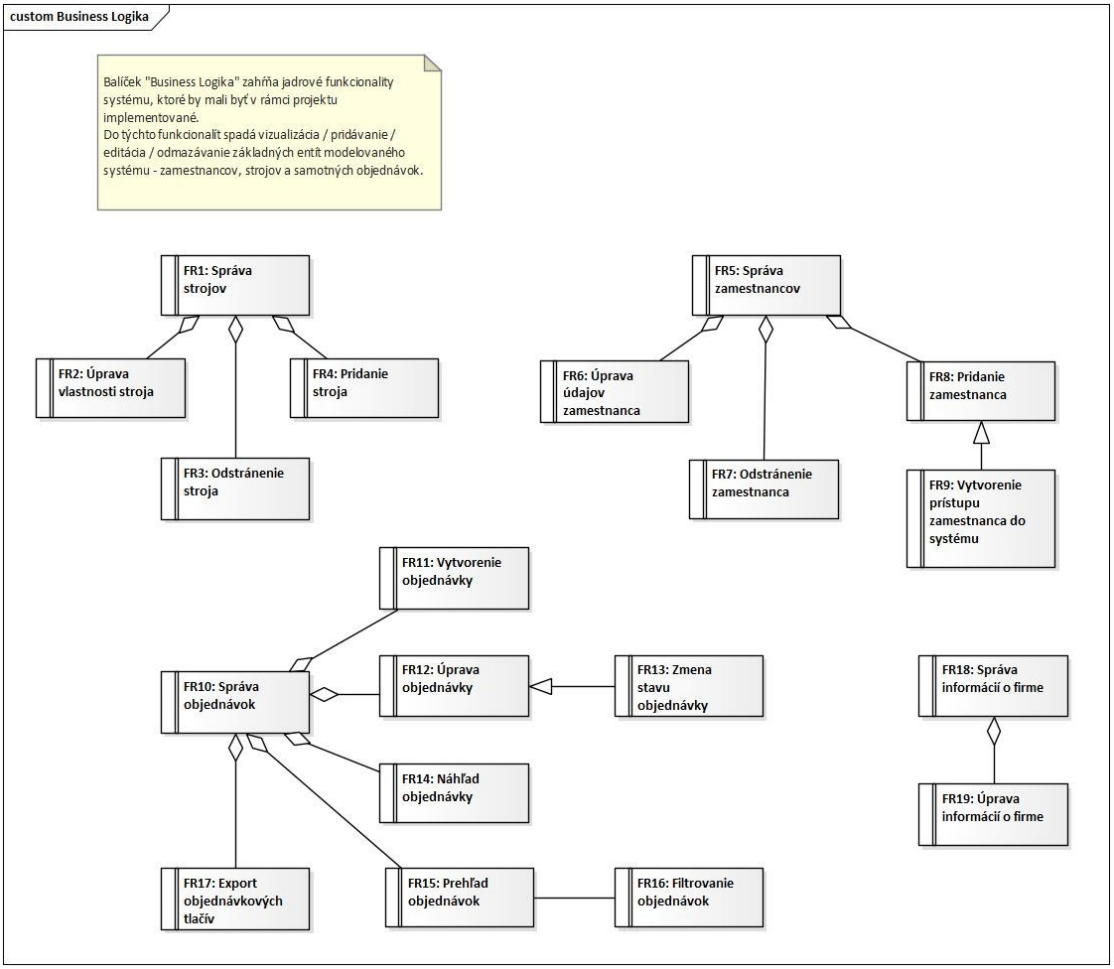
- **Business Logika:**

  Balíček *Požiadaviek* adresujúcich aplikačnú logiku je v prípade navrhovaného systému pre správu objednávok balíčkom bezkonkurenčne najrozsiahlejším - a vzhľadom na primárne zameranie systému objektívne najdôležitejším. Pokiaľ by bol realizovaný detailnejší rozpad, tak je možné členenie balíčku na viacero oblastí záujmu (to reflektuje aj UC model [viď. kapitola *5.2.2 - „Model prípadov použitia“*]).

  Tieto podcelky sú nasledovné:

  - Správa zamestnancov
  - Správa výrobných zariadení
  - Správa objednávok
  - Správa informácií o firme

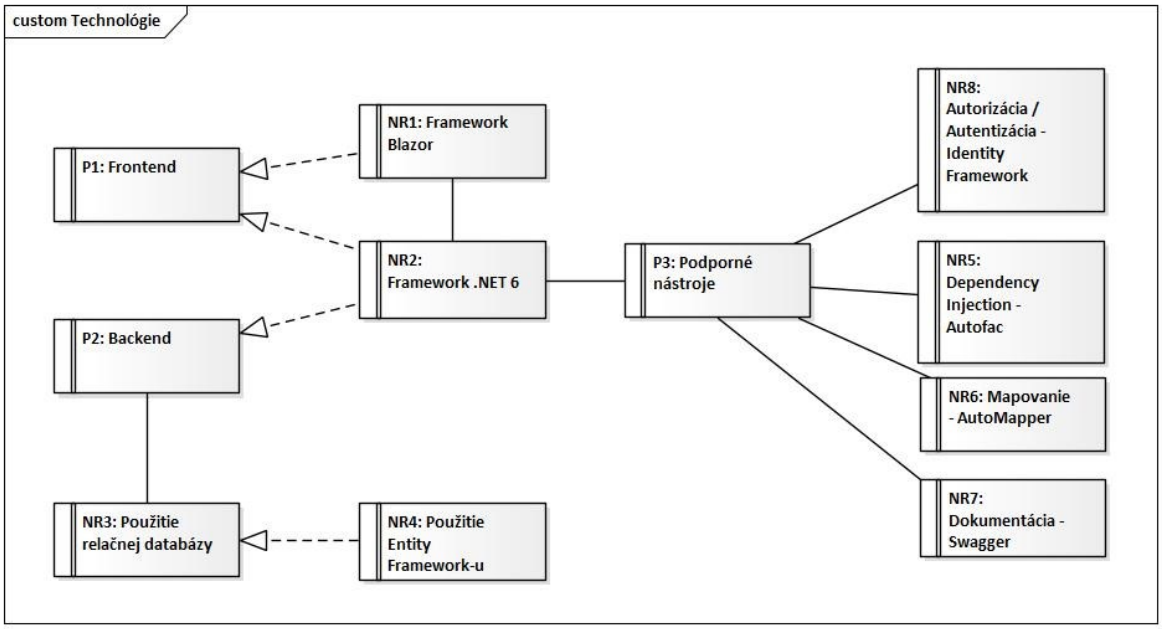
  Všetky uvedené celky tvoriace business logiku, vrátane ich úspešnej implementácie, sú integrálne pre plnú prevádzku systému, v súlade s jeho pôvodnou myšlienkou.



Obrázok 24. Funkčné požiadavky, modul „Business Logika“

5.2.1.2 Nefunkčné požiadavky

Informácie známe na začiatku vypracovávania návrhu systému neumožňovali odvodenie dostatočného množstva poznatkov štandardne tvoriacich nefunkčné *Požiadavky.* Bolo by náročné - a potenciálne zbytočné - robiť predbežné odhady o povahe zariadení, na ktorých bude systém prevádzkovaný, rýchlostiach prístupného hardware / software, sieťovej priepustnosti a pod. Nefunkčné *Požiadavky* z tohto dôvodu obsahujú iba jeden balíček nesúci zhrnutie skutočností, ktoré boli vopred stanovené – a síce definíciu technologických prostriedkov využitých k praktickej realizácii daného systému.



Obrázok 25. Nefunkčné požiadavky, modul „Technológie“

5.2.2 Model prípadov použitia

Model *Prípadov použitia* – resp. jednotlivé parciálne diagramy ktorými je tvorený (*Use Case Diagrams*) – je behaviorálnym typom diagramu [181]. Zachytáva teda správanie systému pri interakcii s *Aktérmi*, ktorí sú s ním schopní nejakým spôsobom interagovať. Diagramy *Prípadov použitia* zachytávajú vzťahy medzi jednotlivými *Prípadmi použitia* a taktiež nesú informáciu o tom, ktorí *Aktéri* sú schopní k týmto *Prípadom použitia* pristupovať. Nezachytávajú však už presný postup interného spracovania danej operácie systémom. Diagramy *Prípadov použitia* by mali byť silne previazané so zozbieranými a analyzovanými funkčnými *Požiadavkami* – je kľúčové, aby boli tieto *Požiadavky* modelovanými *Prípadmi použitia* pokryté (t.j. každá funkcionalita zahrnutá v *Požiadavkách* by mala mať buď vlastný *UC*, alebo byť v rámci *UC* diagramu iným spôsobom opatrená) [189].

[190] Diagram prípadov použitia obsahuje nasledujúce prvky:

- **Prípady použitia:**

  Funkcionalita ktorou systém disponuje za účelom naplniť určitý cieľ *Aktéra* (bez ohľadu na to, či ide o *Aktéra* ľudského). Výstupom z každého *Prípadu použitia* by mal byť pozorovateľný / merateľný výstup, ktorý má z pohľadu zúčastneného užívateľa alebo systému určitú informačnú hodnotu.

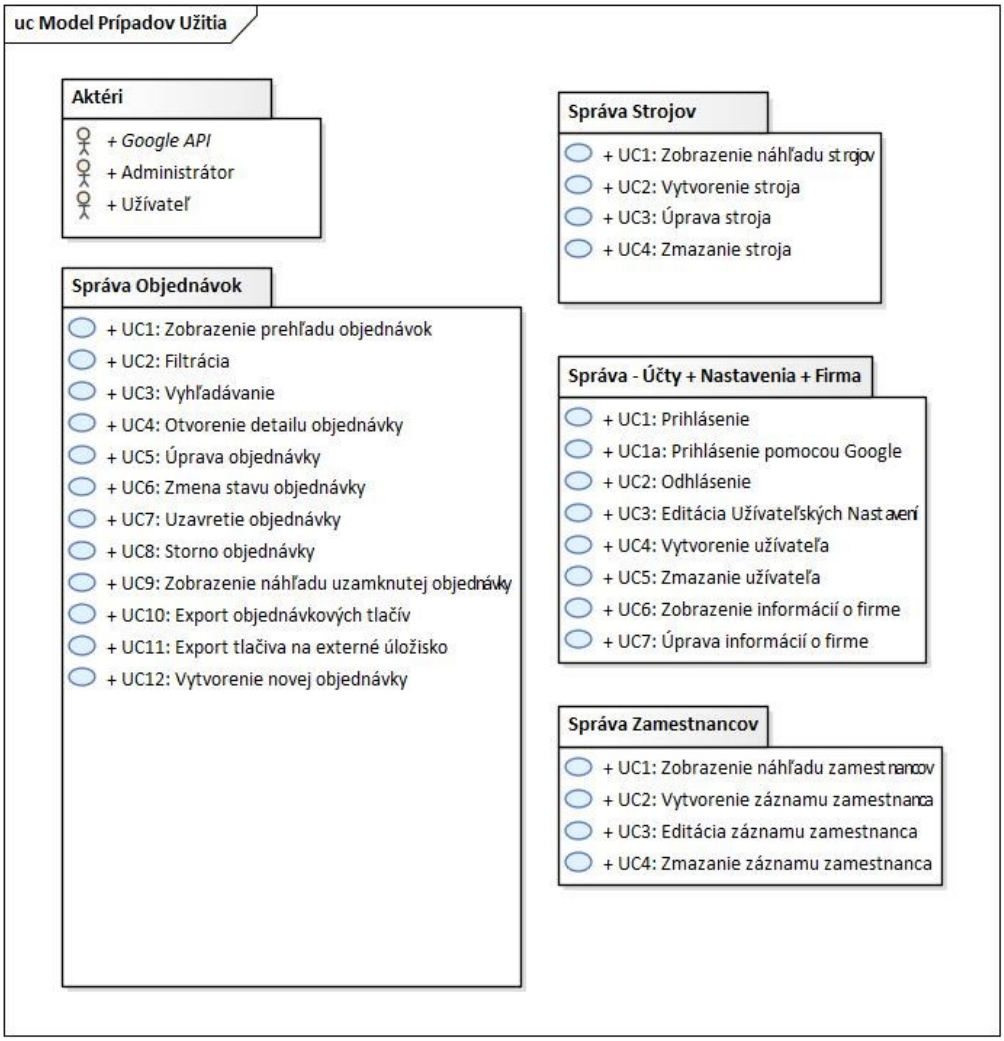
- **Aktéri:**

  Aktér reprezentuje entitu, ktorá so systémom určitým spôsobom interaguje, resp. komunikuje. *Aktéri* môžu mať vlastnú hierarchiu vzťahov popisujúcu ich role z pohľadu modelovaného systému. *Aktérom* nemusí byť výhradne človek – alternatívne ním môže byť organizácia, zariadenie či dokonca iný systém.

- **Vzťahy:**

  Podobne ako v iných diagramoch založených na špecifikácii *UML*, aj diagram *Prípadov použitia* ponúka viacero vzťahov umožňujúcich bližšie upresnenie, akým spôsobom spolu jednotlivé *Prípady použitia* súvisia. Vzťahy popisujú behaviorálne a štrukturálne väzby medzi prvkami modelu.

Diagramy *Prípadov použitia* sú obľúbeným prostriedkom využívaným k abstrahovaniu a zjednodušeniu odovzdávania informácií medzi business analytikmi a vývojármi – zhusteným, ale výstižným spôsobom sprostredkúvajú interakcie užívateľov a externých systémov s vytváraným software a tým poskytujú istú formu popisu slúžiaceho ako základ pre tvorbu procesného toku pri neskoršej implementácii systému [190, 191]. *Enterprise Architect* v kontexte tvorby diagramov *Prípadov použitia* umožňuje k jednotlivým *Prípadom použitia* priložiť *Scenáre* ďalej upresňujúce očakávaný sled interakcií *Aktéra* a systému. *Scenáre* umožňujú podrobný rozpad *Prípadu použitia* ako celku na elementárne a dobre predstaviteľné kroky. Tieto *Scenáre* môžu zahŕňať ako očakávaný / konvenčný priebeh interakcie, tak – najmä v prípade komplexnejších *Prípadov použitia* - alternatívne či chybové vetvy popisujúce prípadné výnimočné a neočakávané situácie súvisiace s interakciou popisovanou P*rípadom použitia* ku ktorému je *Scenár* priradený.



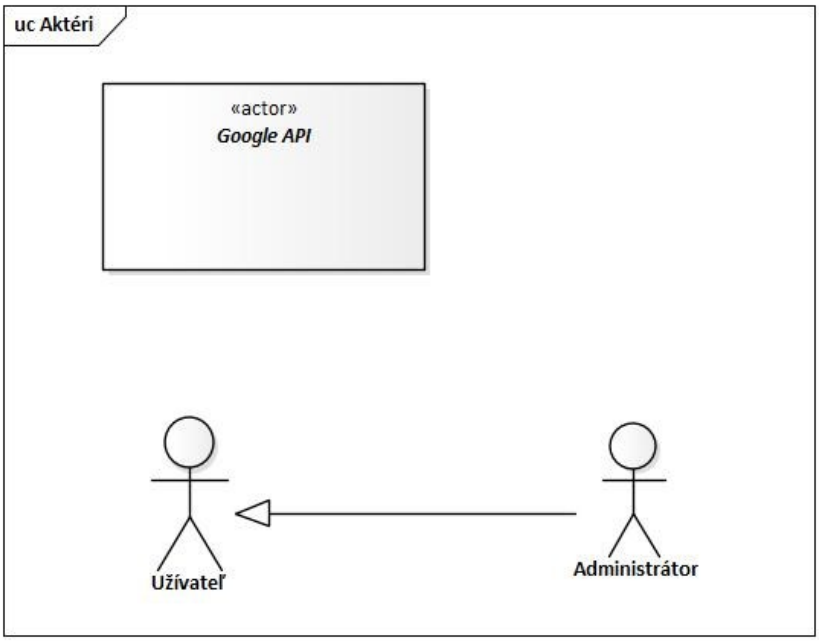
Obrázok 26. Prehľad a členenie prípadov použitia v rámci navrhovaného systému

5.2.2.1 Aktéri

Modelovaný systém pre správu objednávok v oblasti *Aktérov* disponuje pomerne jednoduchou štruktúrou. Základná myšlienka pracuje iba s dvoma užívateľskými rolami, a síce rolou bežného užívateľa – zamestnanca, pristupujúceho iba k základným funkcionalitám systému, ako sú napr. manipulácia s objednávkami. Druhou rolou je rola administrátorská – táto má prístup k rozšíreným možnostiam systému, čo zahŕňa napr. správu zamestnancov a výrobných zariadení či manipuláciu s globálnymi nastaveniami platnými pre systém ako celok.

Grafická reprezentácia *Aktérov* tento vzťah zachytáva prostredníctvom vzťahu *Generalizácie* [192] – Základné funkcionality prístupné užívateľovi bez špeciálnych oprávnení sú prístupné aj administrátorovi aplikácie, nie však naopak.

Náhľad *Aktérov* naviac odhalí *Aktéra* softwarového charakteru – *Google API*. Systém pre správu objednávok využíva viacero funkcionalít, ktoré komunikujú prostredníctvom aplikačných rozhraní spoločnosti *Google* – patria sem akcie zastrešujúce autentifikačné mechanizmy, elektronickú komunikáciu, či prácu s dokumentami na vzdialenom úložisku *Google Drive*.



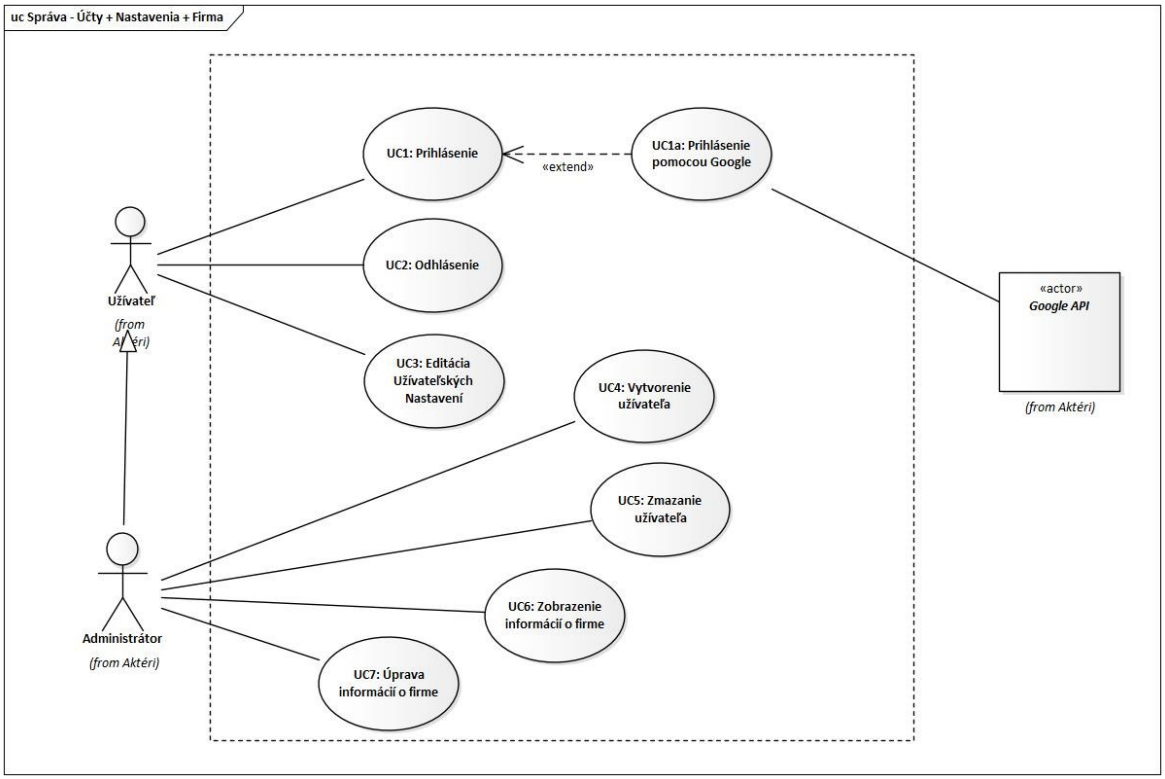
Obrázok 27. Aktéri modelovaného systému vrátane systémového aktéra Google API

5.2.2.2 Funkcionalita

Funkcionalita zachytávaná modelom *Prípadov použitia* bola rozdelená na pod-moduly na základe základných logických celkov, ktorými je navrhovaný systém tvorený. Každý z týchto pod-modulov je reprezentovaný vlastným diagramom. Samozrejme by bolo možné realizovať celý *UC* model systému ako jediný veľký diagram, tento prístup by však pravdepodobne nemal žiaden reálny prínos za cenu straty prehľadnosti.

Interakcie s aplikáciou sú rozdelené pomocou nasledujúcich UC diagramov:

- **Správa – Účty + Nastavenia + Firma:**
  Pomocný modul, zahŕňa možnosti užívateľského nastavenia všeobecného správania aplikácie, zastrešuje autorizačné a autentifikačné mechanizmy (vrátane popisu, akým spôsobom je vlastne zamestnanecký účet s prístupom do systému vytváraný) a taktiež obsahuje *UC* popisujúce manipuláciu administrátora s globálnymi firemnými nastaveniami.

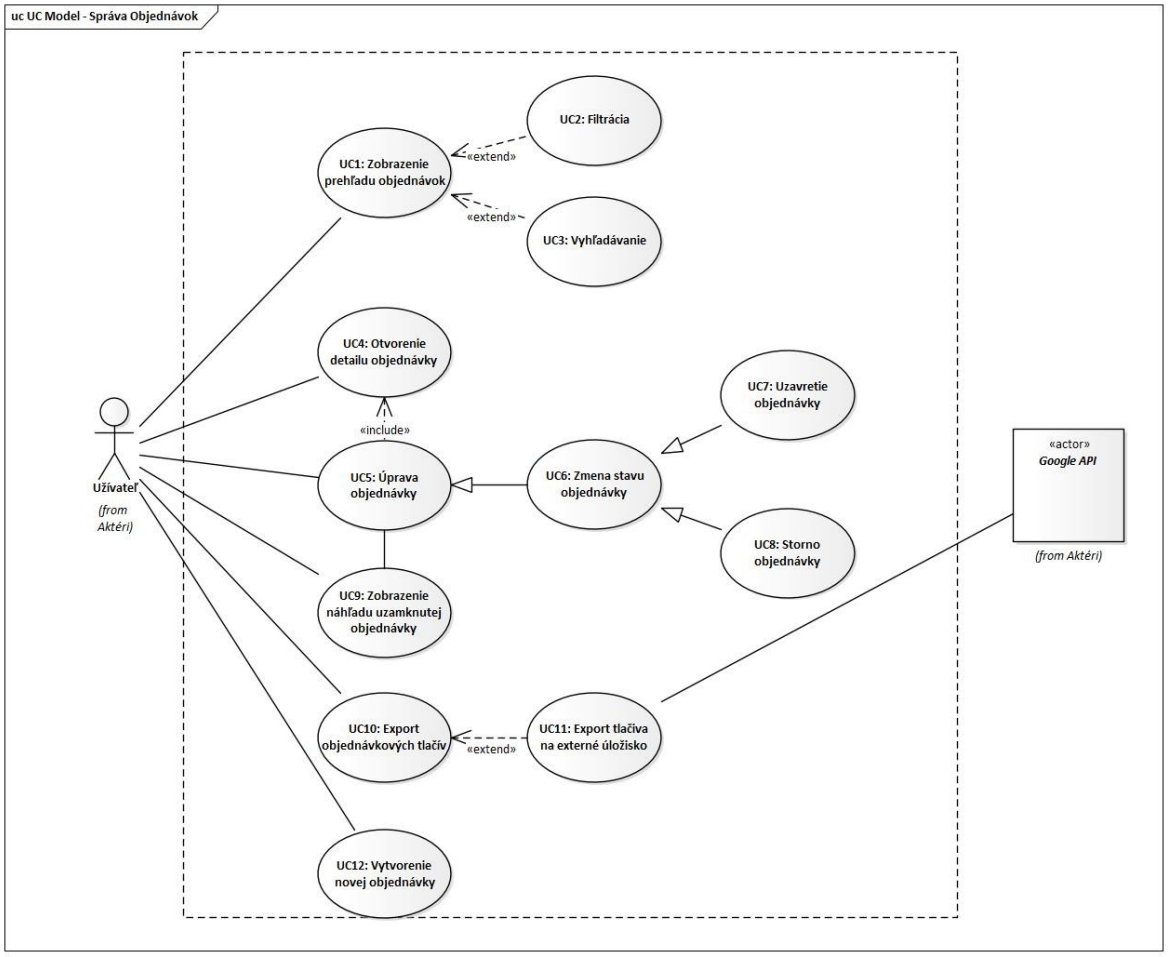


Obrázok 28. Diagram prípadov použitia, modul „Správa – Účty + Nastavenie + Firma“

- **Správa objednávok:**

  Správa objednávok je jednoznačne najväčším logickým celkom (a zároveň najrozsiahlejším diagramom *Prípadov použitia* vytvoreným v rámci analýzy a modelovania tohto systému). Popisuje prácu užívateľa s objednávkami ako takými, pričom okrem základnej *CRUD* funkcionality nad entitami objednávok sprostredkúva aj *Prípady použitia* pokrývajúce súvisiace úkony – nevyhnutné schopnosti filtrácie či rýchleho vyhľadávania, radenie, zmena stavov a procesy súvisiace s exportom. resp. ďalším spracovaním objednávkových tlačív sú integrálnou súčasťou internej firemnej administratívy a v žiadnom prípade nesmú byť v rámci analýzy opomenuté.

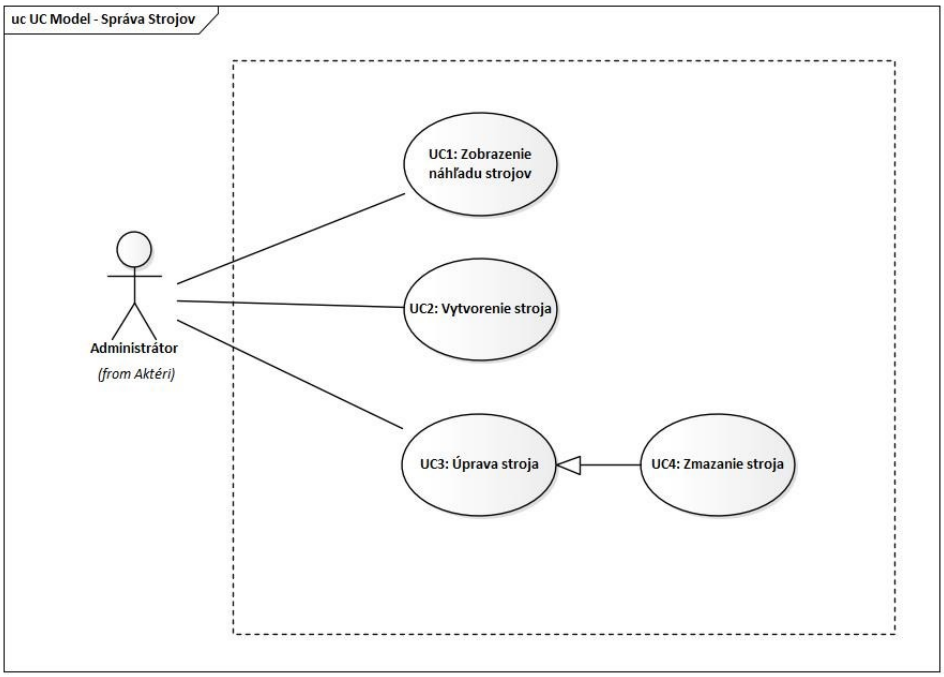
  Modul správy objednávok u mnohých kvetnatejších *Prípadov použitia* obsahuje rozsiahle *Scenáre* vrátane alternatívnych / chybových priechodov v snahe vopred zamedziť neošetreným či problémovým situáciám tým, že boli v analytickej fáze korektne predvídané a zdokumentované.



Obrázok 29. Diagram prípadov použitia, modul „Správa Objednávok“

- **Správa strojov:**

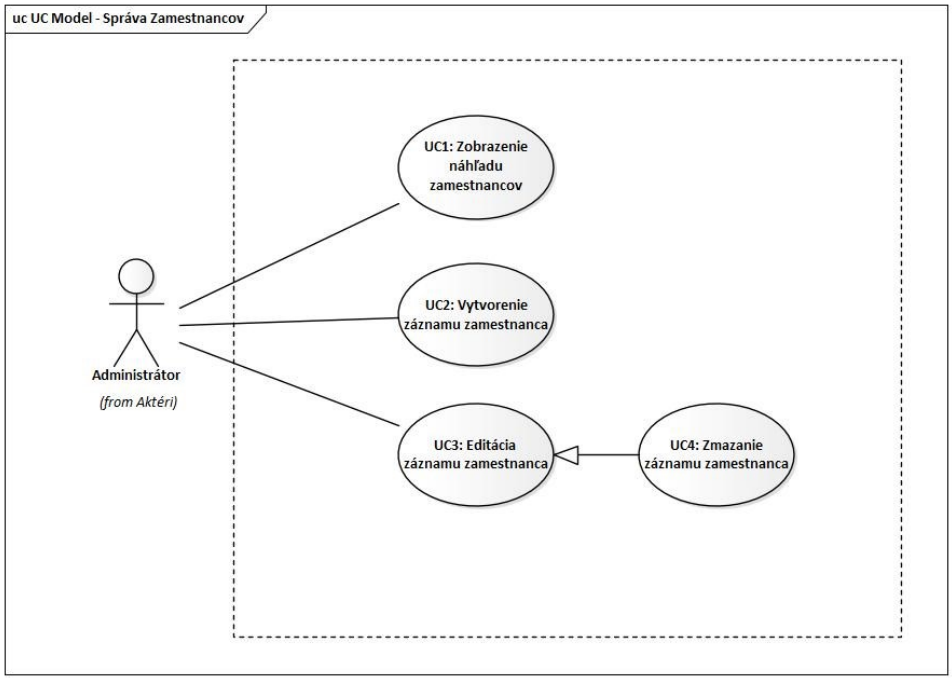
  Diagram *Prípadov použitia* zaoberajúci sa manažmentom strojov a výrobných zariadení popisuje akcie, ktoré je schopný aktér s dostatočnými oprávneniami vykonávať v súvislosti s evidenciou a administratívou výrobných zariadení či strojov v rámci firmy – v tomto prípade sa jedná prakticky výhradne o operácie typu *CRUD*.



