

Vývoj aplikace zaměřené na výuku mediální gramotnosti

Tereza Knápková



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Tereza Knápková
Osobní číslo: A22015
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Vývoj aplikace zaměřené na výuku mediální gramotnosti
Téma práce anglicky: Development of an Application Focused on Media Literacy Education

Zásady pro vypracování

- Provedte analýzu prototypů vytvořených na ateliéru Digitální design FMK a jeden vyberte.
- Na základě literární rešerše platforem pro vývoj mobilních, webových a hybridních aplikací zvolte vhodnou cílovou platformu.
- V praktické části navrhnete a implementujete zvolený prototyp pro vybranou platformu a popíšete kroky implementace.
- Ověřte bezpečnost řešení a proveďte možnosti jeho publikování.

Seznam doporučené literatury:

- **Martin, Robert C., Ottinger, Timothy R., Langr, Jeff, Schuchert, Brett L., Grenning, James W. et al.** *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin series. Upper Saddle River: Prentice Hall, 2009. ISBN 9780132350884.
- **Shneiderman, Ben.** *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6. vydání. Pearson, 2016. ISBN 978-0134380384.
- **Pressman, Roger S., Maxim, Bruce R.** *Software Engineering: A Practitioner's Approach*. 8. vydání. McGraw-Hill International Edition. New York: McGraw-Hill Education, 2015. ISBN 9781259253157.
- **Krug, Steve.** *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3. vydání. Berkeley, California: New Riders, 2014. ISBN 9780321965516.
- **Goldberg, Josh.** *Learning TypeScript: Enhance Your Web Development Skills Using Type-Safe JavaScript*. O'Reilly Media, 2022. ISBN 978-1633437319.
- **Carey, Benedict.** *Jak se učíme: překvapivá pravda o tom, kdy, kde a jak se učíme*. 1. vydání. Brno: Bizbooks, 2015. ISBN 9788026503491

Vedoucí bakalářské práce:

Ing. Štěpán Dlabaja, Ph.D.

Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **27. října 2024**

Termín odevzdání bakalářské práce: **2. června 2025**



doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou 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;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské 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 bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné;
- při tvorbě této bakalářské práce jsem použila nástroj generativního modelu AI [ChatGPT; <https://chatgpt.com>] za účelem zlepšení textu, kontroly pravopisu a jazykové korektury. Při použití tohoto nástroje jsem provedla kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace zaměřené na výuku mediální gramotnosti s využitím interaktivních herních prvků. Cílem bylo vytvořit platformu pro trénink kritického myšlení při práci s informačními zdroji, kterou tvoří front-endová hra „MarsGate“ postavená na frameworku Svelte a backendová služba v ASP.NET Web API. Příběh hry byl vytvořen ve spolupráci se spolkem Zvol si info

Klíčová slova: mediální gramotnost, webová aplikace, edukační hra, Svelte, ASP.NET Web API, Entity Framework Core

ABSTRACT

This bachelor's thesis addresses the design and implementation of a web application focused on teaching media literacy through interactive game elements. The goal was to create a platform for training critical thinking when working with information sources, comprising the frontend game “MarsGate” built on the Svelte framework and a backend service implemented in ASP.NET Web API. The game's story was developed in collaboration with the Zvol si info association.

Keywords: media literacy, educational game, web application, Svelte, ASP.NET Web API, Entity Framework Core

Ráda bych poděkovala vedoucímu práce panu Ing. Štěpánu Dlabajovi, Ph.D. za odborné vedení, cenné rady a čas, který mi věnoval. Dále děkuji paní Bc. Nicole Pelové za spolupráci při tvorbě obsahu hry a celému týmu spolku Zvol si info za podporu při realizaci tohoto projektu.

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

Whatever you are not changing, you are choosing.

— Laurie Buchanan

OBSAH

ÚVOD	13
I TEORETICKÁ ČÁST	14
1 VÝBĚR PROTOTYPU	15
1.1 PROJEKT ZVOL SI INFO	15
1.1.1 Ranné správy	15
1.1.2 Conspirator	16
1