Obrázok 30. Diagram prípadov použitia, modul „Správa Strojov“

- **Správa zamestnancov:**

  Diagram *Prípadov použitia* zaoberajúci sa manažmentom zamestnancov popisuje akcie, ktoré je schopný *Aktér* s dostatočnými oprávneniami vykonávať v súvislosti s evidenciou a administratívou zamestnancov a ich informácií v rámci firmy – v tomto prípade sa jedná prakticky výhradne o operácie typu *CRUD*.



Obrázok 31. Diagram prípadov použitia, modul „Správa Zamestnancov“

5.2.3 Model tried

Diagram *Tried* tak, ako ho vníma *UML* je statickým, štrukturálnym typom diagramu, ktorý popisuje *Triedy* ako stavebné bloky funkcionality v zmysle objektovo orientovaného náhľadu na modelovaný systém, pričom sám je grafickou notáciou užívanou ku konštrukcii a vizualizácii týchto systémov [193]. Základný koncept diagramu *Tried* sa takmer vôbec nelíši od poňatia triedy v smere objektovo orientovaného prístupu k programovaniu – niektoré zdroje uvádzajú, že diagram *Tried* by mal byť závislý na zvolenej implementačnej platforme a tým pádom reprezentovať kompletnú a funkčnú reprezentáciu *Modelov* v prípade ich prepísania do kódu [193, 194].

Triedu v ponímaní diagramu *Tried* je v súlade s *OOP* paradigmou možné vnímať ako predpis či šablónu definujúcu vlastnosti a operácie príslušné všetkým objektom (inštanciám), ktoré z tejto *Triedy* boli odvodené. Jednotlivé objekty zdieľajú množinu vlastností a metód definovaných *Triedou*, avšak hodnoty atribútov, ktorými jednotlivé objekty disponujú, sa medzi sebou vzájomne môžu rôzniť.

[193] UML definuje triedu troma základnými vlastnosťami:

- **Názov:**
  Každá *Trieda* disponuje vlastným názvom, ktorým je definovaná v rámci modelu – a v dôsledku toho v rámci systému ako celku. Názov *Triedy sa* uvádza v hornej časti grafickej reprezentácie *UML Triedy*. Názov je jedinou povinnou súčasťou *Triedy* v rámci *UML* notácie.
- **Stav (Atribúty):**
  Trieda definuje sadu vlastností, ktorými disponuje každá inštancia vytvorená na základe jej predpisu. Typ a rozsah informácií udávaný pri jednotlivých atribútoch závisí následne na metodike zvolenej pri vypracovávaní konkrétneho diagramu. Pokiaľ bol zvolený prístup notácie so signatúrou, uvádzajú sa pri jednotlivých atribútoch naviac aj dátové typy.
  Atribúty umožňujú určiť aj tzv. modifikátor prístupu, ktorý určí do akej miery sú jednotlivé vlastnosti viditeľné / manipulovateľné z prostredia mimo *Triedy* samotnej.
  Modifikátory prístupu môžu byť:
  - **Private:**
    Súkromné atribúty adresuje iba inštancia *Triedy* ktorá ich vlastní.

- **Protected:**
  Chránené atribúty adresuje iba inštancia *Triedy,* ktorá ich vlastní alebo inštancia *Triedy* z nej odvodenej prostredníctvom dedičnosti či alternatívneho analogického mechanizmu.
- **Public:**
  Verejné atribúty je možné vidieť a pristupovať ku nim aj z prostredia mimo *Triedy*.

- **Správanie (Metódy):**
  Definujú akcie (funkcie), o ktorých má inštancia *Triedy* povedomie a dokáže ich využívať. V *UML* sú určené svojim názvom, parametrami a návratovým typom. Podobne ako atribúty umožňujú obmedziť viditeľnosť z prostredia mimo kontext *Triedy* pomocou aplikácie vhodného prístupového modifikátora.

5.2.3.1 Implementácia

Implementácia diagramu *Tried* pre potreby modelovaného systému sa riadi niekoľkými špecifikami vzhľadom na povahu modelovaného systému a účel, ktorý si diagram *Tried* v tomto smere kladie za cieľ naplniť.

- **Modely:**
  Modely vytvoreného diagramu *Tried* reprezentujú dátové modely, s ktorými manipuluje *Backend* aplikácie. Neobsahujú teda atribúty reprezentujúce vizualizačnú logiku. Vzhľadom na to, že sa jedná o modely dátové a nie o modely implementujúce business logiku systému, neobsahujú žiadne vlastné metódy – za manipuláciu s dátovými *Triedami* a ich spracovanie sú zodpovedné iné komponenty softwarového riešenia. Tento prístup k tvorbe diagramu *Tried* bol zvolený z dôvodu, že implementácia *Tried* aplikačnej logiky v prípade modelovaného systému nie je programátorsky náročná – potenciálne náročné však je skonsolidovať tvar spracúvaných dát do uceleného dátového modelu.
- **Identifikátory:**
  Otázka, či má model *Tried* zahŕňať identifikátory modelovaných entít nie novinkou. Obecný konsenzus je, že modely *Tried* môžeme deliť na rôzne podkategórie na základe toho, ktorý aspekt systému zachytávajú [195]:

- **Analytické modely tried:**
  Analytické *Triedne* modely popisujú doménu business problému bez zohľadnenia konkrétnych technológií a infraštruktúrnych riešení.
- **Dizajnové modely tried:**
  Model *Tried* najvernejší reprezentácii problému prostredníctvom kódu, zobrazuje elementárne komponenty programátorského riešenia a väzby medzi nimi.
- **Modely tried dátového modelovania:**
  Podobné diagramom vzťahov medzi entitami (*ERD – Entity Relationship Diagrams*). Modelovanie prebieha z pohľadu databázy, obsahujú informácie o primárnych / cudzích kľúčoch a pod.

Vytvorený diagram *Tried* je kombináciou dizajnového a dátového prístupu k tvorbe tohto typu diagramu. Popisuje dátové modely z pohľadu kódu a zároveň využíva dátové typy a štruktúry odrážajúce zvolené technológie, ale vzhľadom na využitie objektovo-relačného mapovania sprostredkovaného technológiou *Entity Framework* naznačuje, že jednotlivé objekty disponujú vlastnými unikátnymi identifikátormi a zároveň je logika ich prepojenia neoddeliteľnou súčasťou dátovej reprezentácie využívanej na *Backend*-e aplikácie.

- **Modifikátory prístupu:**
  Vyššie popísaný hybridný prístup k tvorbe modelu *Tried* vedie k tomu, že v drvivej väčšine prípadov *Triedy* neskrývajú svoje atribúty – všetky atribúty nutné k zabezpečeniu bezproblémovej implementácie business logiky aplikácie sú označené ako verejné, pričom privátnymi atribútmi chápeme atribúty reprezentujúce kľúče, identifikátory, či atribúty iným spôsobom zabezpečujúce mapovanie *Tried* (resp. entít ktoré reprezentujú) na dostupné databázové štruktúry.

[Model popisujúci triedy v systéme a vzťahy medzi nimi:

viď. *Príloha č. I → DP_EAP_JozefKovac_A20132.eapx → „Model – Triedy“*]

5.2.4 Drôtené modely

Drôtený model (*Wireframe*) vnímame ako predbežný, konceptuálny návrh grafického užívateľského rozhrania, ktorý je prostriedkom k zmysluplnej komunikácii ohľadom kľúčových otázok týkajúcich sa tvorby vizuálneho aspektu aplikácie. Drôtený model je vlastne náčrtom či nákresom nesúcim údaje o štruktúre užívateľského rozhrania [196] – podľa miery detailu vloženej do návrhu drôtených modelov sa môžu pohybovať od jednoduchých, bezfarebných náčrtkov informujúcich o údajoch obsiahnutých v rámci webovej stránky a ich vzájomnom rozmiestnení až po komplexné, interaktívne, profesionálne vypracované kompletné rekonštrukcie navrhovaného riešenia grafického rozhrania. Hlavným dôvodom, prečo sú drôtené modely vyrábané pred samotným *GUI* je jednak potreba dohodnúť sa so zákazníkom na prípadných nekorektných / prebytočných / nesprávne umiestnených / chýbajúcich ovládacích prvkoch, jednak motivácia v podobe množstva ušetreného času a peňazí – prekreslenie návrhu vo včasnej fáze vývoja je omnoho jednoduchšie ako dodatočné úpravy počas prebiehajúcej implementácie, kedy už na môžu byť zmeny v užívateľskom rozhraní komplikované naviazanou logikou [196, 197].

[197] Drôtené modely podľa vernosti očakávanému softwarovému výstupu delíme na:

- **Jednoduché:**
  Hrubá kostra aplikácie, prevažne načrtnuté rukou alebo prostredníctvom vhodného digitálneho nástroja. Preferujú absolútny minimalizmus – čiernobielu farebnú schému, generickú typografiu, žiadne obrázky či efekty.
- **Stredne komplexné:**
  Bližšie kopírujú zamýšľaný finálny design produktu, môžu implementovať obrázky, efekty, interakcie medzi stránkami či prepojenia elementov zobrazenia.
- **Komplexné:**
  Vysoko komplexné drôtené modely sa snažia kopírovať zamýšľané UI vytváraného produktu tak blízko, ako je len možné. Ide o plný návrh vrátane obrázkov, ikoniek, farieb, fontov a ovládacích prvkov prepojených tým spôsobom, aby simulovali funkčný prototyp.

Design tvoreného systému pre správu objednávok bol navrhnutý s využitím typu drôteného modelu, ktorý je podľa vyššie uvedeného členenia niekde na pomedzi medzi jednoduchým a stredne komplexným variantom – jedná sa o konceptuálny návrh v čiernobielej farebnej schéme zameraný na prítomné vizualizačné / ovládacie prvky a ich vzájomné rozloženie.

Neumožňuje prechody medzi stránkami a nedisponuje podobnou pokročilou funkcionalitou, ale mnoho podkladovej logiky nenásilným spôsobom vizuálne naznačuje – napr. prítomnosť obmedzení kladených na vstupné polia, prítomnosť užívateľskou rolou podmieneného navigačného menu či typy vstupov.

Vypracovanie predbežného návrhu grafického rozhrania systému pre správu objednávok v implementačnej fáze práce viedlo k značnej časovej úspore, keďže vzhľadom na vopred načrtnuté, prediskutované a schválené návrhy rozhranie nebolo nutné vymýšľať počas konštrukcie - vývoj tejto časti aplikácie bol efektívne zredukovaný na jednoduché prepisovanie cieľovej podoby dizajnu do kódu vedúceho k požadovanému výsledku.

Podobne ako v ostatných častiach návrhu aplikácie je možné rozdeľovať vypracované drôtené modely do kategórií podľa logických sekcií business logiky, ktorú podľa nich skonštruovaná časť *GUI* užívateľovi sprostredkúva. Tieto moduly sú pre potreby drôtených modelov určené ako: Správa zamestnancov, správa strojov, správa objednávok, nastavenia systému a informácie o firme.

5.2.4.1 Správa zamestnancov

Obrázok 32. Návrh grafického užívateľského rozhrania časti aplikácie zodpovednej za správu zamestnancov

Grafické užívateľské rozhranie obsiahnuté v module správy zamestnancov je rozdelené na dve primárne časti, pričom ich viditeľnosť, resp. rozsah je výrazne ovplyvňovaný aktuálnym stavom zamestnancov evidovaných v rámci systému.

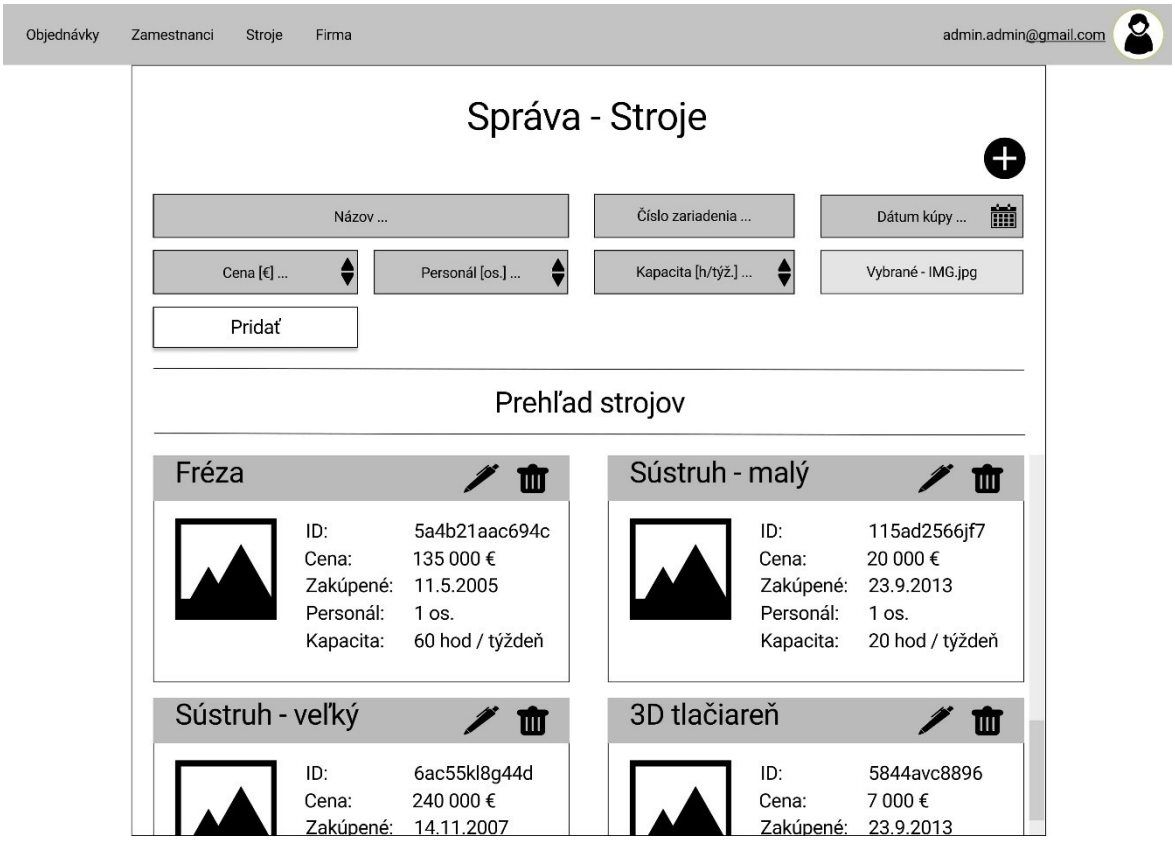
Pridávací formulár

Horná časť grafického rozhrania určeného k správe zamestnaneckých účtov obsahuje pridávací formulár prostredníctvom ktorého je administrátor schopný pridať nového zamestnanca. Účelom ikonky „+“ je skladateľnosť tohto formulára, ktorý je pri načítaní stránky skrytý (po kliknutí na ikonku dochádza k jeho rozbaleniu / zbaleniu).

Karty zamestnancov

Prehľad zamestnancov v spodnej časti stránky zobrazuje náhľad údajov už evidovaných zamestnancov v zmysle základných údajov a profilovej fotografie. V pravej hornej časti kartičky sa následne nachádzajú rýchle akcie umožňujúce výmaz, resp. editáciu záznamu zamestnanca. Prípadná editácia prebieha na samostatnom vyskakovacom formulári.

5.2.4.2 Správa strojov



Obrázok 33. Návrh grafického užívateľského rozhrania časti systému zodpovednej za správu výrobných zariadení

Grafické užívateľské rozhranie obsiahnuté v module správy výrobných strojov či zariadení disponuje (veľmi podobne ako u modulu evidencie zamestnancov) dvomi základnými celkami popísanými v nasledujúcich odstavcoch.

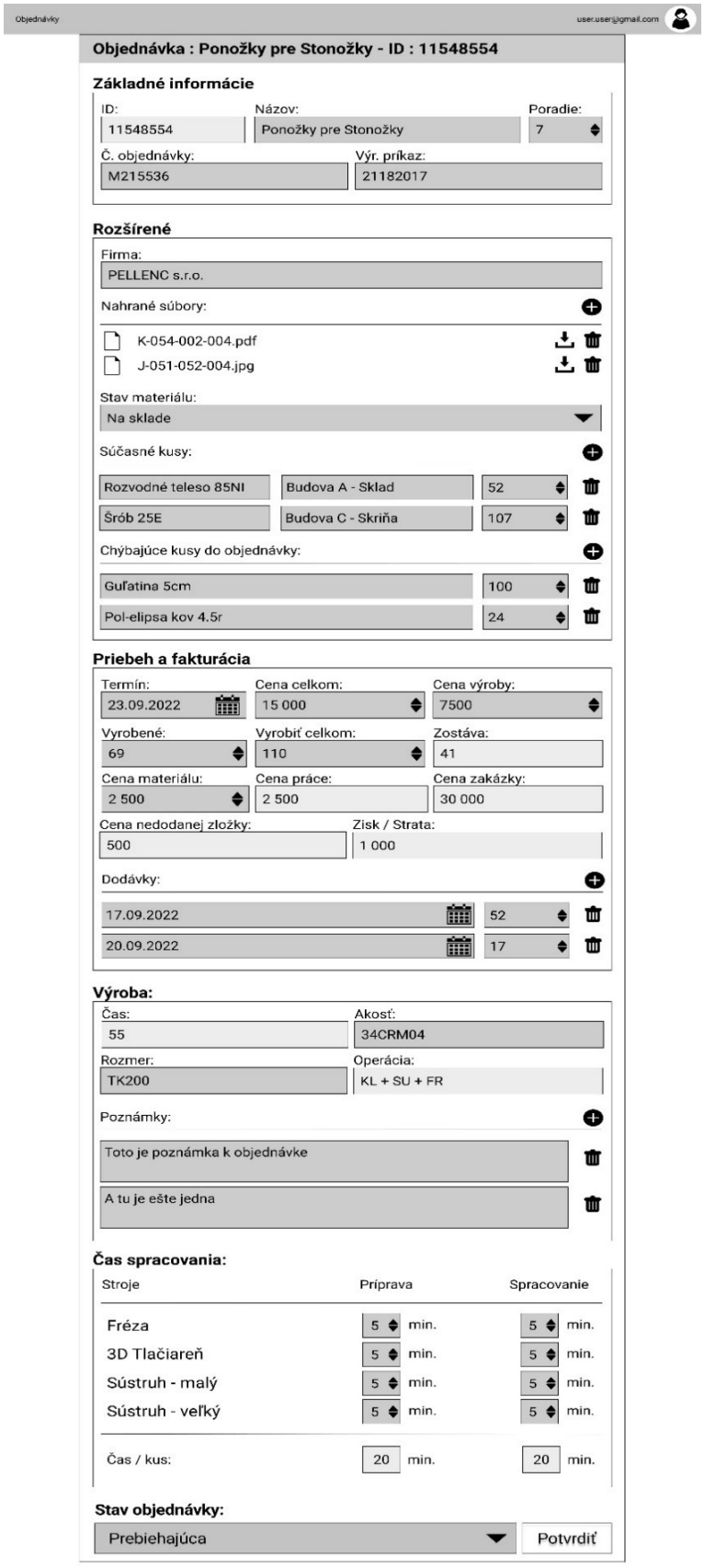
Pridávací formulár

Vrchná časť grafického rozhrania správy strojov disponuje pridávacím formulárom obsahujúcim všetky potrebné údaje pri pridanie nového zariadenia do evidencie v rámci systému, vrátane možnosti naviazania náhľadovej fotografie. Ikonka „+“ slúži k možnosti zbalenia / rozbalenia formulára podľa aktuálnych potrieb užívateľa.

Karty strojov

Prehľad výrobných zariadení v spodnej časti stránky disponuje kartičkami s náhľadovým obrázkom a základnými informáciami uloženými pri jednotlivých strojoch, vrátane panelu s rýchlymi akciami pre potreby výmazu, resp. editácie existujúcej položky výrobného zariadenia. Editácia samotná využíva samostatný vyskakovací formulár.

5.2.4.3 Správa objednávok



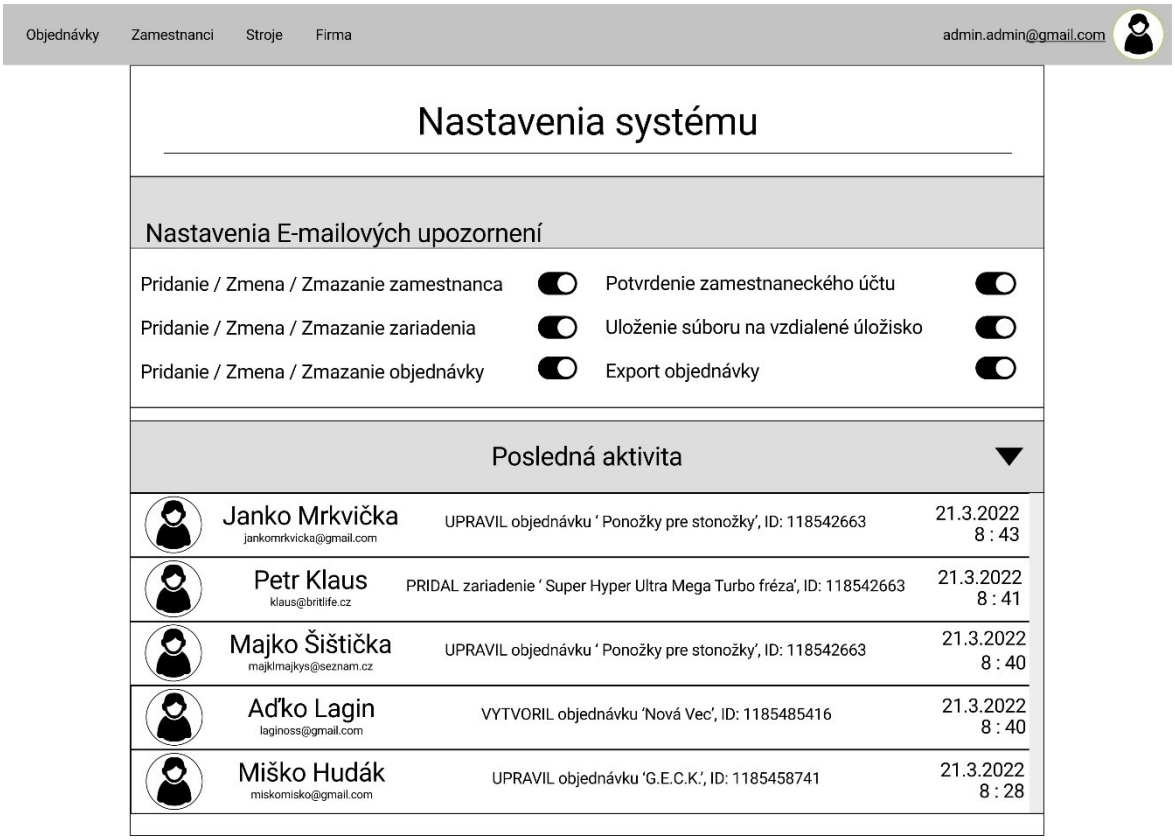
Obrázok 34. Hlavný formulár zodpovedný za manipuláciu so záznamom objednávky

Správa objednávok je súčasťou systému obsahujúcou pravdepodobne najväčší podiel celkovej funkcionality – Z toho dôvodu disponuje výnimočným dielom pozornosti aj v smere realizácie drôtených modelov. Náčrty zachytávajú hlavnú stránku s prehľadom objednávok, filtračný systém objednávok a v neposlednom rade samotný formulár určený k vytvoreniu / náhľadu / editácii objednávky (konkrétna akcia pri ktorej je užívateľovi zobrazovaný záleží prakticky výhradne na tom, v ktorom z množiny definovaných stavov sa manipulovaná objednávka momentálne nachádza). Priložený obrázok zachytáva práve podobu tohto formuláru v čase tvorby najrozsiahlejšiu monolitickú časť systému.

Formulár s údajmi o objednávke je rozdelený na viacero sekcií:

- **Základné informácie:**
  Základné informácie tvoria povinné polia nevyhnutné pre celkovú funkčnosť evidenčného systému a súvisiacich pomocných komponent ako napr. filtrovanie, vyhľadávanie či exporty objednávok.
- **Rozšírené informácie:**
  Dodatočné informácie o objednávke zachytávajú stav materiálu vrátane detailov o chýbajúcich / naskladnených položkách, dodatkovú informáciu o objednávajúcej firme a možnosti manipulácie so súbormi priloženými k tejto objednávke.
- **Priebeh a fakturácia:**
  Sekcia priebehu a fakturácie poskytuje informácie o finančných náležitostiach súvisiacich s objednávkou a dôležité termíny. Zároveň sa v nej nachádza väčšie množstvo prepočtovej logiky – pri analýze potrebných polí bolo stanovené, že nie všetky položky je nutné ukladať, niektoré stačí užívateľovi pri náhľade objednávky zobrazovať za účelom nadobudnutia rýchlych a viditeľných finančne orientovaných výstupov objednávky.
- **Výroba:**
  Sekcia výroba disponuje prostriedkami k špecifikácii detailov k objednávke, vrátane typu operácie, akosti a ďalších špecifík súvisiacich s plánovaným výrobným procesom.
- **Čas a spracovanie:**
  Posledná sekcia formulára významne závisí na ďalších prvkoch systému v smere aktuálneho stavu výrobných zariadení – umožňuje previazať objednávku s údajmi o evidovaných strojoch kvôli potrebám evidencie predpokladanej náročnosti časového spracovania.

5.2.4.4 Nastavenia systému



Obrázok 35. Návrh grafického užívateľského rozhrania časti aplikácie obsahujúcej možnosti užívateľského nastavenia a prehľad aktivity

Stránka systémových nastavení je prístupná s využitím vrchnej navigačnej lišty (skrytá pod ikonkou práve prihláseného užívateľa). Obsahom tejto webovej stránky sú dve hlavné časti zastrešujúce užívateľské nastavenia a monitoring aktivít v rámci systému ako celku.

Nastavenia upozornení

Nastavenie E-mailových upozornení je sekcia umožňujúca užívateľovi špecifikovať udalosti v rámci systému, pri ktorých má byť notifikovaný prostredníctvom automaticky odosielanej E-mailovej správy.

Posledná aktivita

Prehľad aktivity umožňuje užívateľovi disponujúcemu administrátorskou úlohou nahliadať na stránkovaný, formátovaný výpis poslednej aktivity ku ktorej v rámci systému došlo, vrátane informácie o dátume a čase, stručných údajoch určujúcich autora aktivity a krátkym popisom povahy vykonanej akcie. Výpis logu aktivít je pri navigácii na stránku zbalený - až do momentu interakcie užívateľa s príslušným ovládacím prvkom.

5.2.4.5 Informácie o firme

Obrázok 36. Návrh grafického užívateľského rozhrania časti aplikácie umožňujúcej manipulovať s firemnými údajmi

Informácie o firme predstavujú graficky nenáročnú sekciu prístupnú administrátorovi systému. Stránka sa skladá z jediného jednoduchého formulára umožňujúceho zadať do systému základné informácie, údaje o adrese sídliska firmy a kontaktné údaje súčasného konateľa firmy.

Skutočný význam a prínos tejto stránky je zrejmý v prípade snahy exportovať administratívne tlačivá objednávok. Tlačivá nezriedka obsahujú údaje o firme, resp. kontakty na zodpovednú osobu. Možnosť uloženia a editácie týchto dát prostredníctvom formulára v priestore na to vyhradenej stránky s možnosťou uchovania pomocou perzistentného úložiska otvára dvere funkcionalite automatického vypĺňania. Evidencia firemných údajov umožňuje ich premietnutie vo všetkých formulároch či tlačivách, ktoré tieto dáta vyžadujú automaticky - prostredníctvom jednoduchého načítania, bez nutnosti ďalšej intervencie užívateľa.

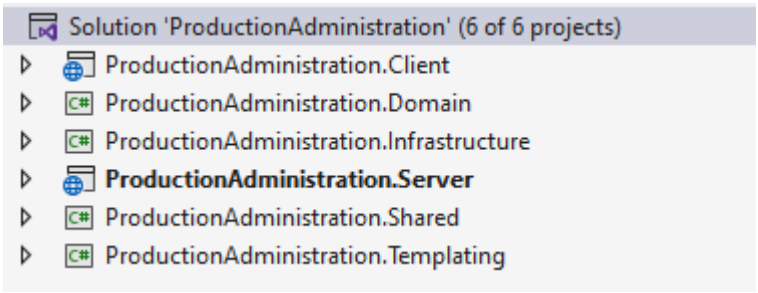
6 IMPLEMENTÁCIA SYSTÉMU

Implementácia systému pre správu objednávok prebiehala na základoch položených predchádzajúcou analytickou fázou, v rámci ktorej boli vytvorené všetky potrebné podklady pre docielenie svižného a efektívneho pracovného tempa. Laický pohľad na vytvorený systém bude pravdepodobne založený výhradne na grafickom rozhraní aplikácie – ponímanie koncového užívateľa teda pochopiteľne inklinuje ku vnímaniu jednej webovej stránky, prostredníctvom ktorej manipuluje s určitou dátovou množinou ako jedného uzatvoreného funkčného celku. Vzhľadom na použitú architektúru a nutnosť prenosu informácií medzi rôznymi typmi dátových entít je však prepojenie funkčných modulov na pozadí v skutočnosti komplexnejšie. Naprieč celou implementáciou bola vyvíjaná aktívna snaha držať sa v rámci kódu princípov N-vrstvovej architektúry prispôsobenej špecifikám použitých technologických riešení – špecifiká jednotlivých stavebných blokov kódu sú následne dôkladnejšie popísané v nasledujúcich kapitolách.

6.1 Projekt

Snaha udržať projekt upravený, navigovateľný a bez problému rozšíriteľný aj v budúcnosti viedla k nastaveniu stabilnej súborovej štruktúry so zameraním na jej intuitívnosť a jednoduchú adaptovateľnosť pre potreby prípadného budúceho vývoja a rozširovania.

6.1.1 Projektová štruktúra



Obrázok 37. Projektová štruktúra systému pre správu objednávok

Riešenie (*Solution*) reprezentujúce projekt sa skladá z niekoľkých zapuzdrených podprojektov, ktorých vzájomná viditeľnosť (resp. spôsob, akým medzi sebou zdieľajú dáta) je stanovená referenciami definovateľnými v príslušných projektových súboroch s príponou *.csproj* (alebo alternatívne priamo z grafického rozhrania vývojového prostredia *Visual Studio*, ktoré bolo k vypracovaniu implementačnej časti práce použité).

Obrázok demonštruje, že kostra systému sa skladá celkovo zo šiestich podprojektov. Každý podprojekt plní pevne danú sadu úloh a existuje medzi nimi systém referencií zabezpečujúcich, že pri ďalšom rozvoji bude možné architektúru používať ďalej bez nutnosti významných úprav.

Jednotlivé projekty sú nasledovné:

- **ProductionAdministration.Client:**
  Projekt obsahujúci *Frontend* aplikácie. Zahŕňa štandardný webový obsah, statické obrázky, ikonky, súbory kaskádových štýlov a skriptov – či už tie poskytované využívanými frameworkami, alebo definované v priebehu vývoja. Primárne sú v ňom zahrnuté šablóny drvivej väčšiny užívateľsky prístupných *.razor* stránok a komponent, z ktorých ich tvoria. Projekt *Client* v neposlednom rade obsahuje API služby určené na komunikáciu medzi *GUI* a *Backend*-om aplikácie.
- **ProductionAdministration.Server:**
  Projekt „*ProductionAdministration.Server"* tvorí vstupnú bránu do hĺbkovej logiky aplikácie – obsahuje triedy plniace funkciu *Controller*-ov, nástroje určené k správe *HTTP* požiadaviek a odpovedí, nástroje zabezpečujúce generovanie dokumentácie, konfigurácie mapovania *DTOs* na doménové objekty a naopak. Nakoniec je v ňom možné nájsť stránky obsluhujúce serverovú zložku rozhrania využívaného systémom *ASP.NET Identity*.
- **ProductionAdministration.Shared:**
  Projekt charakteristický pre *Blazor* v modeli nasadenia *WebAssembly*, zastrešuje všetky prvky, ku ktorým je žiadúci prístup ako z klienta, tak serveru aplikačnej architektúry. Spadajú sem všetky *DTOs* (určené na komunikáciu klienta s *API* a naopak), enumerácie zdieľané medzi *DTOs* a doménovými objektami, validačné konštanty (keďže validácia prebieha ako na strane klienta, tak na strane serveru) a konštanty prevažne odkazované buď v kontexte textácie validácií, alebo za účelom smerovania požiadaviek na vhodné koncové body.
- **ProductionAdministration.Domain:**
  Projekt obsahujúci primárne triedy, ktoré v rámci architektonického vzoru *MVC* klasifikujeme ako *Model*-y – pokrýva služby zodpovedné za business logiku aplikácie (*Services*), doménové modely s ktorými interne pracuje *Backend* a ktoré sú po zohľadnení *ORM* abstrakcie vlastne predpisom konkrétnych riadkov podkladovej relačnej databázy.

- **ProductionAdministration.Infrastructure:**

  Primárnym účelom projektu „*ProductionAdministration.Infrastructure“* je správa dát. Projekt obsahuje triedy repozitárov (*Repositories*) určených k priamej komunikácii s dátovým kontextom. Ďalej je v projekte možné nájsť konfigurácie a deklarácie vzájomných vzťahov jednotlivých doménových objektov zohľadňovaných nástrojom *Entity Framework* pri budovaní objektovo orientovanej abstrakcie nad databázovými tabuľkami.

  „*ProductionAdministration.Infrastructure“* v neposlednom rade obsahuje samotnú triedu databázového kontextu zabezpečujúcu jednotný prístup k úložisku a implementáciu princípu *Unit of Work* [viď. kapitola *3.2.3.2 – “Repozitáre – Unit of Work”*].

- **ProductionAdministration.Templating:**

  Pomocný projekt zastrešujúci rôzne typy exportov vykonávaných aplikáciou – Obsahuje potrebné modely využívané ku generovaniu dynamického obsahu rôznorodého charakteru a podporuje funkcionalitu tvorby ako parametrických *HTML* dokumentov (s využitím šablónovacieho systému *Razor*), tak využívanie a programovú modifikáciu *EXCEL* dokumentov a ich stiahnutie vo formáte *.xlsx*.

Projektovej štruktúre a hierarchii zložiek v nej obsiahnutých zodpovedá prakticky 1 : 1 aj súborová štruktúra riešenia na disku. Táto identita zaručuje jednoduchosť orientácie v rámci projektu bez ohľadu na to, z akého prostredia je k nemu pristupované.

6.1.2 Menné priestory

Dodatočné členenie funkčných celkov v rámci projektu zabezpečuje aktívne použitie menných priestorov (*Namespaces*). Menné priestory slúžia k zachovaniu prehľadnosti aj u projektov väčšieho rozsahu, pričom pomáhajú predchádzať ľudským omylom ako napr. mýlky či konflikty viacerých tried s identickým názvom.

Štruktúra menných priestorov zakomponovaných do projektu sa riadila jednoduchými konštrukčnými pravidlami a pozostávala z nasledujúcich častí (v rovnakom poradí):

1) Prefix „*ProductionAdministration*“ ako názov všetkých funkčných celkov spadajúcich do riešenia administratívneho systému.
2) Názov podprojektu, v ktorom sa menný priestor nachádza – t.j. v prípade, že sa nachádza v šablónovacom projekte bude plná forma prefixu menného priestoru „*ProductionAdministration.Templating*“.
3) Hierarchický priechod súborovou štruktúrou daného projektu – resp. postupnosť jednotlivých priečinkov, ktorej nasledovaním je možné dopracovať sa k súborom dekorovaným určitým názvom menného priestoru.

Vyššie uvedené sekvencie reťazcov spolu - po previazaní bodkami - vytvárajú unikátny identifikátor menného priestoru v rámci projektu.

```
using AutoMapper;
using ProductionAdministration.Domain.Entities.Logs.Enums;
using ProductionAdministration.Domain.Entities.Orders;
using ProductionAdministration.Domain.Infrastructure.Exceptions.Statuses;
using ProductionAdministration.Domain.Models.Orders.Pagination;
using ProductionAdministration.Domain.Repositories.Companies;
using ProductionAdministration.Domain.Repositories.Machines;
using ProductionAdministration.Domain.Repositories.Orders;
using ProductionAdministration.Domain.Services.Files;
using ProductionAdministration.Domain.Services.Logging;
using ProductionAdministration.Domain.Services.Notifications;
using ProductionAdministration.Domain.Services.Orders.Export;
using ProductionAdministration.Shared.Data.Dtos.Files;
using ProductionAdministration.Shared.Data.Dtos.Orders;
using ProductionAdministration.Shared.Data.Dtos.Orders.Pagination;
using ProductionAdministration.Shared.Data.Enums.Files;
using ProductionAdministration.Shared.Data.Enums.Orders.Export;
using ProductionAdministration.Shared.Data.Enums.Orders.Export.Utils;
using ProductionAdministration.Shared.Infrastructure.Utils.Enums;
```

Zdrojový kód 3. Zahrnutie odkazov na rôzne menné priestory s cieľom pristupovať v nich k obsiahnutej funkcionalite

6.2 Frontend

Frontend vytvorenej webovej aplikácie bol konštruovaný účelovým spôsobom a nespoliehal na ťažké využitie výkonnostne náročných vývojových frameworkov či prebytku kaskádových štýlov – jeho funkcia je primárne zastávať informačnú a riadiacu úlohu, čo sa odrazilo aj naprieč celou dizajnovou filozofiou. Snaha autora bola skladať *Frontend* z dobre identifikovateľných a samo-popisných prvkov, ktorých myšlienka by užívateľovi mala byť intuitívne zrejmá a ktorých hlavným účelom je sprostredkovať rýchlu a efektívnu manipuláciu s prezentovanými dátami.

6.2.1 Konfigurácia

Konfigurácia *Frontend*-u je umiestnená do domény projektu „*ProductionAdministration.Client*“, konkrétne do súboru „*Program.cs*“. Oproti konfigurácii nástrojov situovaných v sfére *Backend*-u je nastavenie klienta relatívne jednoduché. Skonštruovaná aplikácia v zmysle konfigurácie obsahuje niekoľko základných blokov:

- Vytvorenie továrne na *HTTP* klienta komunikujúceho so serverom a jeho uspôsobenie zvoleným autorizačným mechanizmom
- Registrácia *API* služieb (*API Services*) do *Dependency Container*-u
- Evidencia mapovacích profilov a registrácia mapovacieho nástroja do *Dependency Container*-u
- Konfigurácia *Frontend*-ovej knižnice *Blazorise* využívanej ku konštrukcii grafického užívateľského rozhrania systému

```
1   var builder = WebAssemblyHostBuilder.CreateDefault(args);
2   builder.RootComponents.Add<App>("#app");
3   builder.RootComponents.Add<HeadOutlet>("head::after");
4
5   SetupConnectionHandling();
6   SetupServices();
7   SetupUtils();
8   SetupMappings();
9   SetupUIEnhancements();
10
11  await builder.Build().RunAsync();
```

Zdrojový kód 4. Hlavný obsah triedy „Program.cs“ v sekcii klienta aplikácie

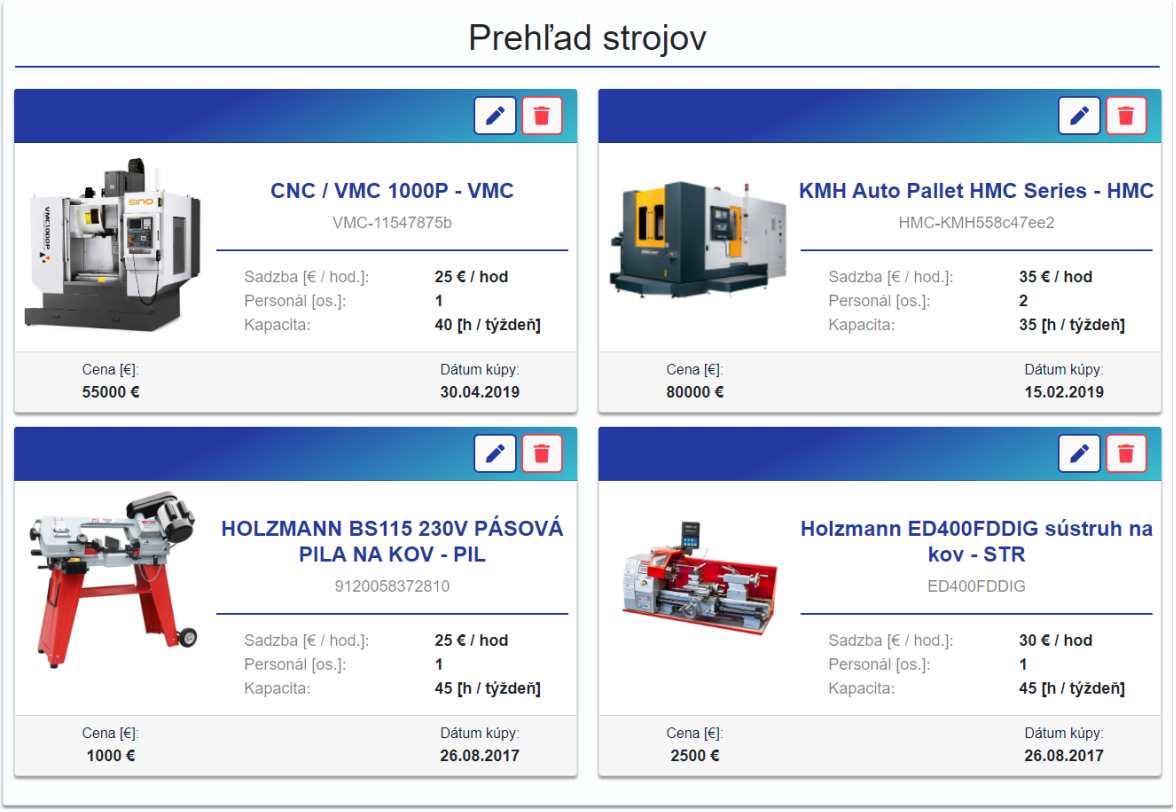
6.2.2 Dizajn

Dizajn grafického rozhrania aplikácie je nezanedbateľným elementom každého užívateľsky prívetivého programu – samozrejme sa v prípade vizuálne príjemného dizajnu jedná o spôsob, ako užívateľovu prácu s aplikáciou spraviť znesiteľnejšou. Nosné prvky dizajnu vytvoreného systému boli koncipované v snahe udržať vyvážený pomer grafickej prívetivosti a jednoduchosti v zmysle logického rozloženia väčšiny kľúčových ovládacích prvkov.

6.2.2.1 Vizuálny štýl

Vizuálny štýl grafického rozhrania obecne bol vzhľadom na primárne zameranie systému – správu objednávok výrobnej firmy – príhodne ladený do chladnejších farieb. Farebným tónom dominuje jemná modrá, výraznejšia tmavšia modrá (primárna farba), pričom farbou zvýraznenia je kontrastná tyrkysová. Ďalej je vo vizuálnej prezentácii softwaru možné nájsť ďalšie, prevažne výraznejšie farby – tieto v súlade s farebnou filozofiou *Frontend*-ového frameworku *Bootstrap* slúžia pre farebné kódovanie rôznych typov akcií či informácií (notifikácia, nebezpečenstvo, varovanie...).

Rozloženie, pozícia a orientačné veľkosti prvkov boli koncipované na základe drôtených modelov zhotovených v analytickej fáze praktickej časti práce. Drobné sporadicky sa vyskytujúce odchýlky je možné pripísať prevažne technologickým obmedzeniam, nutnosti pridania kontrolných prvkov, ktorých potrebnosť bola odhalená až pri implementácii či revíziám rozloženia v záujme zachovania použiteľnosti *GUI* na menších zobrazovacích plochách.



Obrázok 38. Náhľad vizuálnej identity aplikácie – prehľad výrobných zariadení [Obrázky prevzaté z: 198, 199, 200, 201]

6.2.2.2 Knižnica Blazorise

Knižnica *Blazorise* stavia na výhodách frameworku *Blazor,* konkrétne na jeho schopnosti objektového zapuzdrenia častí grafického rozhrania do samostatných a opakovane použiteľných komponent. Motiváciou knižnice je poskytnúť množinu pred-implementovaných komponentov zameraných na riešenie častých problémov webového vývoja (akordeóny, posuvníky...), ktorých integrácia do *Blazor* aplikácie je omnoho jednoduchšia ako vytváranie vlastných komponentov úplne od základu [202].

Blazorise dokáže poskytovať objektovo orientovanú abstrakciu nad viacerými populárnymi *CSS* frameworkami ako napr. *Bootstrap*, *Bulma* či *Material*.

```
<Fields>
    <Column ColumnSize="ColumnSize.Is2.OnWidescreen">
        <Button Color="Color.Danger"
                Class="btn btn-danger btn-sized full-width"
                TextWeight="TextWeight.Bold"
                Clicked="@CleanFilterSettings">
            Vyčistiť filter
        </Button>
    </Column>
    <Column ColumnSize="ColumnSize.Is8.OnWidescreen"
            TextAlignment="TextAlignment.Center">
        <Text>
            Počet položiek zodpovedajúcich zadanému nastaveniu vyhľadávania:
        </Text>
        <Text TextWeight="TextWeight.Bold">
            @(Pagination.TotalCount).
        </Text>
    </Column>
    <Column ColumnSize="ColumnSize.Is2.OnWidescreen">
        <Button TextWeight="TextWeight.Bold"
                Color="Color.Info"
                Class="btn btn-info btn-sized full-width"
                Clicked="@(() => (LoadOrders(null, true)))"
                Disabled="@filteringInProgress">
            @if (filteringInProgress)
            {
                <SpinnerButton DisplayText="@("Filtrujem")" />
            }
            else
            {
                @("Filtrovať")
            }
        </Button>
    </Column>
</Fields>
```

Zdrojový kód 5. Štruktúra dynamickej webstránky využívajúcej rozhranie založené na komponentoch knižnice Blazorise

6.2.3 Funkčné celky

Štrukturálne sa celok aplikácie nezávislým pozorovateľom vnímaný ako *Frontend* v záujme členenia, optimalizácie a navigovateľnosti kódu skladá z viacerých elementárnych stavebných blokov – tieto celky sú medzi sebou prepájané a kombinované s cieľom zabezpečiť dostupnosť a vzhľad grafického užívateľského rozhrania a zároveň funkčnosť ovládacích prvkov najmä v zmysle implicitných operácií a zaručenia komunikácie so službami poskytovanými webovým serverom.

6.2.3.1 Stránky

Podobne ako u staršieho webového frameworku *ASP .NET Core*, navigačným a štrukturálnym základom webovej aplikácie postavenej na frameworku *Blazor* sú webové stránky. Stránky sú súbory s príponou *.razor*, ktoré reprezentujú ucelený funkcionálny celok a podobne ako takmer každý typ dynamickej webovej stránky združujú štýly, skripty a značkovací jazyk definujúci štruktúru a vzájomné interakcie už menovaných zložiek. Praktická časť práce za týmto účelom využívala už spomínanú knižnicu *Blazorise* – štruktúra kódu stránky vo fáze vývoja je teda mierne odlišná od zápisu vo forme čistého *HTML*.

```
@page "/machines"

@attribute [Authorize(Roles = UserRoleNames.AdminRoleName)]

@inject IJSRuntime _jsRuntime
@inject IUIService _uiService

<PageTitle>Správa výrobných zariadení</PageTitle>

<Container Fluid>
    <MachinesTitle />
</Container>

<Container>

    <CascadingValue Value="@CurrentMachines">
        <MachineAddForm />
        <MachinesList />
    </CascadingValue>

</Container>

@code {

    public IList<MachineDto> CurrentMachines = new List<MachineDto>();

    protected override async Task OnInitializedAsync()
    {
        _uiService.RefreshRequested += RefreshView;
        await base.OnInitializedAsync();
    }

    private void RefreshView()
    { StateHasChanged(); }
}
```

Zdrojový kód 6. Ukážka formátu dynamickej HTML stránky v rámci frameworku Blazor WebAssembly

Priložený kód demonštruje základné stavebné bloky stránky. Nachádza sa na ňom identifikátor stránky (*„/machines“*) využívaný za účelom navigácie, atribút obmedzujúci prístup k obsahu stránky osobám s nedostatočným oprávnením, injektovanie pomocných služieb prostredníctvom mechanizmu *Dependency Injection* a následne samotná kompozícia stránky. Sekcie štrukturálne zodpovedajúce usporiadaniu *HTML* tag-ov sú v prípade priloženého kódu zastúpené komponentami.

Poslednou – a pravdepodobne najdôležitejšou - sekciou *Blazor* stránky je blok *@code*, ktorý vo frameworku *Blazor* umožňuje definovať premenné a interakčnú logiku kľúčovú pre zabezpečenie dynamických funkcií stránky.

6.2.3.2 Komponenty

Komponenty v kontexte frameworku *Blazor* môžeme vnímať ako stavebné bloky stránok – v podstate ide o štruktúru takmer ekvivalentnú štruktúre stránky [viď. „*Zdrojový kód 6.*“]. Komponenty od stránok odlišuje absencia direktívy *@page* a nemožnosť využitia atribútu [*Authorize*] – k obmedzenia viditeľnosti komponenty sa využíva podobne fungujúci atribút [*AuthorizeView*].

```
@inject IJSRuntime _jsRuntime

<Row>

    <Column ColumnSize="ColumnSize.Is12"
            TextAlignment="TextAlignment.Center"
            Padding="Padding.Is0">
        <Jumbotron Class="jumbotron-machines jumbotron-centered"
                   Margin="Margin.Is0.FromBottom">
            <JumbotronTitle Size="JumbotronTitleSize.Is1"
                            Class="jumbotron-title-text">
                Správa - Stroje
            </JumbotronTitle>
        </Jumbotron>
    </Column>

<Divider Margin="Margin.Is5.FromBottom" Class="divider-main"
TextColor="TextColor.Primary" />

</Row>
```

Zdrojový kód 7. Ukážka formátu komponentu v rámci frameworku Blazor WebAssembly

6.2.3.3 API Services

Keďže rozsah súborov obsahujúcich stránky a komponenty by značne narástol v prípade, že by boli okrem obsluhy udalostí užívateľského rozhrania zodpovedné aj za pokročilé funkcionality na pozadí (ako napr. *API* volania na server), bola táto logika vyčlenená do špeciálnych tried – tzv. *API Services*. *API Services* v zmysle implementácie objednávkového systému je možné chápať ako bežné *C#* triedy s príponou *.cs* ukrývajúce metódy zabezpečujúce komunikáciu medzi klientom a serverom aplikácie prostredníctvom serializovaných dát (v drvivej väčšine bežných volaní na *API* serveru vo formáte *JSON*).

Okrem samotnej komunikačnej funkcie *API Services* zahŕňajú aj jednoduché využitie mapovacieho charakteru s cieľom spracovania odpovedí serveru, užitočné rozšírenia pre tvorbu reťazcov požiadaviek, či dokonca mechanizmy spracovania a vizualizácie chybových stavov v prípade neočakávanej odpovede od serveru.

```
1   public class DashboardApiService : ApiServiceBase<LogDto>, IDashboardApiService
2   {
3       public DashboardApiService(HttpClient client, IMapper mapper,
4           IErrorHandlingService errorHandlingService) :
5           base(client, mapper, errorHandlingService)
6       { }
7
8       #region Load
9
10      public async Task<LogPaginationDto?> LoadLogs(
11          LogPaginationSettingsDto paginationSettings)
12      {
13          var filter = GetQueryStringFromObject(paginationSettings);
14          var url = string.Format(
15              $"{DashboardControllerConstants.BaseUrl}/" +
16              $"{DashboardControllerConstants.Logs}?{filter}");
17          var response = await _client.GetAsync(url);
18
19          if (response.StatusCode != HttpStatusCode.OK)
20          {
21              await _errorHandlingService.HandleErrorResponse(response);
22              return null;
23          }
24          return await response.Content.ReadFromJsonAsync<LogPaginationDto>();
25      }
26
27      #endregion
```

Zdrojový kód 8. Kód API služby - Konštruktor a funkcia zodpovedná za načítanie stránkovaného zoznamu logov prostredníctvom volania metódou GET

6.2.3.4 Data Transfer Objects (DTOs)

Technicky sú z pohľadu praktickej časti práce objekty dátového prenosu umiestnené v projekte „*ProductionAdministration.Shared*“, t.j. nie je možné ich jednoznačne klasifikovať ani ako súčasť *Frontend*-u, ani ako súčasť *Backend*-u. Prakticky však drvivá väčšina manipulácie s *DTOs* prebieha práve na prostredí *Frontend*-u, kde sú priamo využívané ako podkladové objekty pre *UI* aplikácie. Previazanie na formuláre užívateľského rozhrania je uskutočnené prostredníctvom mechanizmu *Data Binding*, integrovaného vo väčšine *Blazor* komponent slúžiacich ako editačné polia.

```
<Validations @ref="@companyValidations" Mode="ValidationMode.Manual" Model="@CompanyInfo">

    <Fields>
        <Validation>
            <Field ColumnSize="ColumnSize.Is12.OnWidescreen">
                <FieldLabel For="name">@CompanyFields.Name:</FieldLabel>
                <TextEdit ElementId="name" Role="TextRole.Text"
                          Placeholder="Moja firma s.r.o. ..."
                          @bind-Text="@CompanyInfo.Name">
                    <Feedback>
                        <ValidationError Multiline="true" />
                    </Feedback>
                </TextEdit>
            </Field>
        </Validation>
    </Fields>
```

Zdrojový kód 9. Textové vstupné pole s validáciou - previazanie hodnoty zobrazovanej vo formulárovom poli s hodnotou vlastnosti „Name“ patriacu DTO objektu typu „CompanyInfo“ na pozadí

DTOs sú v rámci implementovanej aplikácie zároveň nositeľmi validačných atribútov, čo umožňuje realizovať atribútovú validáciu ako v prostredí klienta (zvyčajne vykonané pri pokuse o odoslanie dát formulára), tak v prostredí serveru (z dôvodu zabezpečenia nepriechodnosti nevhodných či podvrhnutých volaní) s konzistentnými výsledkami. Zároveň ide o prevenciu zanášania doménových modelov validačnou logikou, keďže v momente priechodu procesom mapovania už boli dáta objektu plne validované.

```
/// <summary>
/// Model reprezentujúci entitu adresy
/// </summary>
public class AddressDto
{
    #region Data

    /// <summary>
    /// Mesto
    /// </summary>
    [Required(ErrorMessage = $"Pole {AddressFields.City} je povinné.")]
    [StringLength(128, MinimumLength = 2,
          ErrorMessage = $"{AddressFields.City} musí obsahovať 2 - 128 znakov.")]
    public string City { get; set; }

    /// <summary>
    /// Krajina / Štát
    /// </summary>
    [Required(ErrorMessage = $"Pole {AddressFields.Country} je povinné.")]
    [EnumDataType(typeof(Country),
          ErrorMessage = $"{AddressFields.Country} nie je platným stavom.")]
    public Country Country { get; set; }
```

Zdrojový kód 10. Výstrižok kódu z triedy „AddressDto“, zobrazuje auto-implementované vlastnosti (Properties) dekorované validačnými atribútmi

6.3 Backend

Funkcionalita *Backend*-u je vzhľadom na povahu naprogramovaného systému prioritne zameraná na vykonávanie *CRUD* operácií nad jednotlivými elementárnymi entitami systému. Spracúva požiadavky na načítanie, uloženie, úpravu a mazanie objektov z perzistentného úložiska, prípadne na úpravu údajov entít, ktorých modifikácia v rámci aplikácie dáva zmysel. Systém na správu objednávok však disponuje aj iným typom funkcionality ako jednoduchými operáciami dátového manažmentu. Zodpovednosti *Backend*-u sú v tomto prípade oproti *Frontend*-u značne členitejšia. Nájdeme na ňom šablónovaciu a notifikačnú funkcionalitu, správu užívateľských účtov, logovacie možnosti, exporty objednávkových formulárov, prístup na vzdialené úložisko a e-mail prostredníctvom *Google API* a všemožné konfigurácie, vrátane nastavenia dokumentačného nástroja *Swagger*.

6.3.1 Konfigurácia

Serverový konfiguračný súbor *Program.cs* je v porovnaní s jeho *Frontend*-ovým ekvivalentom výrazne väčší a rozdelený do viacerých parciálnych funkcií podľa logických súvislostí konfigurovaných elementov. Najvýznamnejšie obsiahnuté celky sú:

- Nastavenie frameworku, smerovania a *MVC* mechanizmov
- Konfigurácia databázového pripojenia
- Konfigurácia autentifikačných a autorizačných mechanizmov
- Konfigurácia *Swagger* dokumentácie
- Registrácia služieb
- Registrácia repozitárov
- Registrácia úžitkových tried
- Registrácia služieb *Google*
- Registrácia šablónovacích nástrojov
- Evidencia mapovacích profilov a registrácia mapovacieho nástroja prostredníctvom *Dependency Container*-u

```
1   SetupDatabase();
2   SetupAuthMechanisms();
3   SetupMvc();
4   SetupSwaggerGen();
5   SetupServices();
6   SetupRepositories();
7   SetupUtils();
8   SetupMapper();
9   SetupGoogleFiles();
10  SetupTemplating();
11  SetupSettings();
12
13  var app = builder.Build();
14
15  if (app.Environment.IsDevelopment())
16  {
17      app.UseMigrationsEndPoint();
18      app.UseWebAssemblyDebugging();
19  }
20  else
21  {
22      app.UseExceptionHandler("/Error");
23      app.UseHsts();
24  }
25
26  ConfigureSwagger();
27  ConfigureFramework();
28  ConfigureAuthMechanisms();
29  ConfigureMvc();
30
31  app.Run();
```

Zdrojový kód 11. Hlavný obsah triedy „Program.cs“ v sekcii serveru aplikácie

6.3.2 Infraštruktúra

Záujmom členenia *Backend*-u bolo zachovať nosné myšlienky N-vrstvovej architektúry a profitovať z jej výhod – tento motív je zároveň dôvodom k tomu, že pri väčšine konvenčných operácií sú zapájané až tri kľúčové stupne *Backend*-ovej logiky – *Controllers*, *Services* a *Repositories*.

6.3.2.1 Controllers

Pomyselný priechod klientskej požiadavky *Backend*-om začína (po absolvovaní predspracovania autentifikačným a autentifikačným middleware) v triedach označovaných ako *Controller*. *Controller* je v kontexte vytvoreného systému zodpovedný za validáciu prichádzajúcej požiadavky, delegovanie činnosti príslušnej službe a formuláciu *HTTP* odpovede popisujúcej výsledok vykonanej operácie (či už úspešnej, alebo nie).

```
/// <summary>
/// Endpoint slúžiaci k načítaniu skrátených hlavičiek prehľadu objednávok
/// </summary>
/// <param name="paginationSettings">Objekt zaobaľujúci nastavenia stránkovania pre načítanie prehľadu objednávok</param>
/// <returns>Zoznam nesúci modely údajov objednávok</returns>
[HttpGet($"{OrderControllerConstants.BaseUrl}/{OrderControllerConstants.Info}")]
[ProducesResponseType(typeof(OrderPaginationDto), (int)HttpStatusCode.OK)]
[ProducesResponseType(typeof(BadRequestResultDto), (int)HttpStatusCode.BadRequest)]
public async Task<IActionResult> LoadOrderInfos([FromQuery, Required] OrderPaginationSettingsDto paginationSettings)
{
    IActionResult result;
    try
    {
        ValidateModel();
        var ordersPage = _orderService.LoadOrderInfos(paginationSettings);
        result = GetHttpResult(HttpStatusCode.OK, ordersPage);
    }
    catch (ApiException ex)
    {
        var response = ProcessApiException(ex);
        result = GetHttpResult(response.StatusCode, response);
    }
    return result;
}
```

Zdrojový kód 12. Koncový bod v Controller-i objednávok reagujúci na HTTP metódu GET. Zodpovedá za načítanie stránkovaného zoznamu skrátených informácií.

Metódy *Controller*-ov reagujúce na externé volania určitej *URL* adresy v prostredí webových aplikácií nazývame koncovými bodmi (*endpoints*). Samotná implementácia týchto metód v systéme obsahuje viacero atribútov (na vyššie priloženej ukážke kódu menovite „*HttpGet*“ a „*ProducesResponseType*“) – tieto dekorátory jednak určujú *URL* adresu daného koncového bodu, jednak typ *HTTP* volania, na ktoré má metóda na príslušnej adrese reagovať. Atribút „*ProducesResponseType*“ následne umožňuje priradiť odpovedi odosielanej klientovi korektný návratový kód na základe formátu dátového objektu.

Kód uvedený na poslednom obrázku povoľuje nasledovné možnosti návratovej správy:

- **OK – 200:**
  - Serializovaný *JSON* objekt v tvare „*OrderPaginationDto*“
- **Bad Request – 400:**
  - Serializovaný *JSON* objekt v tvare „*BadRequestResultDto*“.

Základová trieda pre inštancie objektov slúžiacich v rámci aplikácie ako *Controller* - „*ApiControllerBase*“ - okrem ďalších úžitkových funkcií obsahuje mechanizmy na spracovanie ošetrených výnimiek vyvolaných aplikáciou a následnú formuláciu odpovede vo formáte, ktorá pre potreby konzumenta *API* reflektuje pravdepodobnú príčinu problému.

6.3.2.2 Services

Služby (*Services*) sú ťažiskom *Backend*-ovej logiky. Volané buď z *Controller*-u alebo z iných služieb, štandardne zodpovedajú za premapovanie prijatého *DTO* objektu do doménového formátu, vykonanie určitej dátovej operácie a za delegovanie ďalšej činnosti zodpovednému repozitáru – samozrejme za predpokladu, že došlo k zmenám, ktoré je žiadúce premietnuť do perzistentných dát. Operácie vyžadujúce odpoveď vo formáte vhodného *DTO* v prípade úspešného spracovania taktiež využívajú metód služieb za účelom formulácie týchto odpovedí.

```
1   #region Insert
2
3   public EmployeeDto AddEmployee(EmployeeDto addEmployeeModel, Guid userId)
4   {
5       var employee = _mapper.Map<Employee>(addEmployeeModel);
6       employee.User = null;
7       _employeeRepository.Insert(employee);
8
9       if (employee is not null && !string.IsNullOrEmpty(addEmployeeModel.Email) &&
10          addEmployeeModel.Role != UserRole.NONE)
11      {
12          _userService.CreateAccount(userId, employee, addEmployeeModel.Email,
13              addEmployeeModel.Role);
14      }
15
16      LogEmployeeAction(userId, OperationType.INSERT, employee);
17      return ConvertEmployeeToDto(employee);
18  }
19
20  #endregion
```

Zdrojový kód 13. Metóda triedy „EmployeeService.cs“ zabezpečujúca pridanie nového užívateľa a v prípade potreby vytvorenie jeho užívateľského účtu

6.3.2.3 Repositories

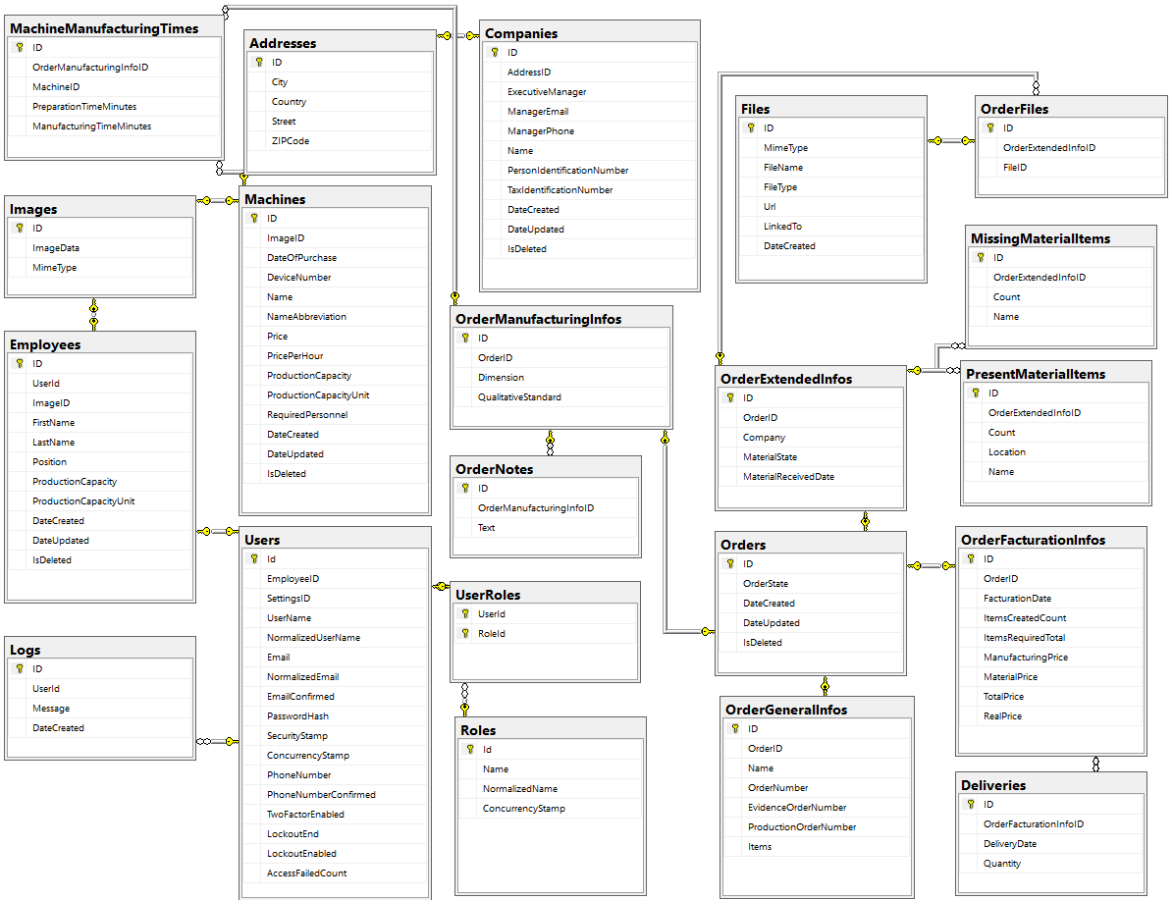
Repozitáre sú triedy zodpovedné za vykonávanie operácií nad dátami perzistentného úložiska, resp. ich abstrakciou získanou prostredníctvom aplikáciou nasadeného *ORM* mechanizmu (*Entity Framework*). Povaha operácií môže byť rôznorodá – môže sa jednať o načítanie, odmazávanie či ukladanie zmien, v niektorých prípadoch dokonca filtráciu záznamov na najnižšej úrovni – spracúvanie náročných dátových operácií na úrovni kolekcií zodpovedajúcich relačným tabuľkám priamo z repozitára šetrí systémové zdroje a urýchľuje spracovanie požiadavky tým, že zabraňuje vytváraniu a transformácii zbytočných premenných. Repozitáre vytvorené v rámci praktickej časti práce štedro využívajú technológiu *LINQ* pre elegantný a čistý zápis vykonávaných operácií.

```
1   public class OrderRepository : IOrderRepository
2   {
3       private readonly ApplicationDbContext _dataContext;
4
5       public OrderRepository(ApplicationDbContext dataContext)
6       {
7           _dataContext = dataContext;
8       }
9
10      #region Load
11
12      public Order? GetById(int id)
13      {
14          return _dataContext.Orders
15              .Include(x => x.GeneralInfo)
16
17              .Include(x => x.ExtendedInfo)
18              .ThenInclude(x => x.MissingItems)
19              .Include(x => x.ExtendedInfo)
20              .ThenInclude(x => x.PresentItems)
21              .Include(x => x.ExtendedInfo)
22              .ThenInclude(x => x.Files)
23              .ThenInclude(x => x.File)
24
25              .Include(x => x.FacturationInfo)
26              .ThenInclude(x => x.Deliveries)
27
28              .Include(x => x.ManufacturingInfo)
29              .ThenInclude(x => x.MachineTimes)
30              .Include(x => x.ManufacturingInfo)
31              .ThenInclude(x => x.Notes)
32
33              .SingleOrDefault(x => x.ID == id);
34      }
35
36      #endregion
```

Zdrojový kód 14. Metóda triedy objednávkového repozitára načítajúca jediný kompletný záznam na základe poskytnutého ID

6.3.3 Databáza

Prítomnosť perzistentného úložiska v systéme správy objednávok je prakticky nutná – túto funkciu zastrešuje databáza *MS SQL Server 2019*. Jedná sa o databázové riešenie relačného charakteru pomerne jednoducho integrovateľné do väčšiny aplikácií založených na technológiách spoločnosti *Microsoft*. Vývoj systému prebiehal nad lokálnou databázou umiestnenou priamo v zariadení autora práce – pri ostrom nasadení aplikácie by samozrejme bolo nutné toto nastavenie nahradiť buď analogickým umiestnením databázy priamo na cieľovom zariadení, alebo vhodnou metódou webhostingu, za zabezpečenia korektnej modifikácie pripájacích reťazcov (*Connection Strings*) [203] uložených v konfiguračnej sekcii aplikácie.



Obrázok 39. Schéma funkcionálne významných databázových tabuliek aplikácie

Programový prístup k databáze z prostredia aplikácie však prebieha prostredníctvom technológie *Entity Framework*. *Entity Framework* je z hľadiska projektu výhodnou voľbou v tom zmysle, že vzhľadom na povahu aplikácie je drobná rýchlostná penalizácia v porovnaní s natívnym *SQL* prístupom k dátam sprevádzajúca jeho použitie znesiteľnou cenou za znateľné zvýšenie efektivity vývoja vplyvom možnosti pristupovať k dátam relačného úložiska rovnakým spôsobom, ako ku klasickým objektom v prostredí *.NET* [203].

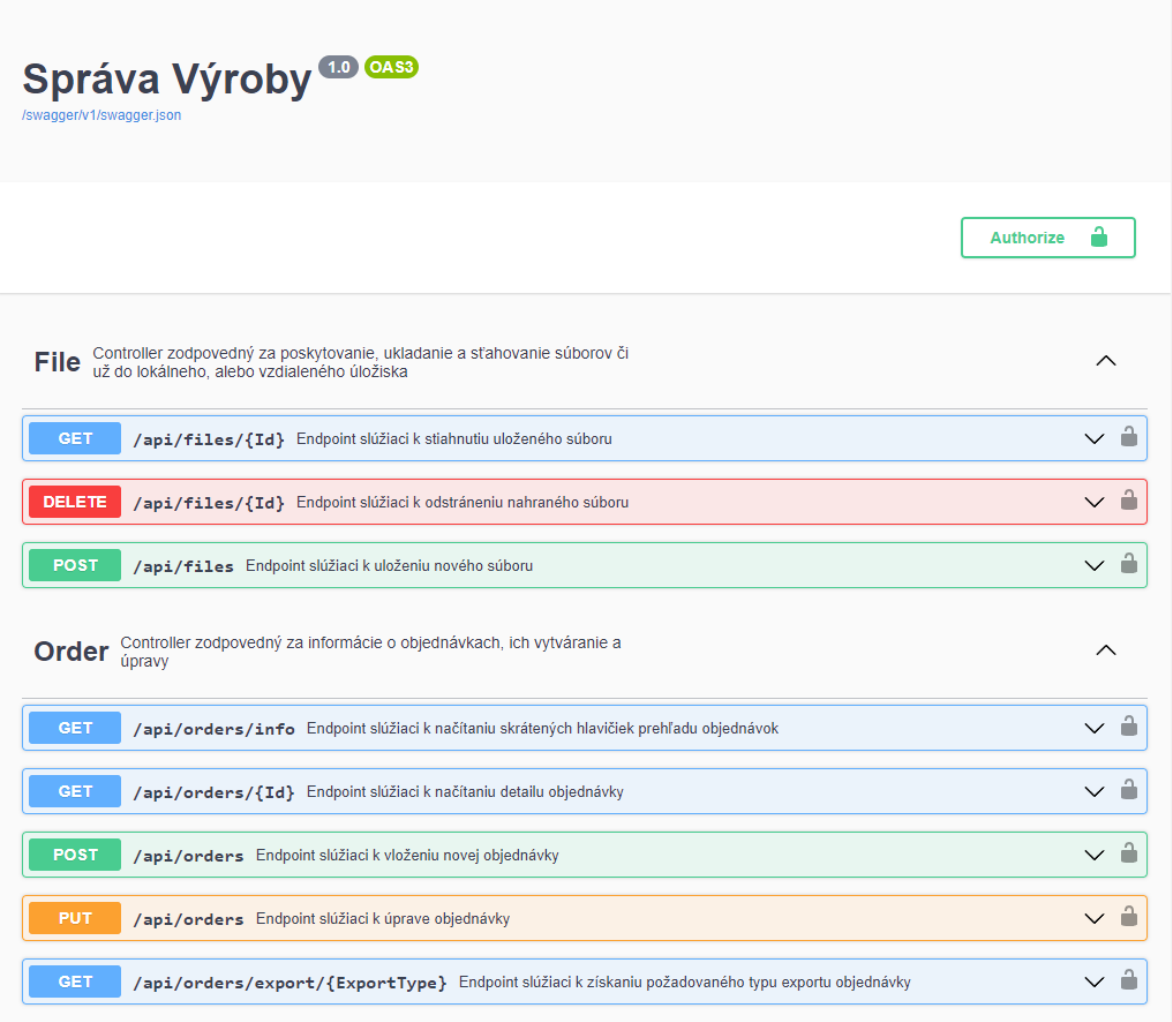
Úložisko je z pohľadu aplikácie zastúpené triedou „*ApplicationDbContext.cs*“. Táto trieda v sebe nesie sadu kolekcií typu *DbSet<T>* zodpovedajúcu jednotlivým databázovým tabuľkám. Taktiež umožňuje konfiguráciu množstva vlastností a špecifík týchto záznamov – ako napr. integritných obmedzení, názvov a veľkostí stĺpcov či cudzích kľúčov prostredníctvom registrácie programovateľných profilov [203].

```
1  public class ApplicationDbContext : ApiAuthorizationDbContext<ApplicationUser>
2  {
3      public ApplicationDbContext(DbContextOptions options, IOptions<OperationalStoreOptions>
4      operationalStoreOptions) : base(options, operationalStoreOptions)
5      { }
6
7      #region Entities
8
9      public DbSet<Machine> Machines { get; set; }
10     public DbSet<Employee> Employees { get; set; }
11     public DbSet<Company> Companies { get; set; }
12     public DbSet<Address> Addresses { get; set; }
13
14     public DbSet<Order> Orders { get; set; }
15     public DbSet<OrderGeneralInfo> OrderGeneralInfos { get; set; }
16     public DbSet<OrderExtendedInfo> OrderExtendedInfos { get; set; }
17     public DbSet<OrderFacturationInfo> OrderFacturationInfos { get; set; }
18     public DbSet<OrderManufacturingInfo> OrderManufacturingInfos { get; set; }
19     public DbSet<MissingMaterialItem> MissingMaterialItems { get; set; }
20     public DbSet<PresentMaterialItem> PresentMaterialItems { get; set; }
21     public DbSet<OrderFile> OrderFiles { get; set; }
22     public DbSet<Delivery> Deliveries { get; set; }
23     public DbSet<OrderNote> OrderNotes { get; set; }
24     public DbSet<MachineManufacturingTime> MachineManufacturingTimes {get;set;}
25
26     #endregion
27 }
```

Zdrojový kód 15. Trieda „ApplicationDbContext.cs“ udržiavajúca úložiskový kontext pre kolekcie reprezentujúce jednotlivé tabuľky

6.3.4 Dokumentácia

Dokumentácia *API* vyvíjaného systému je zabezpečovaná nástrojom *Swagger*. *Swagger* je nástroj vedený pod *Open-Source* licenciou vyvíjaný firmou *SmartBear*, ktorý (okrem mnohých ďalších schopností) umožňuje automatickú tvorbu interaktívnej programovej dokumentácie prostredníctvom komentárov vo formáte *XML* [204].



Obrázok 40. Automaticky generovaná Swagger dokumentácia, zachytáva vrchnú úroveň aplikačného programového rozhrania tried „FileController.cs“ a „OrderController.cs“

Vyvinutý systém vďaka technológii *Swagger* dynamicky generuje dokumentáciu k projektom „*ProductionAdministration.Shared*“ a „*ProductionAdministration.Server*“. Príčina obmedzenia rozsahu na tieto dva projekty je intuitívna – pre potreby konzumenta *API* je dôležité vidieť najmä formát, polia a validačné obmedzenia použitých *DTOs* (umiestnených v projekte „*ProductionAdministration.Shared*“) a taktiež členenie, tvar, *URL* adresy a formáty odpovedí jednotlivých koncových bodov (obsiahnutých v projekte „*ProductionAdministration.Server*“).

Generovanie dokumentácie z internej logiky aplikácie nie je nutné – a v prípade poskytovania dokumentácia rozhrania tretím stranám často prakticky nežiadúce.

6.4 Zabezpečenie

Problematika zabezpečenia webových aplikácií je mimoriadne komplexná – na rozdiel od programov bežiacich v uzatvorenom prostredí bez možnosti externého prístupu, systémy typu Klient – Server bývajú prepojené prostredníctvom počítačovej siete, či už drôtovej alebo bezdrôtovej povahy. Vzdialená komunikácia v kombinácii s mnohými ďalšími prvkami prispieva k mimoriadnej zraniteľností webových aplikácií v prípade zanedbania bezpečnostného faktoru. Prakticky čokoľvek môže pripraviť pôdu pre určitú formu útoku – slabé zabezpečenie stroja, na ktorom aplikácia beží, neošetrené vstupy či absencia validácií, zastaralé technológie, odpočúvanie nešifrovanej sieťovej komunikácie treťou stranou, možnosť neoprávneného prieniku do systému alebo neautorizovaný prístup k dôverným sekciám či údajom systému [205]. Aplikácie využívajúce užívateľské vstupy a perzistentné úložisko relačného charakteru sú taktiež notoricky známe svojou náchylnosťou na útok typu *SQL Injection*.

Zabezpečenie skonštruovaného systému v žiadnom prípade nie je perfektné – skúsená a cieľavedomá osoba by nepochybne bola schopná odhaliť prítomnosť úskalí využiteľných pre prípadné záškodné aktivity. Počas vývoja však bola vyvíjaná aktívna snaha najčastejšie a najvýznamnejšie hrozby buď eliminovať, alebo aspoň do maximálnej možnej miery obmedziť. Prijaté opatrenia sa týkali ako špecifických naprogramovaných funkcionalít aplikácie, tak voľbou vhodných nastavení frameworku spoločne synergizujúcich do stavu, kedy je pre potenciálneho útočníka napadnutie procesov aplikácie síce nie nemožné, ale výrazne ťažšie ako v pôvodnom, nemodifikovanom stave.

Uvedenie aplikácie do produkčného prostredia (samotné nasadenie nie je súčasťou tejto práce) by naviac vyžadovalo dodatočné ošetrenie niekoľkých špecifických a potenciálne problematických oblastí a hrozieb - za zmienku stojí napr. útok pomocou tzv. *Cross Site Scripting* – podrobný návod je poskytnutý v oficiálnej dokumentácii firmy *Microsoft* [206].

6.4.1 Protokol

Hlavným webovým prenosovým protokolom využívaným aplikáciou je protokol *HTTPS*. Konkrétne rozdiely medzi protokolom *HTTP* a *HTTPS* sú spracované v príslušnej kapitole teoretickej časti práce [viď. kapitola *1.4.2 – "HTTPS"*] – pre potreby nášho systému toto špecifikum znamená, že požiadavky / odpovede (a obecne komunikácia medzi klientom a serverom) je šifrovaná, čo ju činí výrazne menej náchylnou na únik dát vplyvom načúvania tretej strany na sieťovom spojení, cez ktoré informácie prúdia.

Nutným dodatkom k využitiu *HTTPS* protokolu v implementovanom projekte je skutočnosť, že aplikácia v súčasnosti využíva vývojársky certifikát poskytovaný vývojovým prostredím *Visual Studio 2022*. Ostrá prevádzka systému by vyžadovala nasadenie a konfiguráciu certifikátu vydaného oprávnenou certifikačnou autoritou.

6.4.2 Autentifikácia

Autentifikácia veľkou mierou spolieha na pridruženú funkcionalitu frameworku *ASP .NET Core Identity*. Nastavenie prepojenia dátovej zložky správy užívateľských účtov je realizované prostredníctvom k tomuto účelu vyhradených sekcií súboru „*Program.cs*" na strane serveru. Implementovaný systém sa však nespolieha výhradne na nepozmenenú formu autentifikačných mechanizmov – uvádza vlastné úpravy, ktoré umožňujú lepšie zapuzdrenie systému ako uzatvoreného celku reagujúceho iba na úzko ohraničenú množinu operácií z okolitého prostredia.

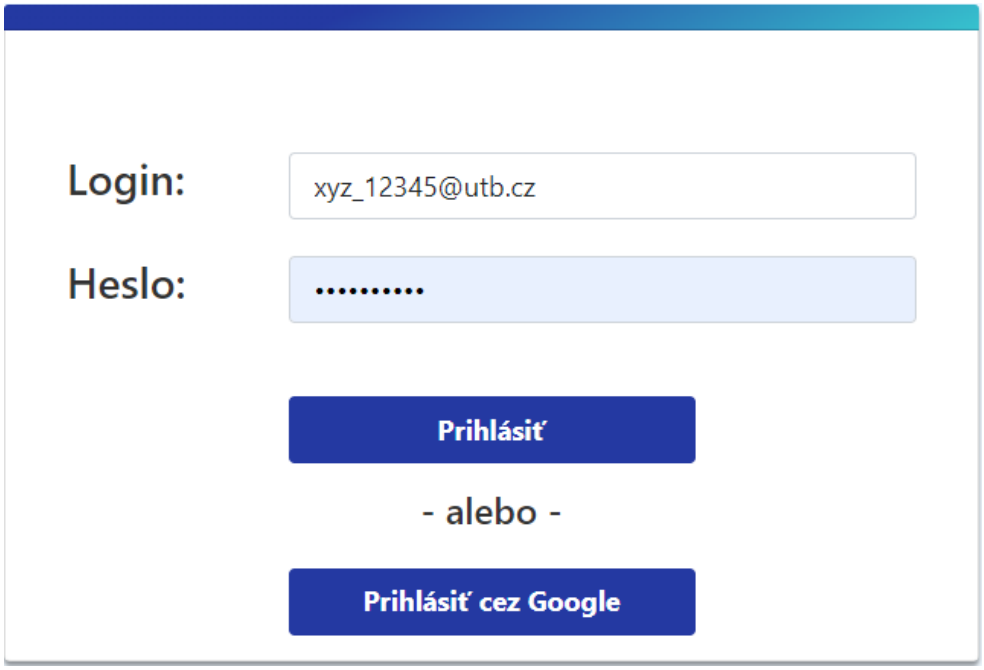
6.4.2.1 Registrácia

Vzhľadom na povahu systému a jeho určenie pre interné využitie v rámci výrobnej firmy – a v neposlednom rade atraktívnu možnosť prevádzky na internej / lokálnej sieti, bez nutnosti viditeľnosti zo siete *Internet* – neobsahuje systém možnosti registrácie. Príslušné časti kódu, formuláre a triedy predpripravené v rámci *ASP .NET Identity* balíčka boli z projektu odstránené – rovnako boli odstránené aj ľubovoľné odkazy umožňujúce navigáciu na niektorú z registračných akcií. Systém pri iniciálnej dátovej migrácii automaticky vytvorí jeden užívateľský a jeden administrátorský účet s pevne stanovenými prihlasovacími údajmi. Odporúčanie pre ostré nasadenie systému je pochopiteľne tieto účty použiť iba na prvotné vytvorenie administrátorského účtu s personalizovanými údajmi a následne zabezpečiť ich okamžité zmazanie.

Registrácia nových užívateľov prebieha priamo z prostredia aplikácie – nový užívateľ je vytvorený vždy, keď je pridaný profil zamestnanca disponujúci oprávneniami manipulácie so systémom. Heslo nového užívateľa je generované ako náhodný reťazec formátu *GUID* a odosielané elektronicky, formou E-mailovej notifikácie *SMTP* kanálom zabezpečeným prostredníctvom protokolu *SSL*.

6.4.2.2 Prihlásenie

Prihlásenie užívateľa prebieha skrz grafické rozhranie dedikovaného prihlasovacieho dialógu – základ dialógu je opäť podložený predpripravenými šablónami a funkciami poskytovanými balíčkom *ASP .NET Identity*, bol však upravený tak, aby korešpondoval s celkovým vizuálnym štýlom aplikácie.



Obrázok 41. Prihlasovací formulár

Prihlasovací dialóg okrem konvenčného prihlásenia podporuje aj prihlásenie skrz externú službu *Google* – po zadaní svojich prihlasovacích údajov do *Google* účtu v sprostredkovanom dialógovom okne je užívateľ v tomto prípade automaticky vpustený do aplikácie – samozrejme za predpokladu, že bol v databáze nájdený užívateľský účet so zodpovedajúcou E-mailovou adresou.

6.4.3 Autorizácia

Autorizačné mechanizmy sú prítomné naprieč celou aplikáciou v značne rôznorodých formátoch. Tieto kontroly umožňujú limitovať obsah, ktorý je prihlásenému užívateľovi prístupný. Nadväznosť autorizácií na autentifikačný mechanizmus je taká, že autentifikačný proces v prípade *ASP .NET Core Identity* slúži zároveň k získaniu reťazca nazývaného „*Bearer Token*“. Pokiaľ bolo prihlásenie úspešné, je token po krátky, časovo obmedzený úsek uložený formou konfigurovateľnou z prostredia súboru „*Program.cs*“ – v prípade systému skonštruovaného v rámci tejto práce ide o lokálne uloženie tokenu do lokálnej pamäte webového prehliadača [207]. Metóda autentifikácie užívateľa serveru prostredníctvom tokenu je založená na predpoklade, že neprihlásený užívateľ sa k tokenu nedostane a tým pádom nie je oprávnený komunikovať so sekciami aplikácie vyžadujúcimi jeho prihlásenie.

```
options.AddSecurityDefinition("JWT Token", new OpenApiSecurityScheme
{
    Type = SecuritySchemeType.ApiKey,
    Name = "Authorization",
    Description = "Copy 'Bearer' + valid JWT Token into field",
    In = ParameterLocation.Header
});
options.AddSecurityRequirement(new OpenApiSecurityRequirement()
{
    {
        new OpenApiSecurityScheme()
        {
            Reference = new OpenApiReference
            {Type = ReferenceType.SecurityScheme, Id = "JWT Token"}
        },
        Array.Empty<string>()
    }
});
```

Zdrojový kód 16. Nastavenie JWT Token-u ako metódy autorizácie klienta

Bearer Token je po obdržaní následne odosielaný v rámci každej požiadavky na server ako pole v hlavičke požiadavky – pokiaľ absentuje, server požiadavku považuje za neautorizovanú a vracia *HTTP* odpoveď *401 – Unauthorized*.

6.4.3.1 Frontend

Frontend využíva autorizáciu najmä v zmysle obmedzenia stránok a kontrolných prvkov, ktoré užívateľovi s určitou úrovňou oprávnení smú / nesmú byť zobrazené.

Úlohu autorizačnej bariéry plnia nasledovné programové konštrukty:

- **Atribút [*Authorize*]:**

  Obmedzuje viditeľnosť, resp. prístupnosť celej stránky ako takej

```
@page "/machines"

@attribute [Authorize(Roles = UserRoleNames.AdminRoleName)]

@inject IJSRuntime _jsRuntime
@inject IUIService _uiService

<PageTitle>Správa výrobných zariadení</PageTitle>
```

Zdrojový kód 17. Aplikácia obmedzovacieho atribútu „Authorize“ na webovú stránku ako celok. Zobrazené nastavenie umožňuje prístup iba užívateľom v roli administrátora

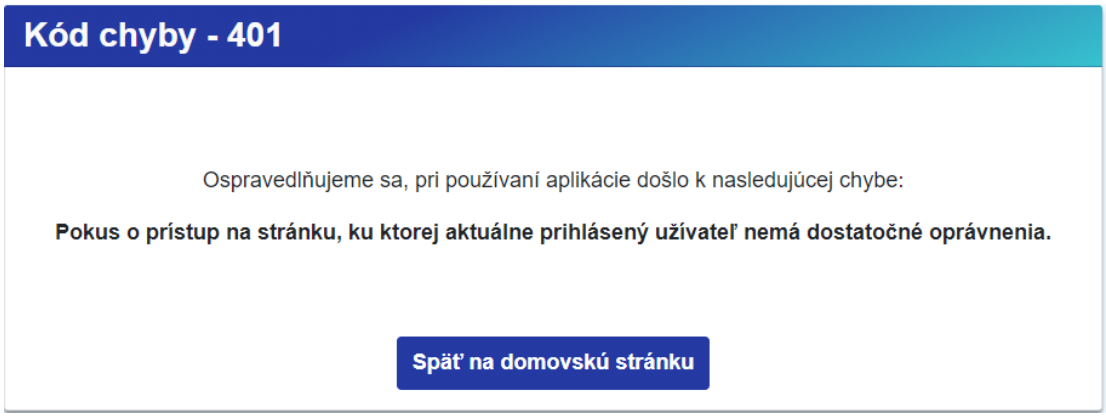
- **Komponent [*AuthorizeView*]:**

  Slúži na limitáciu viditeľnosti obaleného obsahu / komponent renderovaných stránok

```
<AuthorizeView Roles="@UserRoleNames.AdminRoleName">
    <Row>
        <Column ColumnSize="ColumnSize.Is12">
            <LogsList />
        </Column>
    </Row>
</AuthorizeView>
```

Zdrojový kód 18. Aplikácia obmedzovacieho atribútu „AuthorizeView“ vo forme obalového komponentu na časť webovej stránky. Zobrazené nastavenie umožňuje prístup iba užívateľom v roli administrátora

Atribúty poskytujú jednoduchý a funkčný spôsob ako využiť už existujúce mechanizmy zabezpečenia prítomné naprieč aplikáciou k riadeniu prístupnosti obsahu. Program zároveň obsahuje vizualizačné prostriedky napovedajúce užívateľovi, že jeho požiadavka bola vyhodnotená ako neoprávnená.



Obrázok 42. Chybová hláška pri snahe o presmerovanie na stránku neprístupnú užívateľom s nedostatočnou úrovňou oprávnení

6.4.3.2 Backend

Backend integruje autorizačné obmedzenia prioritne v prostredí tried typu *Controller*. Atribútmi [*Authorize*] sú označované buď jednotlivé metódy reprezentujúce koncové body, alebo na celé *Controller*-y (v prípade, že je žiadúce aplikovať uniformnú prístupovú politiku na všetky jeho koncové body bez rozdielu).

Dekorovanie metód a deklarácií tried *Controller*-ov autorizačnými atribútmi sa zabraňuje neautorizovanému prístupu k dátam v prípade, že by sa potenciálnemu útočníkovi nejakým spôsobom podarilo prekonať kontroly implantované do *GUI* aplikácie, prípadne by k internej funkcionalite systému pristupoval formou konzumácie *API* bez prístupu cez grafické rozhranie.

```
/// <summary>
/// Controller zodpovedný za informácie o objednávkach, ich vytváranie a úpravy
/// </summary>
[Authorize]
public class OrderController : ApiControllerBase
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService,
        IHttpContextAccessor httpContextAccessor,
        IHttpStatusCodeToResultMessageDictionary exceptionMessageDictionary) :
        base(httpContextAccessor, exceptionMessageDictionary)
    {
        _orderService = orderService;
    }
}
```

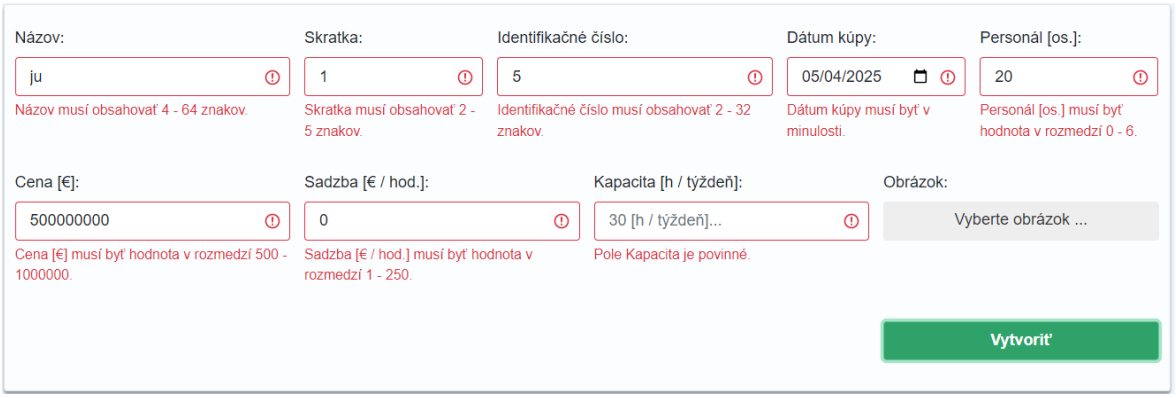
Zdrojový kód 19. Aplikácia obmedzovacieho atribútu „Authorize“ na celú triedu typu Controller. Prístup k jeho metódam je umožnený všetkým prihláseným užívateľom

6.4.4 Validácia

Aplikácie disponujúce rozhraním manipulovateľným či už prostredníctvom užívateľského vstupu alebo formou programového prístupu k ich *API* sú neustále ohrozované možnosťou prieniku dát, ktoré nepodliehajú požadovaným integritným obmedzeniam. Nielen, že je prienik neošetrených a nekorigovaných dát potenciálne významným narušením celkového chodu systému, môže predstavovať aj bezpečnostné riziko poskytnutím dobrej platformy napr. pre útoky typu *SQL Injection*. Systém správy objednávok obsahuje validácie ako *Frontend*-ové, tak validácie *Backend*-ové – obidve formy sú zabezpečované nasadením tzv. *Data Annotation* atribútov na polia *DTOs* v kombinácii s vhodným podporným mechanizmom pre extrakciu chybového hlásenia a jeho spätnú propagáciu autorovi požiadavky.

6.4.4.1 Frontend

Frontend-ové validácie systému sú viazané na jednotlivé editačné formuláre rozmiestnené naprieč celými webovými stránkami aplikácie. Zabudované validačné mechanizmy formulárov majú v kombinácii s obslužným kódom za úlohu neprepustiť nekorektné požiadavky a zároveň poskytovať ľudsky čitateľnú spätnú väzbu užívateľovi, ktorý s nimi prostredníctvom *GUI* manipuluje.



Obrázok 43. Validácie v klientskej časti aplikácie – formulár pre vytvorenie výrobného zariadenia

Validáciami v klientskej časti aplikácie sú opatrené všetky formuláre, pričom pri snahe o vyvolanie akcie nedôjde k žiadnej činnosti, pokiaľ došlo k zlyhaniu čo len jednej z validačných kontrol príslušného formulára.

6.4.4.2 Backend

Alternatívnym prípadom prevolávania internej logiky aplikácie je napr. prístup bez využitia *Frontend*-u a s ním spojeného grafického užívateľského rozhrania, napr. priamym volaním koncových bodov serverového *API*. Ošetrenie tejto formy prístupu je samozrejme tiež nutné – pokiaľ by ošetrenie absentovalo, bolo by mimoriadne jednoduché serveru podsunúť nekorektné dáta. Výhodou v tomto prípade je špecifikum frameworku *Blazor*, ktorý umožňuje zdieľanie kódu medzi klientom a serverom – *DTOs* podliehajúce formulárovým validáciám sú tie isté *DTOs*, ako objekty ktoré očakáva a validuje *Controller*. Zdieľanie validačného mechanizmu zabezpečuje konzistenciu validácie zobrazovanej v klientskej sekcii aplikácie s validáciou poskytovanou konzumentovi pri priamom prevolaní *API* serveru. Validačná funkcia serveru je po definovaní obmedzení kladených na jednotlivé polia v príslušných *DTOs* už relatívne triviálna [viď. „*Zdrojový kód 20.*“].

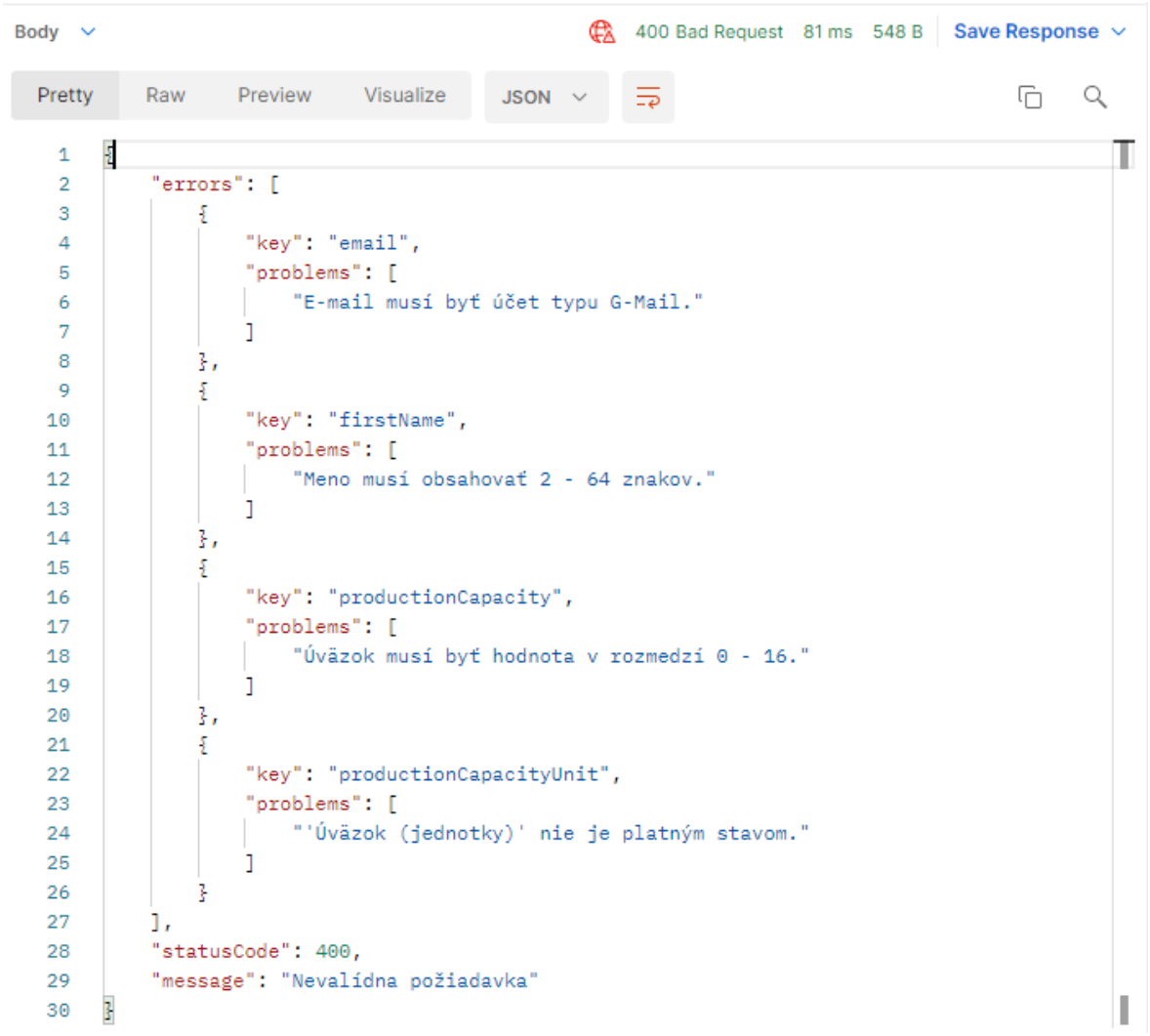
```
#region Validation

/// <summary>
/// Metóda určená k exekúcii validácie pred vstupom modelu do internej logiky
/// </summary>
/// <returns></returns>
protected void ValidateModel()
{
    if (!ModelState.IsValid)
    {
        var problem = ValidationProblem();
        if (problem is not null)
        {
            var ex = new BadRequestException()
            {
                InternalExceptionCode = ApiInternalExceptionCode.BAD_REQUEST,
                Errors = ValidationUtils
                .FormatValidationResponse(problem, ValidationFormat.JSON),
            };
            throw ex;
        }
    }
}

#endregion
```

Zdrojový kód 20. Proces validácie na strane serveru, vo väčšine prípadov vykonávaný okamžite po príchode požiadavky do Controller-u

Validácia objektov je vykonávaná zásadne pred ich postúpením na interné spracovanie príslušnej službe. Vizualizačné mechanizmy klienta sú v rámci kontroly chybových odpovedí indukovaných chybou validácie na strane *Backend*-u taktiež schopné problém detegovať a užívateľsky viditeľným spôsobom komunikovať (prostredníctvom *Toast* notifikácií). Dôkazom o funkčnosti validácie na strane serveru môže byť skúšobné volanie jednotlivých *endpoints* vhodne zvoleným nástrojom – napr. prostredníctvom softwaru *Postman* [208].



Obrázok 44. – Výsledok serverovej validácie pri odoslaní požiadavky s nekorektnými dátami, výstup získaný prostredníctvom nástroja Postman

6.4.5 Databáza

Mimoriadne často spomínaným ohrozením aplikácií manipulujúcich s perzistentným úložiskom sú útoky typu *SQL Injection*. Voči *SQL Injection* sa aplikácia bráni primárne databázovým prístupom s využitím *ORM* riešenia *Entity Framework*. Samotný *Entity Framework* však nebráni exekúcii útoku typu *SQL Injection* – ochranu poskytuje až kombinácia jeho nasadenia s ďalšími faktormi a ošetreniami implantovanými do štruktúry aplikácie a metodiky, akou je s dátami manipulované.

[209, 210] Ochranné opatrenia sa dajú zhrnúť do dvoch hlavných položiek:

- **Parametrizácia databázových otázok:**

  Ľubovoľné databázové operácie manipulujúce s externými premennými (bez ohľadu to, či sú premenné poskytované užívateľom alebo nie) využívajú parametrický, procedurálny prístup – to znamená, že nikdy nedochádza k priamej kombinácii užívateľského vstupu s telom otázky.

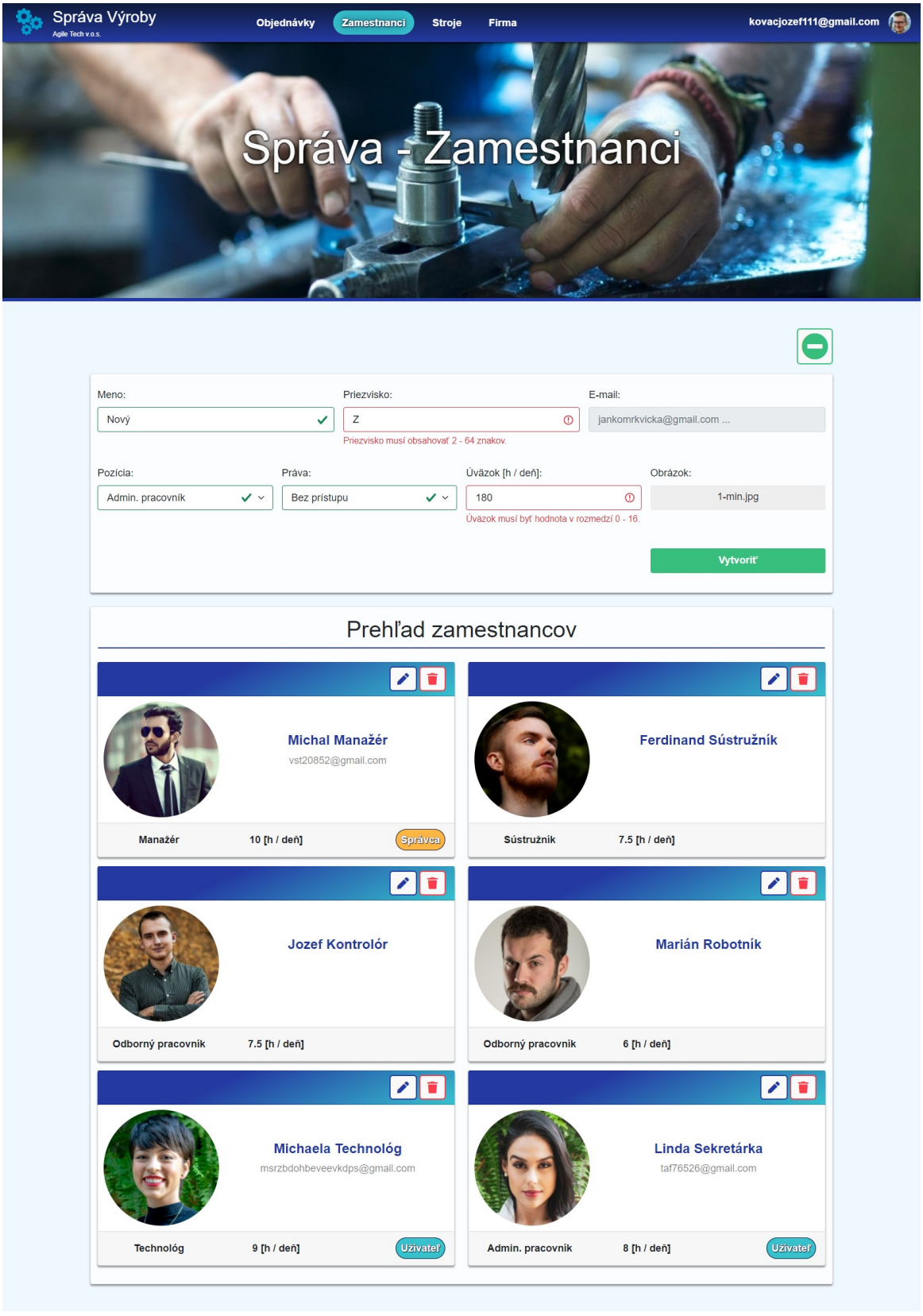
- **Využitie LINQ:**

  Otázky na dáta vykonávané prostredníctvom mechanizmu *LINQ* sú dobrým spôsobom bezproblémovej manipulácie s databázovými kolekciami – okrem jednoduchosti však *LINQ* poskytuje aj bezpečnostný benefit. Kompozícia otázky s využitím funkcionality *LINQ* je spracúvaná prostredníctvom tzv. „*Object Level API*“, bez reťazenia textových častí otázky v surovej forme, vrátane implicitného ošetrenia potenciálne nebezpečných špeciálnych znakov – to činí *LINQ* databázové otázky odolnými voči klasickým *SQL Injection* útokom.

7 VÝSLEDKY

Vytvorená aplikácia pre správu objednávok výrobnej firmy si kládla za cieľ dosiahnuť stav, kedy by ju bolo možné označiť ako *MVP* (*Minimal Viable Product*). Integruje základné myšlienky stanovené v analytickej fáze praktickej časti práce a obsahuje funkcionalitu nevyhnutnú k naplneniu ako funkčných, tak nefunkčných požiadaviek a pokrytiu prípadov použitia adresujúcich dané požiadavky. Užívateľský pohľad na aplikáciu pozostáva z „modulov“ rozčlenených na základe navigácie v rámci webových stránok systému. Nasledujúce kapitoly sa venujú predstaveniu základných častí systému z užívateľského pohľadu a v jednoduchosti popisujú užívateľské možnosti manipulácie s aplikáciou prostredníctvom poskytnutého *GUI*.

7.1 Správa zamestnancov



Obrázok 45. Webstránka správy zamestnancov [Obrázky prevzaté z: 211, 212, 213, 214, 215, 216, 217]

Správa zamestnancov je časťou systému zodpovednou za možnosti evidencie zamestnancov v prostredí firmy – sprostredkúva primárne *CRUD* funkcionalitu nad entitami zamestnancov. Skladá sa z viacerých funkčných podcelkov, každý je v tomto prípade zameraný za jemu vyhradenú časť *CRUD* funkcionality.

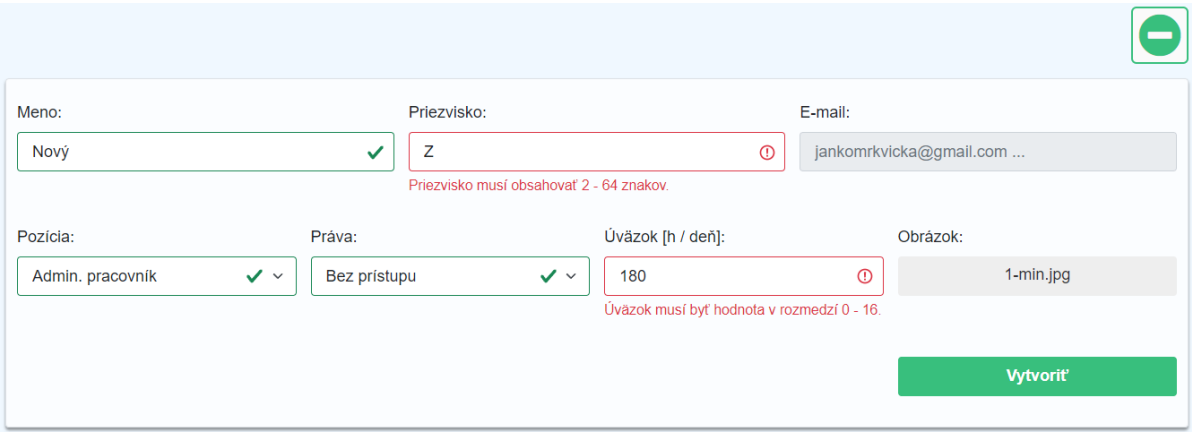
7.1.1 Prehľad zamestnancov

Obrázok 46. Kartička s informáciami zamestnanca v prehľade [Obrázok prevzatý z: 216]

Prehľad zamestnancov je dominantnou črtou stránky. Jedná sa vlastne o priestor vyhradený pre „kartičky“ reprezentujúce jednotlivých zamestnancov obsahujúce jednak najdôležitejšie informácie o zamestnancovi, jednak hlavičku disponujúcu možnosťami rýchlych akcií nad daným zamestnancom – konkrétne o možnosti úpravy či vymazania záznamu zamestnanca. Informácie prehľadu sú načítané asynchrónne ihneď po inicializácii zobrazenia.

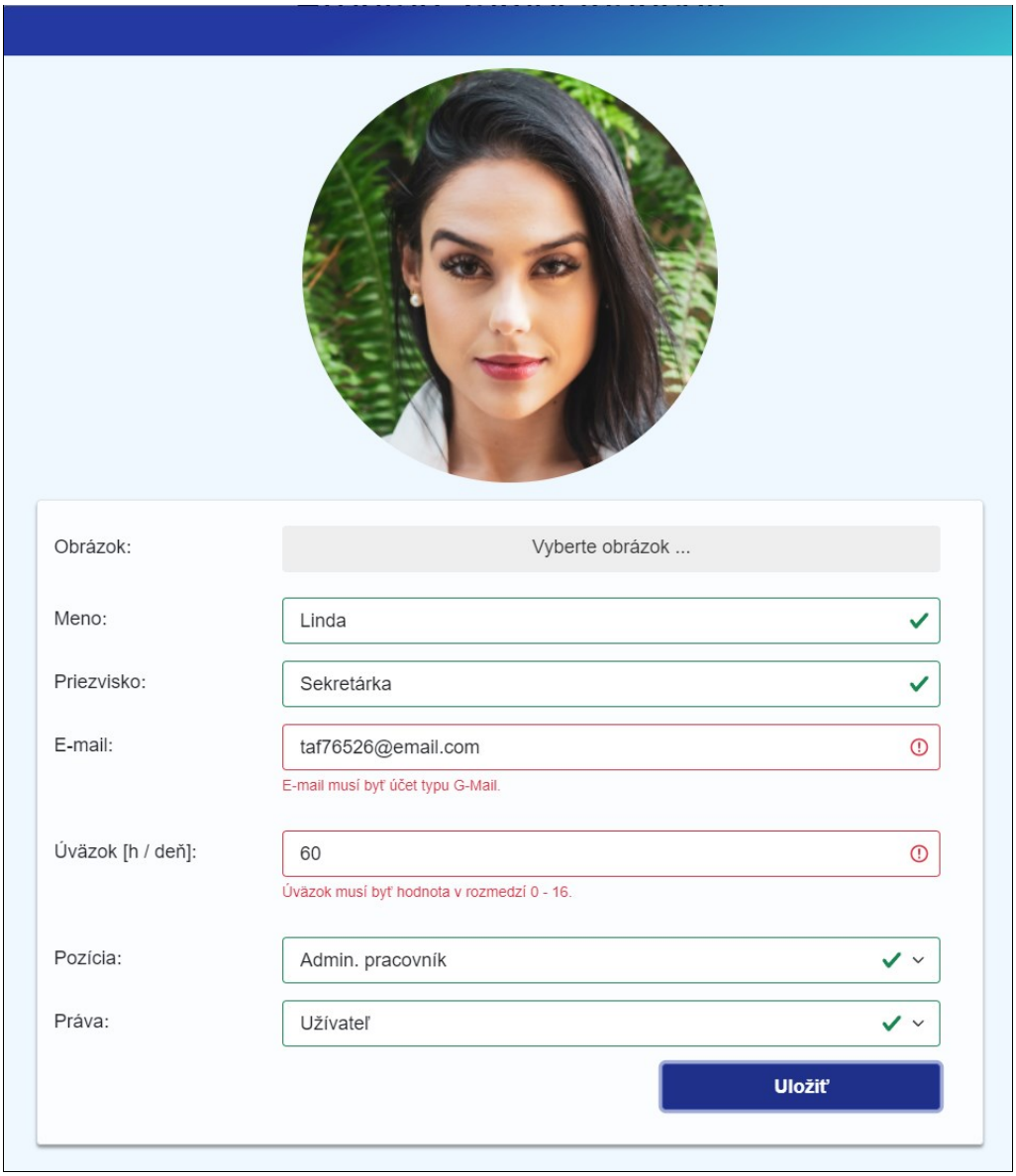
7.1.2 Rýchle akcie

Stránka správy zamestnancov okrem základného zobrazenia poskytuje možnosť výkonu viacerých rýchlych akcií nad záznamom zamestnanca (existujúcim, alebo potenciálnym).



Obrázok 47. Formulár určený k vytvoreniu nového záznamu zamestnanca

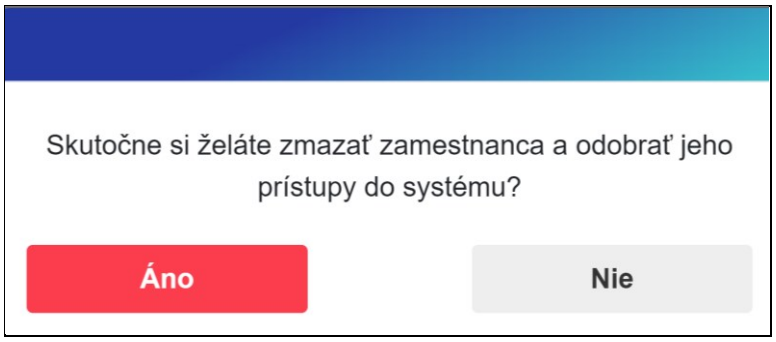
Formulár pridania zamestnanca je plne validovaným formulárom umožňujúcim zadať základné údaje zamestnanca, vrátane jeho profilovej fotky. Akcia taktiež presahuje do funkcionality nepríliš viditeľnej z prostredia *UI* – nadväzuje na ňu založenie užívateľského účtu a vygenerovanie príslušných údajov v prípade, že vytváraný užívateľ má disponovať prístupmi do systému.



Obrázok 48. Modálne okno určené k editácii existujúceho záznamu zamestnanca [Obrázok prevzatý z: 212]

Editačné modálne okno funguje na podobnom princípe ako formulár pridávací. Otvára sa kliknutím na ikonku editácie v karte zamestnanca a po otvorení disponuje aktuálnymi údajmi daného záznamu. Vstupy modálneho okna sú plne validované.

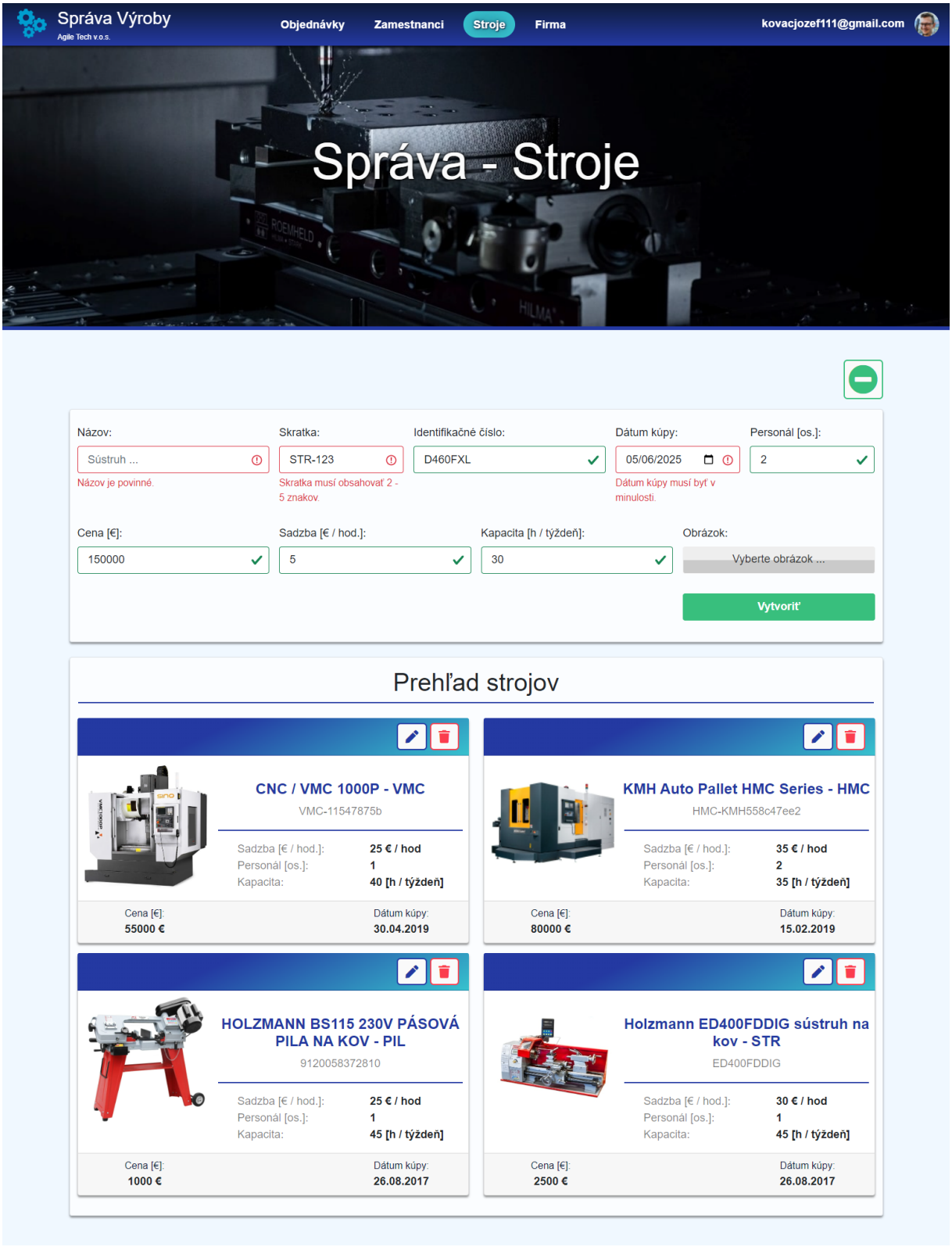
Vykonaná modifikačná akcia má dopad aj na užívateľské práva pridružené k danému záznamu – pokiaľ bol E-mail a prístupové oprávnenia v editačnom formulári odstránené, daný zamestnanec sa už nebude môcť pod svojim užívateľským účtom naďalej prihlasovať do systému.



Obrázok 49. Modálne okno určené k potvrdeniu vymazania záznamu zamestnanca

Mazacie modálne okno sa otvára kliknutím na ikonku výmazu v hlavičke karty zamestnanca a po otvorení sa pýta prihláseného užívateľa, či si skutočne želá akciu vykonať – v prípade, že bude zvolená možnosť „Áno“, dochádza k odstráneniu údajov zamestnanca vrátane jeho potenciálnych prístupových práv do systému.

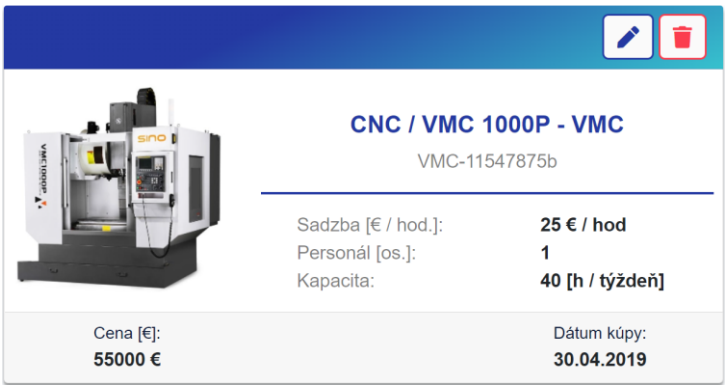
7.2 Správa strojov



Obrázok 50. Webstránka správy strojov [Obrázky prevzaté z: 198, 199, 200, 201, 218]

Správa strojov je principiálne veľmi podobná správe zamestnancov – jedná sa o prostriedok k evidencii výrobných zariadení v prostredí firmy – umožňuje primárne *CRUD* operácie nad entitami strojov. Kopíruje funkčné podcelky modulu zamestnancov a ich zameranie na osobitné časti *CRUD* funkcionality.

7.2.1 Prehľad strojov

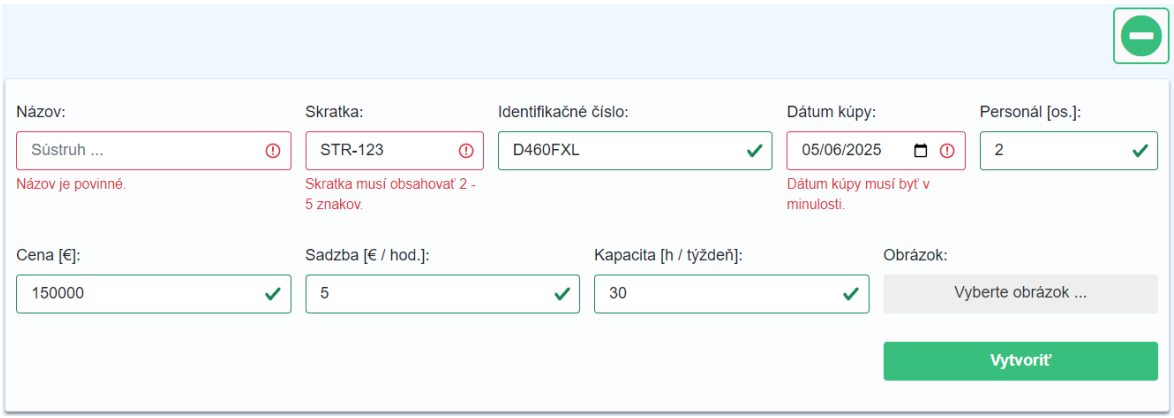


Obrázok 51. Kartička s informáciami stroja v prehľade [Obrázok prevzatý z: 198]

Prehľad strojov je hlavným obsahom tejto webstránky. Priestor je využitý „kartičkami“ reprezentujúcimi jednotlivé výrobné zariadenia udávajúce ich parametre – či už čisto pre evidenčné účely, alebo pre kalkulačné metódy v rámci administratívy objednávok. Každá karta taktiež disponuje hlavičkou s možnosťou rýchlych akcií nad daným zariadením – konkrétne o možnosti úpravy či vymazania záznamu. Informácie prehľadu sú načítané asynchrónne ihneď po inicializácii zobrazenia.

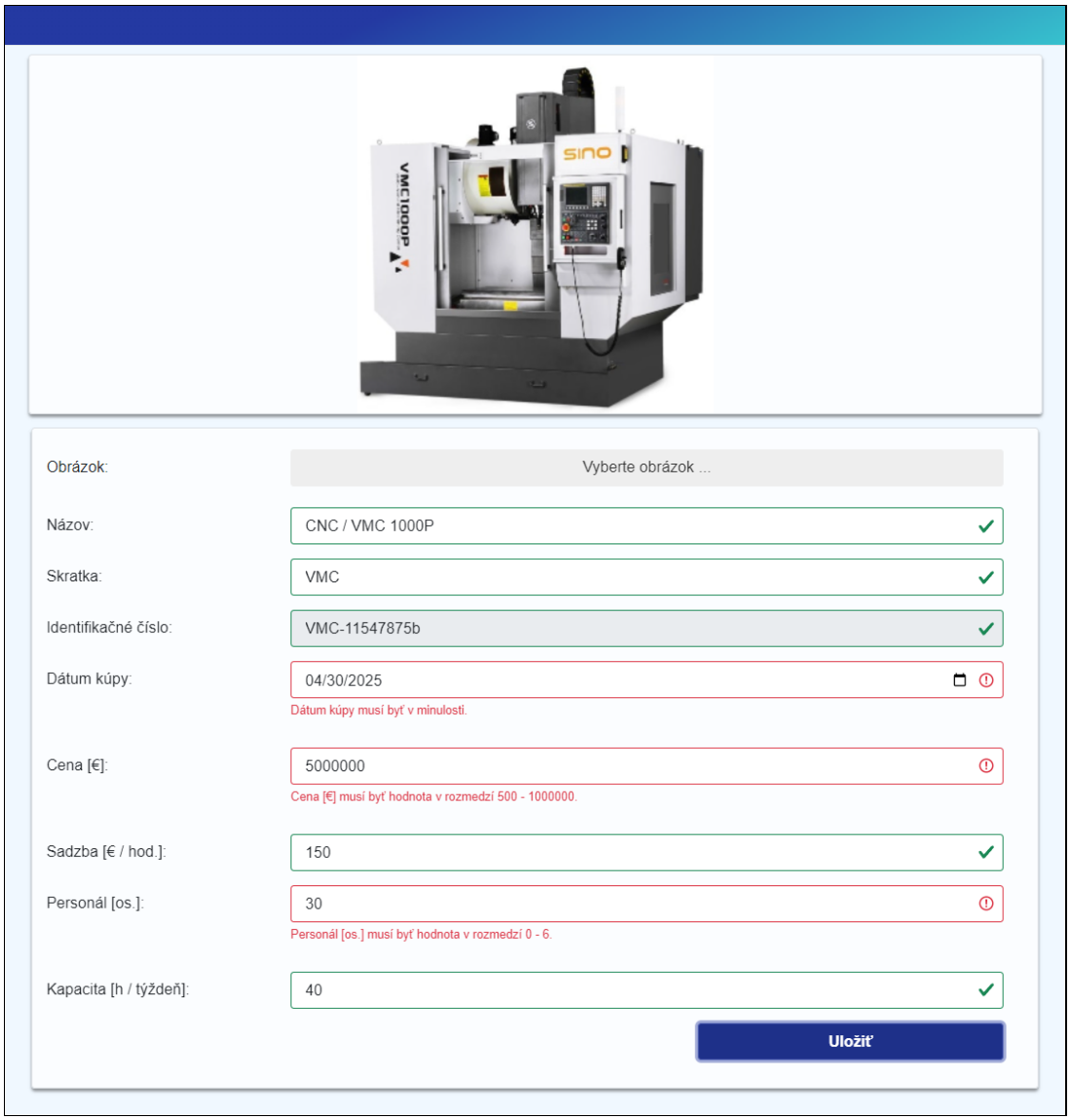
7.2.2 Rýchle akcie

Správa výrobných zariadení pracuje na podobnom princípe ako správa zamestnancov – umožňuje vykonávať niekoľko rýchlych akcií nad pomyselnou entitou výrobného stroja.



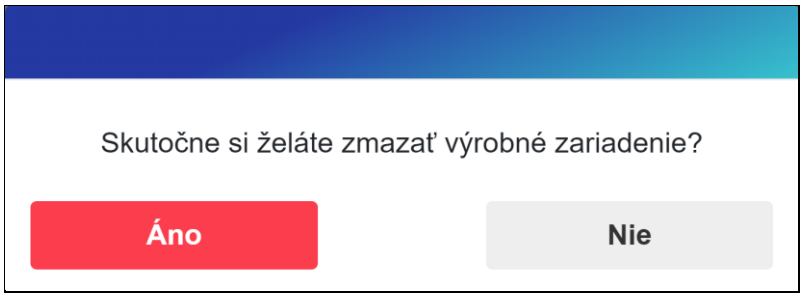
Obrázok 52. Formulár určený k vytvoreniu nového záznamu stroja

Formulár pridania výrobného zariadenia je plne validovaným formulárom umožňujúcim zadať základné údaje stroja, vrátane jeho náhľadového obrázku.



Obrázok 53. Modálne okno určené k editácii existujúceho záznamu stroja [Obrázok prevzatý z: 198]

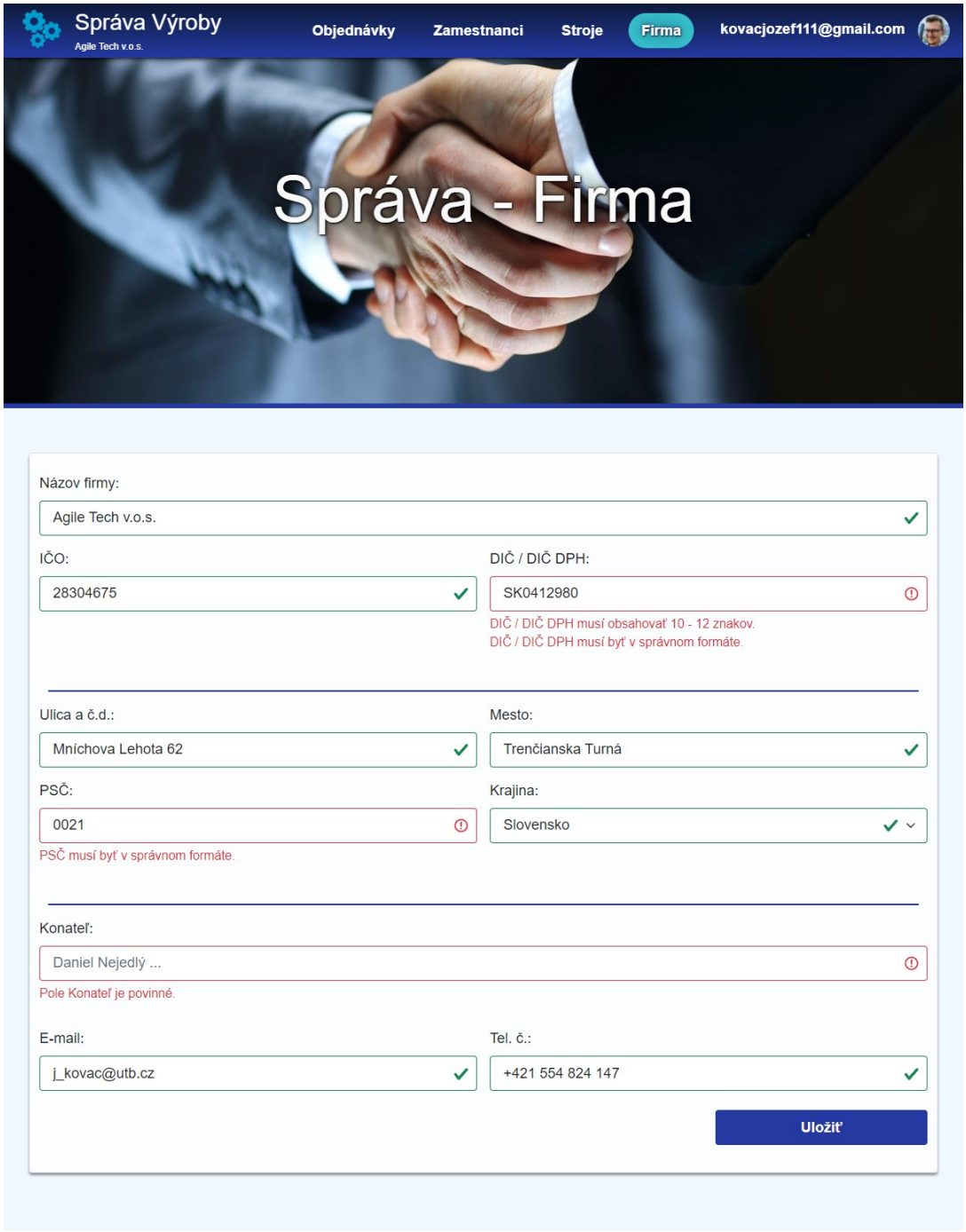
Editačné modálne okno funguje na podobnom princípe ako formulár pridávací. Otvára sa kliknutím na ikonku editácie v hlavičke karty stroja a po otvorení disponuje aktuálnymi údajmi daného záznamu. Analogicky ako v pridávacom formulári, aj v ňom sú vstupy plne validované.



Obrázok 54. Modálne okno určené k potvrdeniu vymazania záznamu stroja

Mazacie modálne okno sa otvára kliknutím na ikonku výmazu v hlavičke karty stroja a po otvorení sa pýta prihláseného užívateľa, či si skutočne želá akciu vykonať – v prípade, že bude zvolená možnosť „Áno“ dochádza k odstráneniu údajov výr. zariadenia.

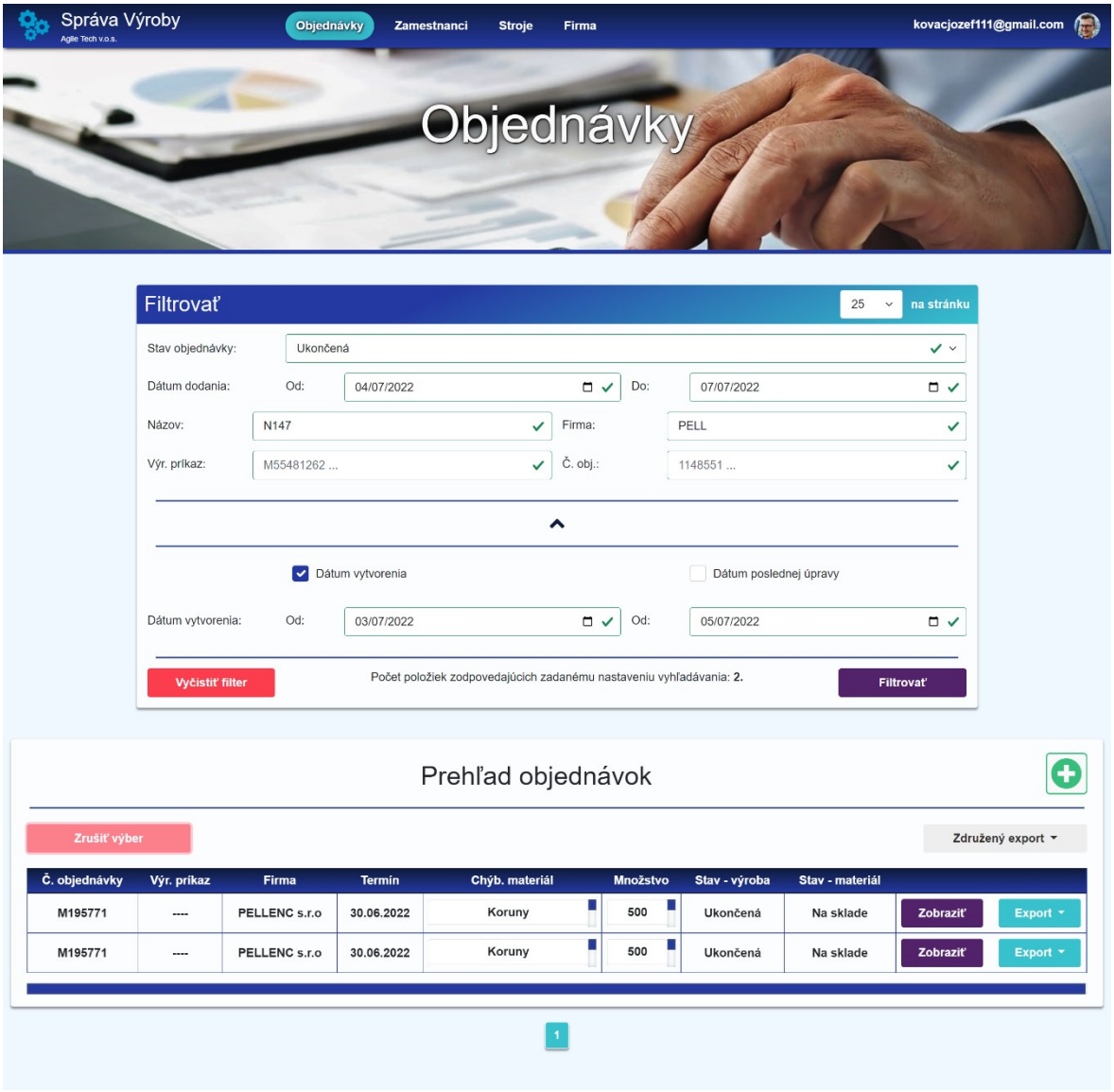
7.3 Správa firemných údajov



Obrázok 55. Webstránka správy firemných údajov [Obrázok prevzatý z: 219]

Stránka firemných údajov je stránkou pozostávajúcou primárne z jediného veľkého editačného formulára – tento formulár má za úlohu umožniť úpravu informácií o firme, ktoré sú následne premietané do mnohých ďalších funkcionalít naprieč aplikáciou, napr. do exportných tlačív objednávok. Formulár samotný – podobne ako ostatné formuláre – disponuje validačnými mechanizmami.

7.4 Správa objednávok



Obrázok 56. Webstránka správy objednávok [Obrázok prevzatý z: 220]

Hlavným modulom systému správy objednávok je neprekvapivo samotná webstránka obsahujúca zoznam objednávok a pridružené ovládacie prvky. Modul objednávok vďaka svojmu rozsahu v podkladovej logika zapája nejakou formou prakticky všetky ostatné logické moduly a je v určitom zmysle kulmináciou funkcionality poskytovanou podpornými časťami systému.

7.4.1 Prehľad objednávok a filtrácia

Dominantou vrchnej časti stránky manipulujúcej s objednávkovými položkami je filtračný formulár umožňujúci obmedziť výber zobrazovaných položiek na základe viacerých kľúčových parametrov. Tento formulár – podobne ako ostatné formuláre v rámci systému – disponuje vlastnými validáciami, aj keď výrazne benevolentnejšími ako u formulárov manipulujúcich s perzistentnými dátami z toho dôvodu, že užívateľovi môže postačovať filtrácia podľa menšieho množstva kritérií. Zoznam objednávok je naviac stránkovaný, čo ďalej šetrí objem dátového prenosu a motivuje užívateľa využiť rýchle *Frontend*-ovi radenie zabudované do použitého komponentu dátovej mriežky.

Hodnoty zadané vo formulári filtračnej sekcie v čase potvrdenia filtračnej operácie ovplyvňujú stav objednávok v zozname umiestnenom priamo pod ňou. Za účelom ušetrenia dátového prenosu medzi klientom a serverom sú v zozname zobrazované iba krátke náhľady objednávok umožňujúce bezproblémovú identifikáciu jednotlivých položiek – kompletné informácie o položke sa načítavajú dodatočne až v momente snahy užívateľa s touto položkou manipulovať.

7.4.2 Manipulácia s objednávkou

Užívateľ dokáže s objednávkou priamym spôsobom manipulovať troma rôznymi spôsobmi – môže využiť pridávacie akčné tlačidlo umiestnené nad zoznamom existujúcich objednávok pre vytvorenie novej objednávky, môže existujúcu objednávku v niektorom z aktívnych stavov otvoriť za účelom editácie (editačný režim objednávkového formuláru pochopiteľne zaručí vyplnenie vstupných polí prostredníctvom aktuálne uložených dát) a taktiež smie na objednávku bez možnosti editácie nahliadať – táto možnosť je prístupná v prípade, že sa objednávka nachádza v niektorom z definitívne uzatvorených stavov – napr. „Ukončená“ či „Zrušená“.

Objednávkový formulár samotný je na základe logicky súvisiacich informácií členený do sekcií zložených z užívateľsky manipulovateľných polí rôznych dátových typov. Nie všetky polia objednávky sú užívateľsky upravovateľné – niektoré z nich reprezentujú dopočítavané parametre u ktorých síce je dôležitá ich vizualizácia užívateľovi, ale ich uloženie už nie je kľúčové, keďže sa jedná iba o deterministickým výpočtom získateľný údaj závislý na hodnotách ostatných vstupov.

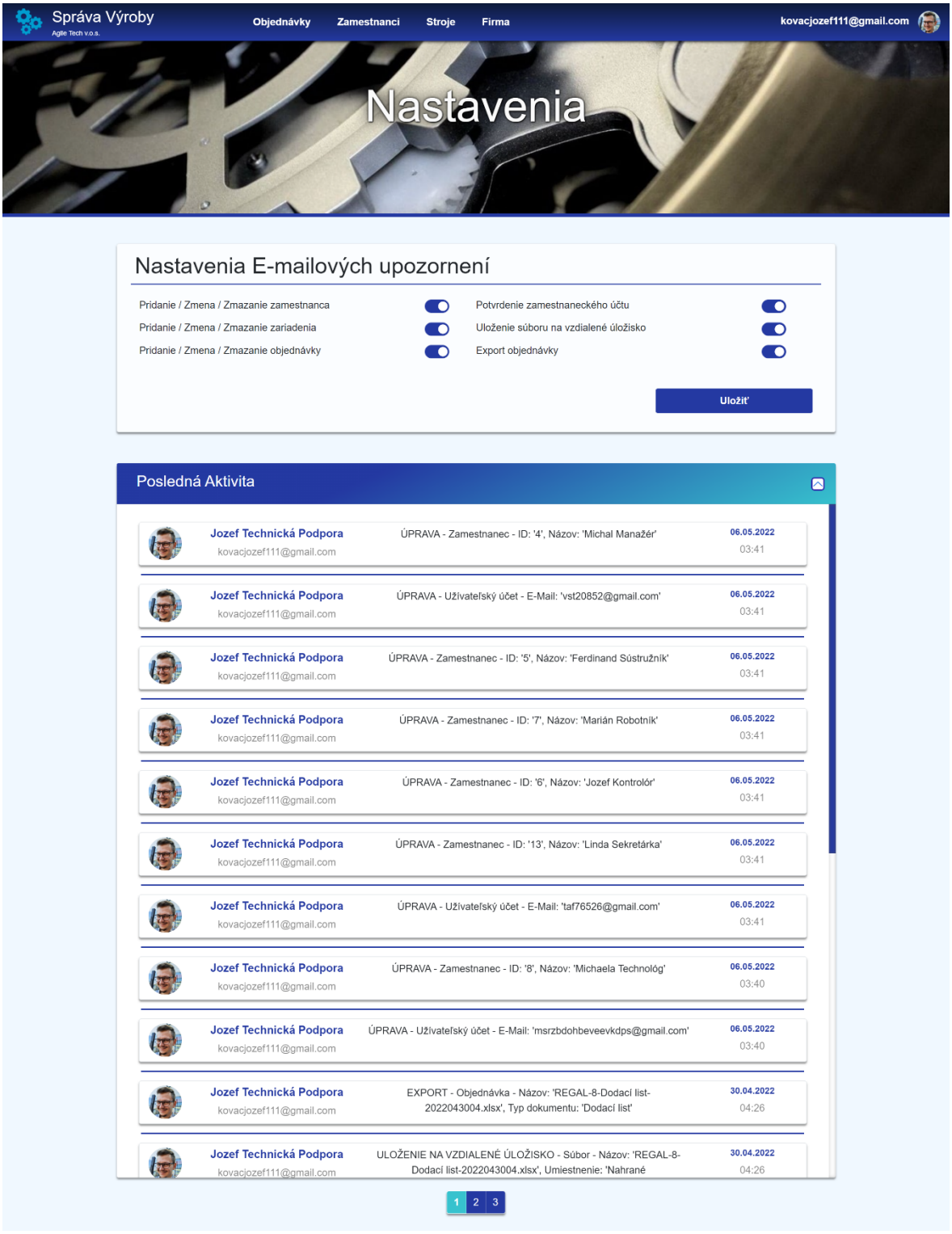
[Drôtený model editačného modálneho okna pre úpravu objednávky:

viď. *Príloha č. I* → DP_FIG_JozefKovac_A20132.pdf → *strana č. 13*]

7.4.3 Export

Správa objednávok obsahuje okrem zabezpečenia *CRUD* funkcionality nad jednotlivými objednávkami naviac možnosti exportu objednávkových tlačív vo formáte *.xlsx.* Podporované možnosti exportu zahŕňajú export jednoduchý (z jednej objednávky, priamo na riadku v prehľade objednávok – dokument typu „Sprievodný List“, alebo export združený (z jednej a viacerých objednávok), podporujúci dokumenty typu „Dodací list“ a „Fakturácia“. Akčné tlačidlo združeného exportu sa nachádza priamo nad zoznamom objednávok. Okrem vygenerovania a stiahnutia exportovaného tlačiva zabezpečuje funkcia exportu aj uloženie kópie dokumentu na externé úložisko *Google Drive* za využitia *API* vedeného firmou *Google*.

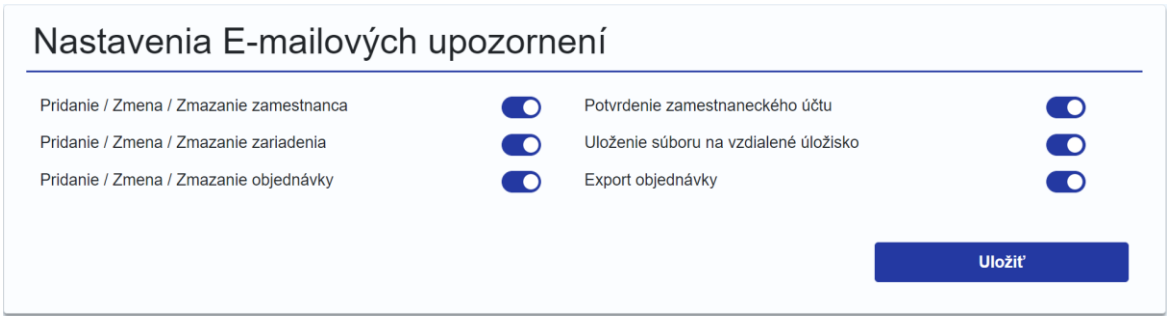
7.5 Užívateľské nastavenia



Obrázok 57. Webstránka správy užívateľských nastavení [Obrázok prevzatý z: 221]

Stránka užívateľských nastavení v sebe skrýva pre prihláseného užívateľa možnosti editácie jeho osobných preferencií ohľadom nastavenia typov notifikácií o operáciách v systéme, ktoré mu budú odosielané formou *HTML* E-mailového upozornenia. Táto stránka je zároveň zaujímavá tým, že na ňu síce užívatelia disponujúci ľubovoľnou úrovňou užívateľských oprávnení, ale bežným užívateľom nie je prístupný kompletný prehľad spätnej aktivity.

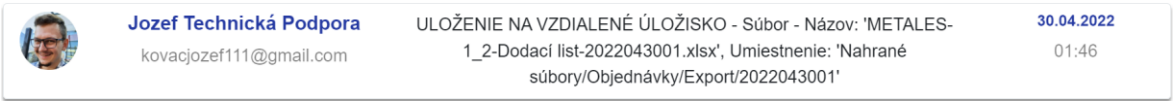
7.5.1 Nastavenia



Obrázok 58. Formulár úpravy užívateľských nastavení E-mailových notifikácií

Formulár užívateľských nastavení je štrukturálne nepríliš náročným prostriedkom k upravovaniu sady užívateľsky ovplyvniteľných pravdivostných vlajok, prostredníctvom *Frontend*-u zobrazených formou prepínačov. Po kliknutí na tlačidlo uloženia dochádza k odoslaniu nastavenia na *API* a jeho prepisu do perzistentného úložiska – po tomto bode už bude z pohľadu serveru nastavenie viditeľné a rešpektované relevantnou E-mailovou notifikačnou službou.

7.5.2 Aktivita



Obrázok 59. Záznam o užívateľskej aktivite viditeľný vo výpise aktivít na stránke nastavení

Ďalším prvkom zahrnutým na stránke užívateľských nastavení je záznam užívateľskej aktivity – jedná sa vlastne o chronologicky zoradený, stránkovaný prehľad krátkych správ, ktoré administrátorovi umožňujú nadobudnúť prehľad o ľubovoľnej aktivite, ku ktorej v prostredí systému došlo. Viditeľnosť a možnosť manipulácie s týmto prehľadom je striktne obmedzená na úlohu administrátora systému – bežný užívateľ ho na stránke nastavení neuvidí.

ZÁVER

Výstupom teoretickej časti diplomovej práce je v súlade s bodmi zadania stručná analýza technológií určených pre vývoj a zabezpečenie webových aplikácií. Problematika je spracovaná formou popisnej enumerácie mnohých technológií a konkrétnych, na nich postavených praktických riešení za poskytnutia charakterizácie ich kľúčových znakov, použiteľnosti, obľúbenosti a celkovej pozície na súčasnom trhu. Práca sa ďalej zaoberá oboznámením čitateľa s rozpoznávacími prvkami a modelmi nasadenia týchto technológií, stanovením ich slabých a silných stránok a prípadným určením vhodnosti aplikácie jednotlivých riešení v presne stanovených prípadoch, mnohokrát zohľadnených už v ich samotných konštrukčných princípoch. Zvláštny dôraz pri spracovaní popisu technológií a vhodnosti ich použitia v kontexte tvorby webových systémov stredného rozsahu bol kladený na frameworky *.NET Core* a *ASP .NET Core Blazor* z dôvodu ich významu pre konštrukciu praktickej časti práce. Ako v obecnom prehľade znalostí, tak v kapitolách vyhradených frameworku *ASP .NET Core Blazor* je naviac neustále prítomná otázka zabezpečenia webového systému zameriavajúca sa na konkrétne mechanizmy prakticky využívané k zaisteniu čo najväčšej odolnosti aplikácie proti manipulácii neželanej povahy.

Teoretická časť diplomovej práce sa taktiež vo veľkej miere zaoberá spracovaním informácií nadobudnutých literárnou rešeršou zameranou na odhalenie architektonických a návrhových vzorov vyznačujúcich sa svojou popularitou a vhodnosťou pre oblasť vývoja webových aplikácií. Tieto architektonické vzory boli dôkladne charakterizované a popísané – práca sa usiluje o stanovenie ich významu a ústredných myšlienok, poskytnutie popisu ich silných stránok a odôvodnenie ich úlohy vo vývoji moderného webového softwaru.

Analytická fáza praktickej časti práce mala za úlohu zhotoviť zatiaľ hypotetický systém pre správu objednávok na konceptuálnej úrovni, a síce s využitím metód softwarového inžinierstva a analytických postupov aplikovaných v oblasti vývoja softwarových produktov. Podoba systému pre správu objednávok bola týmto spôsobom vopred naplánovaná, navrhnutá a načrtnutá postupom zahŕňajúcim analýzu a spracovanie funkčných a nefunkčných požiadaviek, ich pretavenie do diagramu prípadov použitia a následným stručným náčrtom ostrej implementácie prostredníctvom tvorby modelu tried (zachytávajúceho dátové entity v rámci systému) a jednoduchých drôtených modelov grafického užívateľského rozhrania webového prostredia aplikácie.

Implementačná fáza vývojového cyklu systému pre správu objednávok sa zameriavala prioritne na programátorskú prácu s frameworkom *ASP .NET Core Blazor* a s ním kompatibilnými webovými technológiami (*HTML*, *CSS*, *JavaScript*) v spojení s relačnou databázou *MS SQL Server*. Kapitoly popisujúce proces a špecifiká implementácie systému sa v súlade so zadaním práce orientujú jednak na prezentáciu a objasnenie technologického riešenia a previazania jeho základov s konkrétnymi architektonickými a návrhovými vzormi spracovanými v teoretickej časti práce, jednak na demonštráciu aplikácie z užívateľského pohľadu a vizuálnu prezentáciu hlavných funkčných modulov a ich významu pre koncového užívateľa. Zvýšená miera pozornosti – a s tým súvisiaca podkapitola nezanedbateľného rozsahu – bola venovaná taktiež mechanizmom a riešeniam adresujúcim obecné zabezpečenie vyvinutej webovej aplikácie. Popis nasadených bezpečnostných opatrení sa snaží prioritne riešiť najčastejšie a najzávažnejšie nedostatky a aplikovať riešenia v oblasti bezpečnostných slabých miest – od zaistenia šifrovaného prenosu dát medzi klientom a serverom, cez autorizačné a autentifikačné procesy až po implementáciu ochranných mechanizmov proti populárnym typom útokov.

Finálnym výsledkom diplomovej práce je tento dokument cielený na spracovanie problematiky písomnou formou a k nemu pridružené sprievodné materiály – menovite sa jedná o zdrojové kódy vytvoreného systému pre správu objednávok, diagramy a iné výstupy analytickej fázy praktickej časti práce a prípadné ďalšie prílohy uvedené v zozname príloh. Práca v rámci možností hojne narábala so zdrojmi uvedenými v literárnom prehľade – či už priamym čerpaním informácií referovaných primárne v teoretickej časti práce, alebo nadobudnutými znalosťami, najmä pri snahe o korektnú a efektívnu konštrukciu implementačných prvkov. Diplomová práca sa usilovala poskytnúť čo najkvalitnejšie spracovanie obšírnej problematiky – jej primárnym cieľom naprieč celým procesom tvorby bolo riadne pokryť všetky body zadania a venovať dostatočnú pozornosť aj mnohokrát zanedbávanej praktickej časti práce za účelom demonštrácie funkčnosti a efektivity teoreticky popísaných konceptov a konštruktov a ich bezproblémovej zlučiteľnosti s vývojom interných systémov menšieho až stredného rozsahu za využitia moderných webových technológií.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Internet use: Individuals using the Internet. *ITU: Committed to connecting the world* [online]. Geneva, Switzerland, 2022, 15.11.2021 [cit. 2022-05-09]. Dostupné z: https://www.itu.int/itu-d/reports/statistics/2021/11/15/internet-use/

[2] WEGNER, Philipp. Global IoT market size grew 22% in 2021: These 16 factors affect the growth trajectory to 2027. In: *IOT Analytics: Market insights for the Internet of Things* [online]. 2022, 30.3.2022 [cit. 2022-05-09]. Dostupné z: https://iot-analytics.com/iot-market-size/

[3] GANDHI, Prashant, Somesh KHANNA a Sree RAMASWAMY. Which Industries Are the Most Digital (and Why)?. *Harvard Business Review* [online]. Brighton, Massachusetts: Harvard Business Publishing, 2016, 1.4.2016, **2016**(4) [cit. 2022-05-10]. ISSN 0017-8012. Dostupné z: https://hbr.org/2016/04/a-chart-that-shows-which-industries-are-the-most-digital-and-why?referral=00060

[4] NAIK, Nitin, Paul JENKINS, Philip DAVIES a David NEWELL. Native Web Communication Protocols and Their Effects on the Performance of Web Services and Systems. *2016 IEEE International Conference on Computer and Information Technology (CIT)* [online]. IEEE, 2016, 2016, 219-225 [cit. 2022-05-10]. ISBN 978-1-5090-4314-9. Dostupné z: doi:10.1109/CIT.2016.100

[5] PETERSEN, Hillar. *From Static and Dynamic Websites to Static Site Generators*. Tartu, 2016. Bakalárska práca. UNIVERSITY OF TARTU - Institute of Computer Science. Vedoucí práce Karl Blum, MSc, Tõnu Tamme, MSc.

[6] What is Web Application Architecture? Components, Models, and Types: Models of Web Application Components. In: *Hackr.io: Programming* [online]. 3.3.2022 [cit. 2022-05-10]. Dostupné z: https://hackr.io/blog/web-application-architecture-definition-models-types-and-more

[7] OLUWATOSIN, Haroon Shakirat. Client-Server Model. *IOSR Journal of Computer Engineering* [online]. School of Computing Universiti Utara Malaysia Kedah, Malaysia, 2014, 2.2014, **2014**(1 - vol. 16), 57-71 [cit. 2022-05-10]. ISSN 22788727. Dostupné z: doi:10.9790/0661-16195771

[8] BOURAS, Christos a Agisilaos KONIDARIS. Modeling Data-Intensive Web Sites for Personalization, Integrity and Performance. *Information Modeling for Internet Applications* [online]. IGI Global, 2003 [cit. 2022-05-10]. ISBN 9781591400509. Dostupné z: doi:10.4018/9781591400509.ch012

[9] HUMAN, Hendrik. Difference Between Host and Server: Ultimate Guide 2022. In: *Review42* [online]. 2022, 15.4.2022 [cit. 2022-05-10]. Dostupné z: https://review42.com/resources/difference-between-host-and-server/

[10] MITCHELL, James G., Jay E. ISRAEL a Howard E. STURGIS. *Separating data from function in a distributed file system*. Palo Alto Research Center: Xerox Corporation, 1978, **1978**, 13.

[11] Client-Server: Benefits of Client-Server Computing. *Heavy.AI: Analytics for Decision-Making* [online]. 2022 [cit. 2022-05-10]. Dostupné z: https://www.heavy.ai/technical-glossary/client-server

[12] BOTHA, Reinhardt A. *Information Security in the client / server environment*. Nelson Mandela University, 1997. Diplomová práca. Nelson Mandela University. Vedoucí práce Prof J.H.P. Eloff.

[13] SINHA, Alok. Client-server computing: Time-Shared computing. *Communications of the ACM* [online]. 1992, 1992, **1992**(7 - vol. 35), 77-98 [cit. 2022-05-10]. ISSN 0001-0782. Dostupné z: doi:10.1145/129902.129908

[14] KUMAR, Saurabh. What is the Difference Between Thick Client and Thin Client?. *IT Junction: Simplifying Digitalization* [online]. 3.7.2019 [cit. 2022-05-10]. Dostupné z: https://itjunction.org/2019/10/05/what-is-the-difference-between-thick-client-and-thin-client/

[15] GREGERSEN, Erik, ed. Server: Computing. In: *Encyclopædia Britannica* [online]. [cit. 2022-05-10]. Dostupné z: https://www.britannica.com/technology/server

[16] POSEY, Brien. What is a Server?: Types of servers. In: *WhatIs.com: Network Hardware* [online]. Newton, Massachusetts, United States, 2022, 8.2021 [cit. 2022-05-10]. Dostupné z: https://www.techtarget.com/whatis/definition/server

[17] What is the difference between webpage, website, web server, and search engine?: Summary. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 1998, 27.4.2022 [cit. 2022-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines

[18] ayushbansal_, rajeev0719singh a itskawal2000, ed. Static vs Dynamic Website: Types of Website. In: *Geeks for Geeks* [online]. Noida, Uttar Pradesh, 7.7.2021 [cit. 2022-05-10]. Dostupné z: https://www.geeksforgeeks.org/static-vs-dynamic-website/

[19] QI, Xiaoguang a Brian D. DAVISON. Web page classification. *ACM Computing Surveys* [online]. Lehigh University, 2009, **2009**(2 - vol. 41), 1-31 [cit. 2022-05-10]. ISSN 0360-0300. Dostupné z: doi:10.1145/1459352.1459357

[20] NIXON, Robin. *Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites*. 2nd ed. Sebastopol: O'Reilly, 2012. ISBN 978-1-449-31926-7.

[21] Communication Protocol: What Does Communication Protocol Mean?. In: *Techopedia* [online]. 2011, 30.9.2020 [cit. 2022-05-10]. Dostupné z: https://www.techopedia.com/definition/25705/communication-protocol

[22] HALL, Eric A., LOUKIDES, Mike, ed. O'REILLY & ASSOCIATES. *Internet Core Protocols: The Definitive Guide*. Sebastopol: O'Reilly Media, 2000, 472 s. 1st ed. ISBN 1-56592-572-6.

[23] GOURLEY, David, Brian TOTTY, Marjorie SAYER, Sailu REDDY a Anshu AGGARWAL, MUI, Linda, ed. *HTTP: The Definitive Guide* [online]. Sebastopol: O'Reilly Media, 2002 [cit. 2022-05-11]. ISBN 978-1-56592-509-0.

[24] AFSARI, Kereshmeh, Charles M. EASTMAN a Dennis R. SHELDEN. Data Transmission Opportunities for Collaborative Cloud-Based Building Information Modeling. *Blucher Design Proceedings* [online]. São Paulo: Editora Blucher, 2016, 2016, **2016**(vol. 3), 907-913 [cit. 2022-05-11]. Dostupné z: doi:10.5151/despro-sigradi2016-448

[25] FIELDING, R., J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH a T. BERNERS-LEE. *Hypertext Transfer Protocol -- HTTP/1.1* [online]. 1999 [cit. 2022-05-11]. Dostupné z: https://www.hjp.at/doc/rfc/rfc2616.html. Návrh štandardu.

[26] NAIK, Nitin, Paul JENKINS, Philip DAVIES a David NEWELL. Native Web Communication Protocols and Their Effects on the Performance of Web Services and Systems. *2016 IEEE International Conference on Computer and Information Technology (CIT)* [online]. IEEE, 2016, 2016, **2016**, 219-225 [cit. 2022-05-11]. ISBN 978-1-5090-4314-9. Dostupné z: doi:10.1109/CIT.2016.100

[27] CHEN, Jiajia a Weiqing CHENG. Analysis of web traffic based on HTTP protocol. *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* [online]. IEEE, 2016, 2016, **2016**, 1-5 [cit. 2022-05-11]. Dostupné z: doi:10.1109/SOFTCOM.2016.7772120

[28] COOKSEY, Brian. Chapter 2: Protocols: HTTP Requests. In: *Zapier* [online]. 22.4.2014 [cit. 2022-05-11]. Dostupné z: https://zapier.com/learn/apis/chapter-2-protocols/

[29] ASHEN GAMAGE, Thilina. Evolution of HTTP — HTTP/0.9, HTTP/1.0, HTTP/1.1, Keep-Alive, Upgrade, and HTTPS: Understanding how HTTP works in the real world. In: *Medium* [online]. 17.11.2017 [cit. 2022-05-11]. Dostupné z: https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0

[30] HTTP request methods. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 2022, 3.10.2021 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[31] HTTP requests. In: *IBM: Documentation* [online]. Armonk, New York, 13.10.2021 [cit. 2022-05-11]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests

[32] Appendix B: Hypertext Transfer Protocol. *Sun ONE: Open Net Environment* [online]. Austin, Texas, United States: Oracle, 2004 [cit. 2022-05-11]. Dostupné z: https://docs.oracle.com/cd/E19857-01/817-6247-10/agaphttp.html

[33] HTTP responses. In: *IBM: Documentation* [online]. Armonk, New York, 13.10.2021 [cit. 2022-05-11]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-responses

[34] HTTP response status codes. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 2022, 28.4.2022 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[35] HICKMAN, Kipp E.B. *The SSL Protocol* [online]. Netscape Communications Corp, 1994 [cit. 2022-05-11]. Dostupné z: http://www.webstart.com/jed/papers/HRM/references/ssl.html. Špecifikácia.

[36] DAS, Manik Lal a Navkar SAMDARIA. On the security of SSL/TLS-enabled applications. *Applied Computing and Informatics* [online]. 2014, **10**(1-2), 68-81 [cit. 2022-05-11]. ISSN 22108327. Dostupné z: doi:10.1016/j.aci.2014.02.001

[37] WILLIAMS, Lawrence. HTTP vs HTTPS: What is Difference Between HTTP and HTTPS? Full Form. In: *Guru99: Guru99 is totally new kind of learning experience* [online]. 2022, 5.3.2022 [cit. 2022-05-11]. Dostupné z: https://www.guru99.com/difference-http-vs-https.html

[38] OPPLIGER, R., R. HAUSER a D. BASIN. SSL/TLS Session-Aware User Authentication. *Computer* [online]. IEEE, 2008, **2008**(3 - vol. 41), 59-65 [cit. 2022-05-11]. ISSN 0018-9162. Dostupné z: doi:10.1109/MC.2008.98

[39] DV, OV, IV, and EV Certificates: What's the Difference Between DV, OV, IV, and EV Certificates?. In: *SSLI.com: Trust is what we do* [online]. Houston, United States, 17.12.2021 [cit. 2022-05-11]. Dostupné z: https://www.ssl.com/article/dv-ov-and-ev-certificates/

[40] Web Technology: Frontend Languages. In: *Geeks for Geeks* [online]. Noida, Uttar Pradesh, 30.9.2021 [cit. 2022-05-11]. Dostupné z: https://www.geeksforgeeks.org/web-technology/#:~:text=Web%20Technology%20refers%20to%20the,used%20to%20access%20web%20pages.&text=Hyperlinked%20resources%20on%20the%20World,interfaces%20provided%20by%20Web%20browsers.

[41] Markup Language: Markup Language Definition. In: *TechTerms.com: The Computer Dictionary* [online]. Sharpened Productions, 1.6.2011 [cit. 2022-05-11]. Dostupné z: https://techterms.com/definition/markup_language

[42] HTML: HyperText Markup Language. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 2.5.2022 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[43] HTML Tree Structure: 768x453. In: *InterviewBit: Everything you need to crack your Next Tech Interview* [online]. 10.2021 [cit. 2022-05-11]. Dostupné z: https://www.interviewbit.com/blog/wp-content/uploads/2021/10/HTML-Tree-Structure-768x453.png

[44] Usage statistics of markup languages for websites. In: *W3Techs: Web Technology Surveys* [online]. 10.5.2022 [cit. 2022-05-11]. Dostupné z: https://w3techs.com/technologies/overview/markup_language

[45] PEDAMKAR, Priya. Versions of HTML: Introduction to Versions of HTML. *Educba: Software Development* [online]. Powai, Mumbai, Maharashtra, India [cit. 2022-05-11]. Dostupné z: https://www.educba.com/versions-of-html/

[46] TABARÉS, Raúl. HTML5 and the evolution of HTML; tracing the origins of digital platforms. *Technology in Society* [online]. 2021, 5.2021, **2021**(vol. 65) [cit. 2022-05-11]. ISSN 0160791X. Dostupné z: doi:10.1016/j.techsoc.2021.101529

[47] HTML 4.01 Specification: W3C Recommendation 24 December 1999 - superseded 27 March 2018. *W3C: World Wide Web Consortium* [online]. 1997-1999 [cit. 2022-05-11]. Dostupné z: https://www.w3.org/TR/html401/

[48] Difference between HTML and HTML5?: Difference between Html and Html5. In: *JavaTPoint* [online]. [cit. 2022-05-11]. Dostupné z: https://www.javatpoint.com/html-vs-html5#:~:text=HTML%20has%20many%20updates%20over,full%20support%20for%20running%20JavaScript.

[49] HTML: Living Standard. *WHATWG: Web Hypertext Application Technology Working Group* [online]. 10.5.2022 [cit. 2022-05-11]. Dostupné z: https://html.spec.whatwg.org/multipage/

[50] LAWSON, Bruce a Remy SHARP. *Introducing HTML5*. Berkeley: New Riders PTG, 2011. Voices that matter. ISBN 9780321687296.

[51] CSS: Cascading Style Sheets. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 2.5.2022 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[52] HTML & CSS: What is CSS?. *W3C: World Wide Web Consortium* [online]. 2016 [cit. 2022-05-11]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss

[53] CSS selectors. In: *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 25.4.2022 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors

[54] WAY, Jeffrey. The 30 CSS Selectors You Must Memorize: CSS Selectors. In: *EnvatoTuts+* [online]. 25.9.2020 [cit. 2022-05-11]. Dostupné z: https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048

[55] Part 3: Introduction to CSS. In: *Code.makery: Learning how to code* [online]. 10.8.2018 [cit. 2022-05-11]. Dostupné z: https://code.makery.ch/library/html-css/part3/css-rule.png

[56] CSS Syntax Module Level 3: W3C Candidate Recommendation Draft. *W3C: World Wide Web Consortium* [online]. 24.12.2021 [cit. 2022-05-11]. Dostupné z: https://www.w3.org/TR/css-syntax-3/

[57] New CSS3 Features: Learn How This Version Improved CSS2. In: *Bitdegree: learn* [online]. 2014, 23.1.2020 [cit. 2022-05-11]. Dostupné z: https://www.bitdegree.org/learn/css3-features

[58] BOS, Bert. CSS current work & how to participate. In: *W3C: World Wide Web Consortium* [online]. 8.5.2022 [cit. 2022-05-11]. Dostupné z: https://www.w3.org/Style/CSS/current-work

[59] BOSE, Shreya. Top 5 CSS Frameworks for Developers and Designers: Advantages of using CSS Frameworks. *BrowserStack* [online]. 2011, 22.6.2021 [cit. 2022-05-11]. Dostupné z: https://www.browserstack.com/guide/top-css-frameworks

[60] Best CSS Frameworks in 2022. *Dev: Dev Community* [online]. 27.1.2022 [cit. 2022-05-11]. Dostupné z: https://dev.to/theme_selection/best-css-frameworks-in-2020-1jjh

[61] CSS Frameworks: Rankings. *The State of CSS 2021* [online]. 2021, 2021 [cit. 2022-05-11]. Dostupné z: https://2021.stateofcss.com/en-US/technologies/css-frameworks/

[62] *Bootstrap: Build fast, responsive sites with Bootstrap* [online]. [cit. 2022-05-11]. Dostupné z: https://getbootstrap.com/

[63] LAAZIRI, Majida, Khaoula BENMOUSSA, Samira KHOULJI, Kerkeb MOHAMED LARBI a Abir El YAMAMI. Analyzing bootsrap and foundation font-end frameworks: a comparative study. *International Journal of Electrical and Computer Engineering (IJECE)* [online]. 2019, **9**(1), 713-722 [cit. 2022-05-11]. ISSN 2088-8708. Dostupné z: doi:10.11591/ijece.v9i1.pp713-722

[64] *HEY LOOK, IT'S EVERY BOOTSTRAP WEBSITE EVER: The only bootstrap page* [online]. [cit. 2022-05-11]. Dostupné z: https://www.dagusa.com/

[65] Sass. *Foundation: The most advanced responsive front-end framework in the world* [online]. Foundation Yetinauts, 2020 [cit. 2022-05-11]. Dostupné z: https://get.foundation/sites/docs/sass.html

[66] DEGROAT, T. J. The History of JavaScript: Everything You Need to Know. *Springboard* [online]. 19.8.2019 [cit. 2022-05-11]. Dostupné z: https://www.springboard.com/blog/data-science/history-of-javascript/#:~:text=The%20name%20JavaScript%20came%20from,about%20language%20at%20the%20time.

[67] JavaScript. *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 27.4.2022 [cit. 2022-05-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[68] WIRFS-BROCK, Allen a Brendan EICH. JavaScript: the first 20 years. *Proceedings of the ACM on Programming Languages* [online]. 2020, **4**(HOPL), 1-189 [cit. 2022-05-11]. ISSN 2475-1421. Dostupné z: doi:10.1145/3386327

[69] OZEROVA, Elena. JAVASCRIPT SPECIFICS: ASYNCHRONICITY. *Sigma Software: Your Reliable Technology Partner* [online]. Kharkiv, Ukrajina, 21.6.2021 [cit. 2022-05-11]. Dostupné z: https://sigma.software/about/media/javascript-specifics-asynchronicity

[70] YIM, Vivian. Synchronous vs. Asynchronous JavaScript. In: *Medium: Stay curious* [online]. 28.1.2022 [cit. 2022-05-11]. Dostupné z: https://miro.medium.com/max/1400/1*l5gIa6aTae0RhlQzC0CDrQ.png

[71] JavaScript Callbacks: What are callbacks. *JavaScript Tutorial* [online]. [cit. 2022-05-11]. Dostupné z: https://www.javascripttutorial.net/javascript-callback/

[72] DELCEV, Sanja a Drazen DRASKOVIC. Modern JavaScript frameworks: A Survey Study. *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)* [online]. IEEE, 2018, 2018, 106-109 [cit. 2022-05-11]. ISBN 978-1-5386-4927-5. Dostupné z: doi:10.1109/ZINC.2018.8448444

[73] ARORA, Simran Kaur. 10 Best JavaScript Frameworks to Use in 2022: Top 10 JavaScript Frameworks. *Hackr.io: Programming* [online]. 29.4.2022 [cit. 2022-05-11]. Dostupné z: https://hackr.io/blog/best-javascript-frameworks

[74] Front-end Frameworks: Front-end frameworks and libraries. *The State of JS 2021* [online]. 2021, 2021 [cit. 2022-05-11]. Dostupné z: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/

[75] Introduction: What is Vue. *Vue.js: The Progressive JavaScript Framework* [online]. 2014 [cit. 2022-05-11]. Dostupné z: https://vuejs.org/guide/introduction.html

[76] SO, Preston. Vue.js. *Decoupled Drupal in Practice* [online]. Berkeley, CA: Apress, 2018, 13.12.2018, 381-397 [cit. 2022-05-11]. ISBN 978-1-4842-4071-7. Dostupné z: doi:10.1007/978-1-4842-4072-4_20

[77] HAVIV, Amos Q. *MEAN Web Development: Develop your real-time MEAN application efficiently using a combination of MongoDB, Express, Angular and Node* [online]. 2nd ed. Birningham - Mumbai: Packt Publishing, 2016 [cit. 2022-05-11]. ISBN 978-1-78588-630-0. Dostupné z: https://books.google.cz/books?id=n5vcDgAAQBAJ&printsec=frontcover&dq=editions:1SOSukNAvh0C&hl=cs&sa=X&redir_esc=y#v=onepage&q&f=false

[78] *Angular JS: Oficiálna dokumentácia frameworku Angular JS* [online]. 2021 [cit. 2022-05-11]. Dostupné z: https://angularjs.org/

[79] *JQuery - Write less, do more: Oficiálna dokumentácia knižnice JQuery* [online]. [cit. 2022-05-11]. Dostupné z: https://jquery.com/

[80] BIBEAULT, Bear, Yehuda KATZ a Aurelio DE ROSA. *JQuery in Action* [online]. 3rd ed. Manning Publications, 2015 [cit. 2022-05-11]. ISBN 9781617292071. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=ZzkzEAAAQBAJ&oi=fnd&pg=PT19&dq=jquery&ots=fijhXUd8PB&sig=0CKMoeyE7Pz0oxF_Ez1PG_P73Fw&redir_esc=y#v=onepage&q=jquery&f=false

[81] JQuery Usage Statistics. *Built With: Find out what websites are Built With* [online]. 2022 [cit. 2022-05-11]. Dostupné z: https://trends.builtwith.com/javascript/jQuery

[82] GACKENHEIMER, Cory. What Is React?. *Introduction to React* [online]. Berkeley, CA: Apress, 2015, 2015, 1-20 [cit. 2022-05-11]. ISBN 978-1-4842-1246-2. Dostupné z: doi:10.1007/978-1-4842-1245-5_1

[83] MARIANO, Carl Lawrence. *Benchmarking JavaScript Frameworks* [online]. 2017 [cit. 2022-05-11]. Dostupné z: https://web.archive.org/web/20200322022524id_/https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1100&context=scschcomdis. Dizertačná práca. Technological University Dublin - School of Computing.

[84] SINGHAL, Palak. Frontend vs Backend. *Geeks for Geeks* [online]. Noida, Uttar Pradesh, 31.8.2021 [cit. 2022-05-12]. Dostupné z: https://www.geeksforgeeks.org/frontend-vs-backend/

[85] GISSEL, Gunar. What Is Business Logic?. In: *Dev* [online]. 29.7.2018 [cit. 2022-05-12]. Dostupné z: https://dev.to/tiffany/what-is-business-logic-1gdf

[86] BONURA, Diego, Rosario CULMONE a Emanuela MERELLI. Patterns for web applications. *Proceedings of the 14th international conference on Software engineering and knowledge engineering - SEKE '02* [online]. New York, New York, USA: ACM Press, 2002, 2002, 739- [cit. 2022-05-12]. ISBN 1581135564. Dostupné z: doi:10.1145/568760.568887

[87] MINO, Santiago. 8 Reasons Why PHP Is Still So Important For Web Development. *Jobsity* [online]. 29.4.2022 [cit. 2022-05-12]. Dostupné z: https://www.jobsity.com/blog/8-reasons-why-php-is-still-so-important-for-web-development

[88] PHP Usage Statistics. In: *Built With: Find out what websites are Built With* [online]. Sydney NSW 2000, Australia, 2022 [cit. 2022-05-12]. Dostupné z: https://trends.builtwith.com/framework/PHP

[89] FLEURY, Daniel. 7 Global Websites That Use PHP in 2022. *TrioBlog: Grow your skills* [online]. 2022 [cit. 2022-05-12]. Dostupné z: https://www.trio.dev/blog/companies-using-php

[90] ROZHNOVSKY, Alexander. Why use PHP: Main advantages and disadvantages. *Light It Global* [online]. 2021 [cit. 2022-05-12]. Dostupné z: https://light-it.net/blog/why-use-php-main-advantages-and-disadvantages/

[91] LERDORF, Rasmus, Kevin TATROE a Peter MACINTYRE. *Programming PHP* [online]. 3rd ed. O'Reilly Media, 2013 [cit. 2022-05-12]. ISBN 9781449392772. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=7OjvOmol3CcC&oi=fnd&pg=PR9&dq=php&ots=1sRm4Ya8z8&sig=5cVBYojNY5U58OilEroNzblXeU8&redir_esc=y#v=onepage&q=php&f=false

[92] MARUYAMA, Hiroshi, Kent TAMURA, Naohiko URAMOTO, et al. *XML and Java: Developing Web Applications*. 2nd ed. Boston, MA 02116: Addison-Wesley, 2002. ISBN 0-201-77004-0.

[93] ARNOLD, Ken, James GOSLING a David HOLMES. *THE Java™ Programming Language*. 4th ed. Boston, Massachusetts, United States: Addison Wesley Professional, 2005. ISBN 0-321-34980-6.

[94] Your First Cup: Oficiálna dokumentácia firmy Oracle. *Documentation: Information about our products and services with targeted solutions, getting started guides, and content for advanced use cases.* [online]. Austin, Texas: Oracle, 2012 [cit. 2022-05-12]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

[95] Python Introduction: What is Python?. *W3Schools: Learn to Code* [online]. Refsnes Data [cit. 2022-05-12]. Dostupné z: https://www.w3schools.com/python/python_intro.asp

[96] TANEJA, Sheetal a Pratibha R. GUPTA. Python as a Tool for Web Server Application Development. *JIMS 8i-International Journal of Information, Communication and Computing Technology(IJICCT)* [online]. 2014, 1. 2014 - 6. 2014, (1 - vol. 2) [cit. 2022-05-12]. Dostupné z: https://www.jimsindia.org/8i_journal/volumeii/python-as-a-tool-for-web-server-application-development.pdf

[97] Top 10 Features Of Python You Need To Know. *Tech-Act: Actionable Certifications & Training* [online]. 31.1.2022 [cit. 2022-05-12]. Dostupné z: https://www.tech-act.com/blog/data-science/top-10-features-of-python-you-need-to-know/

[98] Most Popular Backend Frameworks – 2012/2021. In: *Statistics & Data* [online]. 2021 [cit. 2022-05-12]. Dostupné z: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/

[99] STAUFFER, Matt, YOUNG, Alicia, ed. *Laravel: Up & Running: A Framework for Building Modern PHP Apps* [online]. 2nd ed. Sebastopol: O'Reilly Media, 2019 [cit. 2022-05-12]. ISBN 978-1-492-04121-4. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=HcqPDwAAQBAJ&oi=fnd&pg=PP1&dq=laravel&ots=bgLU8xunGD&sig=ouYVO-ozmfVOvjQlPaG1AW9sM5E&redir_esc=y#v=onepage&q=laravel&f=false

[100] MCCOOL, Shawn. *Laravel Starter: The definitive introduction to the Laravel PHP web development framework*. Birningham - Mumbai: Packt Publishing, 2012. ISBN 978-1-78216-090-8.

[101] *Ruby on Rails: Technical overview* [online]. [cit. 2022-05-12]. Dostupné z: http://kelas-karyawan-bali.kurikulum.org/IT/en/2420-2301/Rails-web-framework_3884_kelas-karyawan-bali-kurikulumngetesumum.html

[102] TATE, Bruce A. a Curt HIBBS. *Ruby on Rails: Up and Running: Lightning Fast Web Development* [online]. Sebastopol: O'Reilly Media, 2006 [cit. 2022-05-12]. ISBN 978-0-596-10132-9. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=JJ7SHIRDfOcC&oi=fnd&pg=PR5&dq=ruby+on+rails&ots=JEy6Ll4VyZ&sig=InsKx0etYHM_Za_h9hx7jSobDP8&redir_esc=y#v=onepage&q&f=false

[103] HARTL, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails* [online]. 3rd ed. Boston, Massachusetts, United States: Addison-Wesley Professional, 2015 [cit. 2022-05-12]. ISBN 978-0-13-407770-3. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=0tSmCAAAQBAJ&oi=fnd&pg=PT24&dq=ruby+on+rails&ots=v-GMEe65T8&sig=j4ikq4jjyfyYRxf3Hy-GDXAy0TY&redir_esc=y#v=onepage&q&f=false

[104] BÄCHLE, Michael a Paul KIRCHBERG. Ruby on Rails. *IEEE Software* [online]. 2007, **24**(6), 105-108 [cit. 2022-05-12]. ISSN 0740-7459. Dostupné z: doi:10.1109/MS.2007.176

[105] DAUZON, Samuel, Aidas BENDORAITIS a Arun RAVINDRAN. *Django: Web Development with Python* [online]. Birningham - Mumbai: Packt Publishing, 2016 [cit. 2022-05-12]. ISBN 978-1-78712-138-6. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=vKjWDQAAQBAJ&oi=fnd&pg=PP1&dq=django+framework+python&ots=2nRFzznXtK&sig=2RfKG3EvMYWEh4o1qbzacusIN5k&redir_esc=y#v=onepage&q&f=false

[106] Django's release process: Release cadence. *Django: The web framework for perfectionists with deadlines.* [online]. 2005 [cit. 2022-05-12]. Dostupné z: https://docs.djangoproject.com/en/dev/internals/release-process/

[107] VINCENT, William S. *Django for beginners: build websites with Python and Django*. [USA?]: William S. Vincent, [2018-2019]. ISBN 978-1983172663.

[108] STONEBRAKER, Michael. SQL databases v. NoSQL databases. *Communications of the ACM* [online]. 2010, **53**(4), 10-11 [cit. 2022-05-12]. ISSN 0001-0782. Dostupné z: doi:10.1145/1721654.1721659

[109] VATIKA, Sharma a Dave MEENU. SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering* [online]. Vol. 8.2012, **2012**(8 - vol. 2), 20-27 [cit. 2022-05-12]. ISSN 2277 128X.

[110] AKHTAR, Zubair. Relational Database Benefits and Limitations (Advantages & Disadvantages): Relational Database Benefits. *Database Town* [online]. 2.8.2021 [cit. 2022-05-12]. Dostupné z: https://databasetown.com/relational-database-benefits-and-limitations/

[111] ACID properties of transactions. *IBM: Oficiálna dokumentácia firmy IBM* [online]. IBM Corporation, 2017, 31.1.2022 [cit. 2022-05-12]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions

[112] Ranking of the most popular relational database management systems worldwide. In: *Statista: Empowering people with data* [online]. 1.2022 [cit. 2022-05-12]. Dostupné z: https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/

[113] KONG, Chao, Ming GAO, Weining QIAN, Minqi ZHOU, Xueqing GONG, Rong ZHANG a Aoying ZHOU. ACID Encountering the CAP Theorem: Two Bank Case Studies. *2015 12th Web Information System and Application Conference (WISA)* [online]. IEEE, 2015, 2015, 235-240 [cit. 2022-05-12]. ISBN 978-1-4673-9371-3. Dostupné z: doi:10.1109/WISA.2015.63

[114] LITOIU, Marin, Purwa GAIKWAD, Mark SHTERN, Brian RAMPRASAD, Nasim BEIGI-MOHAMMADI, Saeed ZAREIAN, Marios FOKAEFS a Hamzeh KHAZAEI. How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey. *Big Data and Information Analytics* [online]. 2016, **1**(2/3), 185-216 [cit. 2022-05-12]. ISSN 2380-6966. Dostupné z: doi:10.3934/bdia.2016004

[115] *DB-Engines: Knowledge Base of Relational and NoSQL Database Management Systems* [online]. solid IT, 2022 [cit. 2022-05-12]. Dostupné z: https://db-engines.com/en/ranking

[116] PECINOVSKÝ, Rudolf. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Brno: Computer Press, 2007, 527 s. ISBN 9788025115824.

[117] MARTIN, Robert C. *Clean architecture: a craftsman's guide to software structure and design*. Boston: Prentice Hall, [2018]. Robert C. Martin series. ISBN 978-0-13-449416-6.

[118] HUNT, John. *Scala Design Patterns: Patterns for Practical Reuse and Design* [online]. University of Bristol, London: Springer International Publishing, 2013, **2013** [cit. 2022-05-14]. Dostupné z: doi:10.1007/978-3-319-02192-8

[119] FOWLER, Martin. *Patterns of enterprise application architecture*. 17th ed. Boston: Addison-Wesley, 2003. Addison-Wesley signature series. ISBN 0-321-12742-0.

[120] QURESHI, M. Rizwan Jameel a Fatima SABIR. *A comparison of model view controller and model view presenter* [online]. 2014, 8.2014 [cit. 2022-05-14]. ISSN 1013-5316. Dostupné z: https://www.researchgate.net/publication/265052131_A_comparison_of_model_view_controller_and_model_view_presenter

[121] REENSKAUG, Trygve. *The original MVC reports* [online]. Dept. of Informatics, University of Oslo, 12.2.2007, 11 [cit. 2022-05-14]. Dostupné z: https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf

[122] Model–view–controller: Diagram of interactions within one possible take on the MVC pattern. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-05-14]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/1280px-MVC-Process.svg.png

[123] LEFF, A. a J.T. RAYFIELD. Web-application development using the Model/View/Controller design pattern. *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference* [online]. IEEE Comput. Soc, 2001, 118-127 [cit. 2022-05-14]. ISBN 0-7695-1345-X. Dostupné z: doi:10.1109/EDOC.2001.950428

[124] PERES, Ricardo. *Mastering ASP.NET Core 2.0: MVC patterns, configuration, routing, deployment, and more*. Birmingham: Packt, 2017, xi, 471 s. ISBN 9781787283688.

[125] O'NEIL, Elizabeth J. Object/relational mapping 2008. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08* [online]. New York, New York, USA: ACM Press, 2008, 2008, 1351- [cit. 2022-05-14]. ISBN 9781605581026. Dostupné z: doi:10.1145/1376616.1376773

[126] VERNON, Vaughn. *Implementing Domain-Driven Design*. Addison-Wesley, 2013. ISBN 9780133039887.

[127] BUTANI, Anand. 5 essential patterns of software architecture. *RedHat* [online]. Raleign, North Carolina, United States, 16.12.2020 [cit. 2022-05-14]. Dostupné z: https://www.redhat.com/architect/5-essential-patterns-software-architecture

[128] ANDERSON, Rick, Taylor MULLEN a Dan VICAREL. Razor syntax reference for ASP.NET Core. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-6.0

[129] Architecture Styles. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 6.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/

[130] N-tier architecture style. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

[131] ADAMKÓ, Attila. *Internet Tools and Services: Lecture Notes - Ch. 4* [online]. 2014 [cit. 2022-05-14]. Dostupné z: https://gyires.inf.unideb.hu/GyBITT/08/ch04.html. Výukové materiály. University of Debrecen.

[132] NOACK, Jörg, Hamarz MEHMANECHE, Homayoun MEHMANECHE a Andreas ZENDLER. *Architectural Patterns for Web Applications* [online]. Bonn-Germany, Munich-Germany: Informatikzentrum der Sparkassenorganisation (SIZ) Bonn/Germany, MGM GmbH Munich/Germany, 1-10 [cit. 2022-05-14]. Dostupné z: doi:10.1.1.39.9641

[133] GRACA, Herberto. Layered Architecture. In: *HerbertoGraca* [online]. 3.8.2017 [cit. 2022-05-14]. Dostupné z: https://herbertograca.files.wordpress.com/2017/07/2010s-layered-architecture.png

[134] PIETSCHMANN, Stefan, Tobias NESTLER a Florian DANIEL. *Application composition at the presentation layer* [online]. New York, New York, USA: ACM Press, 2010, 2010, 461- [cit. 2022-05-14]. ISBN 9781450304214. Dostupné z: doi:10.1145/1967486.1967558

[135] Three-Tier Architecture: The three tiers in detail. *IBM: Oficiálna dokumentácia firmy IBM* [online]. [cit. 2022-05-14]. Dostupné z: https://www.ibm.com/cz-en/cloud/learn/three-tier-architecture

[136] TORRES, Alexandre, Renata GALANTE, Marcelo S. PIMENTA a Alexandre Jonatan B. MARTINS. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology* [online]. 2017, **82**, 1-18 [cit. 2022-05-14]. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2016.09.009

[137] SINGH, Anil. Understanding about Object Relation Mapper(ORM): What is ORM? Why we use ORM?. In: *Code Sample: Learn Simple* [online]. 2017, 26.1.2016 [cit. 2022-05-14]. Dostupné z: https://1.bp.blogspot.com/-BoLqS7MxDgA/VqhVvforIcI/AAAAAAAALi0/6x6VfY8qcLM/s400/ORM%2BObject%2BRelation%2BMapper.png

[138] CHEN, Tse-Hsun, Weiyi SHANG, Zhen Ming JIANG, Ahmed E. HASSAN, Mohamed NASSER a Parminder FLORA. Detecting performance anti-patterns for applications developed using object-relational mapping. *Proceedings of the 36th International Conference on Software Engineering* [online]. New York, NY, USA: ACM, 2014, 2014-05-31, 1001-1012 [cit. 2022-05-14]. ISBN 9781450327565. Dostupné z: doi:10.1145/2568225.2568259

[139] Design the infrastructure persistence layer: The Repository pattern. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 13.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[140] SINGH, Jasminder. Unit of Work in Repository Pattern. *C# Corner* [online]. 2022, 30.11.2018 [cit. 2022-05-14]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/#:~:text=Unit%20of%20Work%20is%20the,update%2Fdelete%20and%20so%20on.

[141] ISRAELI, Elad. DIP, IoC, DI — Know Them Better. *Medium: Stay curious* [online]. 25.1.2021 [cit. 2022-05-14]. Dostupné z: https://medium.com/nmc-techblog/dip-ioc-di-know-them-better-abed5b57fd20

[142] RAZINA, Ekaterina a David JANZEN. *Effects of Dependency Injection on Maintainability* [online]. Cambridge MA, USA, 2007 [cit. 2022-05-14]. ISBN 978-0-88986-706-2. Dostupné z: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1035&context=csse_fac

[143] Inversion of Control Tutorial: Introduction. In: *Tutorial AZ* [online]. [cit. 2022-05-14]. Dostupné z: https://tutorialaz.com/storage/images/principles-and-patterns.png

[144] PALKAR, Rikam. What is Dependency Injection and What Are Its Types? How to implement DI?. *C# Corner* [online]. 24.12.2019 [cit. 2022-05-14]. Dostupné z: https://www.c-sharpcorner.com/blogs/what-is-dependency-injection-and-what-are-its-types2

[145] SANDERSON, Steve. Blazor: a technical introduction: Deeper technical details about Blazor. *Steve Sanderson's Blog* [online]. 6.2.2018 [cit. 2022-05-14]. Dostupné z: https://blog.stevensanderson.com/2018/02/06/blazor-intro/

[146] PRICE, Mark J. *C# 9 and .NET 5 - modern cross-platform development: build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code*. Fifth edition. Birmingham: Packt, 2020. ISBN 978-1-80056-810-5.

[147] ASP.NET Core Blazor documentation. *Microsoft Docs* [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-4]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor

[148] GRACE, David. Why You Should Use Blazor over JavaScript Frameworks to Build Your Single-Page Application. *Telerik: Modern UI made easy* [online]. 24.3.2020 [cit. 2022-05-14]. Dostupné z: https://www.telerik.com/blogs/why-you-should-use-blazor-over-javascript-frameworks-to-build-your-single-page-application

[149] ASP.NET Core Blazor. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 17.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0

[150] .NET is Free: .NET is an open-source developer platform with no licensing costs and free development tools for Linux, macOS, and Windows. *Microsoft .NET: Free. Cross-platform. Open source. A developer platform for building all your apps.* [online]. [cit. 2022-05-14]. Dostupné z: https://dotnet.microsoft.com/en-us/platform/free

[151] G., Piotr. What is the difference between "razor" and "cshtml" files?. In: *Stack Overflow* [online]. 23.5.2019 [cit. 2022-05-14]. Dostupné z: https://stackoverflow.com/questions/55590605/what-is-the-difference-between-razor-and-cshtml-files#:~:text=I%20might%20be%20wrong%20but,fit%20into%20a%20Razor%20page.

[152] HEJLSBERG, Anders, Mads TORGERSEN, Scott WILTAMUTH a Peter GOLDE. *The C# Programming Language: Microsoft Windows Development Series*. 3rd ed. Pearson Education, 2008. ISBN 978-0-32159-225-5.

[153] OOAD - Object Oriented Paradigm: Object-Oriented Programming. *Tutorials Point: Simply Easy Learning* [online]. [cit. 2022-05-14]. Dostupné z: https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm

[154] ALBAHARI, Joseph. *C# 9.0 in a Nutshell: The Definitive Reference*. O'Reilly Media, 2021. ISBN 978-1-09810-093-3.

[155] ASP.NET Core Blazor hosting models: Blazor Server. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 17.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0

[156] Overview of ASP.NET Core SignalR: What is SignalR?. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-6.0

[157] BERES, Jason. Blazor Server vs. Blazor WebAssembly: Just the Facts: Blazor Server. In: *Infragistics* [online]. 23.2.2021 [cit. 2022-05-14]. Dostupné z: https://www.infragistics.com/community/resized-image/__size/2080x0/__key/communityserver-blogs-components-weblogfiles/00-00-00-03-12/blazor_2D00_server_2D00_signalr_2D00_example.png

[158] ASP.NET Core Blazor state management. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/state-management?view=aspnetcore-6.0&pivots=server

[159] SATAPATHI, Ashirwad. All You Need To Know About Blazor: Hosting models available for Blazor. *C# Corner* [online]. 28.9.2020 [cit. 2022-05-14]. Dostupné z: https://www.c-sharpcorner.com/article/all-you-need-to-know-about-blazor/

[160] Choose Between Traditional Web Apps and Single Page Apps (SPAs): Blazor. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 14.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

[161] SPA (Single-page application). *MDN Web Docs: Resources for Developers, by Developers* [online]. Mozilla Foundation, 8.10.2021 [cit. 2022-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/SPA

[162] *WebAssembly: Oficiálna stránka formátu WebAssembly* [online]. 2017 [cit. 2022-05-14]. Dostupné z: https://webassembly.org/

[163] BERES, Jason. Blazor Server vs. Blazor WebAssembly: Just the Facts: Blazor WebAssembly. *Infragistics* [online]. 23.2.2021 [cit. 2022-05-14]. Dostupné z: https://www.infragistics.com/community/resized-image/__size/1040x0/__key/communityserver-blogs-components-weblogfiles/00-00-00-03-12/web_2D00_assembly_2D00_diagram.png

[164] Secure ASP.NET Core Blazor WebAssembly. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 29.4.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/security/webassembly/?view=aspnetcore-6.0

[165] JUNEJA, Majmeet. Difference between Authentication and Authorization. *Geeks for Geeks* [online]. Noida, Uttar Pradesh, 7.7.2020 [cit. 2022-05-14]. Dostupné z: https://www.geeksforgeeks.org/difference-between-authentication-and-authorization/

[166] Overview of ASP.NET Core authentication. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0

[167] MEIER, J.D., Alex MACKMAN, Michael DUNNER a Srinath VASIREDDY. *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication* [online]. In: . Microsoft Corporation, 2002, 2002 [cit. 2022-05-14]. Dostupné z: https://cgtweb1.tech.purdue.edu/courses/cgt456/Private/Notes/BuildingSecureASP_NETApplications_v2.pdf

[168] Introduction to authorization in ASP.NET Core: Authorization Types. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-6.0

[169] Role-based authorization in ASP.NET Core. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-6.0

[170] Policy-based authorization in ASP.NET Core. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 26.3.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies?view=aspnetcore-6.0

[171] Building an identity server from scratch: Talking about authentication and authorization in asp.net core. *Develop Paper* [online]. 2021, 22.7.2021 [cit. 2022-05-14]. Dostupné z: https://developpaper.com/building-an-identity-server-from-scratch-talking-about-authentication-and-authorization-in-asp-net-core/

[172] Authorization in Blazor: Authorization in ASP.NET Core. *Pragim Technologies* [online]. 2020 [cit. 2022-05-14]. Dostupné z: https://www.pragimtech.com/blog/blazor/authorization-in-blazor/

[173] ASP.NET Core Blazor authentication and authorization. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 5.5.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/security/?view=aspnetcore-6.0

[174] ANDERSON, Rick. Introduction to Identity on ASP.NET Core. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 9.5.2022 [cit. 2022-05-14]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio

[175] ASP.NET Core: Identity. *GitHub: Where the world builds software* [online]. [cit. 2022-05-14]. Dostupné z: https://github.com/dotnet/aspnetcore#:~:text=ASP.NET%20Core%20is%20an,and%20open%2Dsource%20application%20runtime.

[176] MCCORMICK, Mike. Waterfall vs. Agile Methodology. *MPCS: Project Management Resources* [online]. MPCS, 9.8.2012 [cit. 2022-05-14]. Dostupné z: http://www.mccormickpcs.com/images/Waterfall_vs_Agile_Methodology.pdf

[177] JEREMIAH, John. Survey: Is agile the new norm?. *TechBeacon* [online]. 2022 [cit. 2022-05-14]. Dostupné z: https://techbeacon.com/app-dev-testing/survey-agile-new-norm,

[178] ALAM, Ayaz. Importance of Software Design. *Medium: Stay curious* [online]. 27.11.2019 [cit. 2022-05-14]. Dostupné z: https://medium.com/swlh/importance-of-software-design-7ffea48ede17

[179] LYNCH, Allison. What is UML: Unified Modeling Language. *EDrawSoft: Unlock Diagram Possibilities* [online]. 15.2.2022 [cit. 2022-05-14]. Dostupné z: https://www.edrawsoft.com/what-is-uml-diagram.html?gclid=CjwKCAiA4KaRBhBdEiwAZi1zzp53cm13RezbUTDiaD4LsKU6yAQxG1THalooF9sIb1Wh5yJ1otVjaxoCghMQAvD_BwE

[180] *The Unified Modeling Language* [online]. [cit. 2022-05-14]. Dostupné z: https://www.uml-diagrams.org/

[181] UML 2.5 Diagrams Overview: Classification of UML 2.5 Diagrams. *The Unified Modeling Language* [online]. 2009 [cit. 2022-05-14]. Dostupné z: https://www.uml-diagrams.org/uml-25-diagrams.html

[182] SPARX, Geoffrey. Enterprise Architect User Guide. *Sparx Systems: Enterprise Architect: Make Your Vision a Reality* [online]. 11.2014 [cit. 2022-05-14]. Dostupné z: https://sparxsystems.com/bin/EAUserGuide.pdf

[183] Products: Instant access - get started in minutes. *Sparx Systems: Enterprise Architect: Make Your Vision a Reality* [online]. [cit. 2022-05-14]. Dostupné z: https://sparxsystems.com/products/ea/shop/

[184] BRACEY, Kezz. What is Figma?. *EnvatoTuts+* [online]. Envato Pty, 26.11.2018 [cit. 2022-05-14]. Dostupné z: https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272

[185] Figma: Education Plan. *Figma* [online]. [cit. 2022-05-14]. Dostupné z: https://www.figma.com/education/apply

[186] MAGUIRE, Martin a Nigel BEVAN. User Requirements Analysis. *Usability* [online]. Boston, MA: Springer US, 2002, 2002, 133-148 [cit. 2022-05-15]. IFIP Advances in Information and Communication Technology. ISBN 978-1-4757-6910-4. Dostupné z: doi:10.1007/978-0-387-35610-5_9

[187] GRADY, Jeffrey O. *System Requirements Analysis*. Burlington, MA, USA: Elsevier, 2010. ISBN 978-0-12-088514-5.

[188] Functional vs Non Functional Requirements. *Geeks for Geeks* [online]. Noida, Uttar Pradesh, 29.4.2020 [cit. 2022-05-15]. Dostupné z: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/#_=_

[189] UML Use Case. *The Unified Modeling Language* [online]. 2009 [cit. 2022-05-14]. Dostupné z: https://www.uml-diagrams.org/use-case.html

[190] Use-case diagrams. *IBM: Oficiálna dokumentácia firmy IBM* [online]. 3.4.2021 [cit. 2022-05-15]. Dostupné z: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case

[191] FAUZAN, Reza, Daniel SIAHAAN, Siti ROCHIMAH a Evi TRIANDINI. *Use Case Diagram Similarity Measurement: A New Approach* [online]. IEEE, 2019, 2019, 3-7 [cit. 2022-05-15]. ISBN 978-1-7281-2133-8. Dostupné z: doi:10.1109/ICTS.2019.8850978

[192] UML Use Case: Relationships Between Use Cases. *The Unified Modeling Language* [online]. 2009 [cit. 2022-05-14]. Dostupné z: https://www.uml-diagrams.org/use-case.html

[193] UML Class Diagram Tutorial. *Visual Paradigm* [online]. [cit. 2022-05-15]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

[194] UML Class and Object Diagrams Overview: Domain Model Diagram. *The Unified Modeling Language* [online]. [cit. 2022-05-15]. Dostupné z: https://www.uml-diagrams.org/class-diagrams-overview.html#domain-model-diagram

[195] REZA, Jurgen a Reinaldo LUCKMAN. Should I put id attributes in UML class diagrams?. In: *Stack Overflow* [online]. 26.2.2013 [cit. 2022-05-15]. Dostupné z: https://stackoverflow.com/questions/15080090/should-i-put-id-attributes-in-uml-class-diagrams#:~:text=So%20if%20you%20are%20using,yes%2C%20include%20the%20id%20attribute.

[196] GULIZZONI, Peldi. What Are Wireframes?. *Balsamiq: Life's too short for bad software!* [online]. 2008 [cit. 2022-05-15]. Dostupné z: https://balsamiq.com/learn/articles/what-are-wireframes/

[197] VELARDE, Orana. What is a Wireframe?: Guide With Types, Benefits & Tips (2022). *Visme* [online]. Maryland: DBA Visme, 12.8.2021 [cit. 2022-05-15]. Dostupné z: https://visme.co/blog/what-is-a-wireframe/

[198] SINO V10P Vertical Machining Center. *Lead: Machine Tools* [online]. [cit. 2022-05-16]. Dostupné z: https://www.leadmachinetools.co.za/sites/default/files/styles/updated_product_grid/public/2021-01/V8P_0.jpg?itok=jzlTS59H

[199] KMH Auto Pallet HMC Series. In: *Kent CNC* [online]. Tustin, CA, USA [cit. 2022-05-16]. Dostupné z: https://kentcnc.com/wp-content/uploads/2018/06/Kent-CNC-HMC-with-APC-3-600x572.jpg

[200] Pásová pila na kov Holzmann BS115. In: *Holzmann-Zipper* [online]. 2012 [cit. 2022-05-16]. Dostupné z: https://www.holzmann-zipper.cz/img_detail/BS115.jpg

[201] Soustruh na kov Holzmann ED400FDDIG. In: *Holzmann-Zipper* [online]. 2012 [cit. 2022-05-16]. Dostupné z: https://www.holzmann-zipper.cz/img_detail/ED400FDDIG.jpg

[202] FAQ. *Blazorise Docs: Oficiálna dokumentácia technológie Blazorise* [online]. [cit. 2022-05-16]. Dostupné z: https://blazorise.com/docs/faq

[203] FREEMAN, Adam. *Pro Entity Framework Core 2 for ASP.NET Core MVC*. [Berkeley, CA]: Apress, 2018, 1 online zdroj. ISBN 9781484234358. Dostupné také z: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&an=1794405&scope=site

[204] *Swagger - Supported by SMARTBEAR: API Development for Everyone* [online]. SmartBear Software, 2021 [cit. 2022-05-16]. Dostupné z: https://swagger.io/

[205] WILLIAMS, Lawrence. 10 Most Common Web Security Vulnerabilities. *Guru99: Guru99 is totally new kind of learning experience.* [online]. Guru99, 5.3.2022 [cit. 2022-05-16]. Dostupné z: https://www.guru99.com/web-security-vulnerabilities.html

[206] HASAN, Fiyaz, Rick ANDERSON a Steve SMITH. Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 19.4.2022 [cit. 2022-05-16]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0

[207] Blazor: Json Web Token (JWT) Authentication Example - Simple: A simple example of adding JWT Bearer authentication to Blazor WebAssembly (WASM). *PROWARE Tech* [online]. [cit. 2022-05-16]. Dostupné z: https://www.prowaretech.com/articles/current/blazor/wasm/jwt-authentication-simple

[208] Introduction. *Postman: Oficiálna dokumentácia nástroja Postman* [online]. 22.4.2022 [cit. 2022-05-16]. Dostupné z: https://learning.postman.com/docs/getting-started/introduction/

[209] SAMPATH, -. Entity Framework + SQL injection. In: *Stack Overflow* [online]. Stack Exchange, 7.9.2016 [cit. 2022-05-16]. Dostupné z: https://stackoverflow.com/questions/39359463/entity-framework-sql-injection

[210] Security Considerations (Entity Framework): General Security Considerations. *Microsoft Docs: Oficiálna dokumentácia firmy Microsoft* [online]. 6.11.2021 [cit. 2022-05-16]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations?redirectedfrom=MSDN

[211] INFLAÇÃO NA INDÚSTRIA BATE RECORDE HISTÓRICO. In: *Digital Money Informe* [online]. 27.8.2021 [cit. 2022-05-16]. Dostupné z: https://i0.wp.com/www.digitalmoneyinforme.com.br/wp-content/uploads/2021/08/operario-trabalha-em-maquina-industrial-freepik-2021-digital-money-informe.jpg?w=936&ssl=1

[212] FELIPE, Mateus Campos. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 13.11.2019 [cit. 2022-05-16]. Dostupné z: https://unsplash.com/photos/uLDwvGvWrs0

[213] FELIPE, Mateus Campos. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 13.11.2019 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1573607217032-18299406d100?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=830&q=80

[214] AKYURT, Engin. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 13.2.2020 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1581629774175-42f704962488?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=774&q=80

[215] CHAN, Cody. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 16.7.2019 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1563235453-a57d94b5a552?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=774&q=80

[216] MIZAN, Kazi. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 23.3.2020 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1584940120743-8981ca35b012?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=774&q=80

[217] HOLOSCHCHUK, Roman. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 30.5.2020 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1590873803005-539ede4d828a?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=774&q=80

[218] SMYTH, Daniel. In: *Unsplash: The internet's source of freely-usable images. Powered by creators everywhere.* [online]. 25.9.2020 [cit. 2022-05-16]. Dostupné z: https://images.unsplash.com/photo-1601045378965-58f245425f7f?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1931&q=80

[219] Handshake: ID: 132708305. In: *Shutterstock* [online]. [cit. 2022-05-16]. Dostupné z: https://www.shutterstock.com/cs/image-photo/handshake-hand-holding-on-black-background-132708305

[220] Administrative Worker: ID: 750073606. In: *Shutterstock* [online]. [cit. 2022-05-16]. Dostupné z: https://www.shutterstock.com/cs/image-photo/businessman-working-on-desk-office-using-750073606

[221] SELLERS, Samantha. Settings Wallpaper. In: *CuteWallpaper: Search For Free Wallpapers To Download* [online]. [cit. 2022-05-16]. Dostupné z: https://cutewallpaper.org/21/settings-wallpaper/1920x1080-Gear-Ubuntu-desktop-PC-and-Mac-wallpaper.jpg

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

IPC Inter-Process Communication

DoS Denial of Service

DDoS Distributed Denial of Service

GUI Graphical User Interface

WWW World Wide Web

HTML Hyper Text Markup Language

CSS Cascading Style Sheets

HTTP Hyper Text Transfer Protocol

HTTPS Hyper Text Transfer Protocol Secured

IETF Internet Engineering Task Force

URL Uniform Resource Locator

MVC Model – View – Controller

OOP Object – Oriented Programming

API Application Programming Interface

JSON JavaScript Object Notation

XML Extensible Markup Language

SSL Secure Sockets Layer

TLS Transport Layer Security

XAML Extensible Application Markup Language

DOM Document Object Model

SGML Standard Generalized Markup Language

KML Keyhole Markup Language

W3C World Wide Web Consortium

HSL Hue, Saturation, Lightness

RGBA	Red, Green, Blue, Alpha
CLI	Command Line Interface
SASS	Syntactically Awesome Style Sheets
MEAN	MongoDB, Express.js, Angular, Node.js
MV*	Model – View – Whatever
MVVM	Model – View – View Model
UI	User Interface
CMS	Content Management System
WORA	Write Once Run Anywhere
JVM	Java Virtual Machine
ORM	Object – Relational Mapping
SQL	Structured Query Language
CRUD	Create, Read, Update, Delete
CoC	Convention over Configuration
DRY	Don´t Repeat Yourself
RoR	Ruby on Rails
BSD	Berkeley Software Distribution
MVT	Model – View – Template
NoSQL	Not only SQL (Structured Query Language)
ACID	Atomicity, Consistency, Isolation, Durability
CAP	Consistency, Availability, Partition Tolerance
DTO	Data Transfer Object
DSL	Domain Specific Language
OSI	Open System Interconnection model
ISO	International Organization for Standardization
WPF	Windows Presentation Foundation

IoC	Inversion of Control
DI	Dependency Injection
LINQ	Language Integrated Query
CDN	Content Delivery Network
SPA	Single Page Application
SEO	Search Engine Optimization
UML	Unified Modeling Language
UC	Use Case
SysML	Systems Modeling Language
BPMN	Business Process Model and Notation
ERD	Entity Relationship Diagrams
MVP	Minimal Viable Product

ZOZNAM OBRÁZKOV

ZOZNAM ZDROJOVÝCH KÓDOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Príloha P I: CD s diplomovou prácou vrátane zdrojových kódov a pridružených softwarových produktov

PRÍLOHA P I: CD

Priložené CD obsahuje:

- Diplomovú prácu vo formáte .docx: DP_JozefKovac_A20132.docx
- Diplomovú prácu vo formáte .pdf: DP_JozefKovac_A20132.pdf
- EA Projekt - diagramy vo formáte .eapx: DP_EAP_JozefKovac_A20132.eapx
- Export – Figma vo formáte .pdf: DP_FIG_JozefKovac_A20132.pdf
- Komprimované zdrojové kódy aplikácie: DP_PRACT_JozefKovac_A20132.zip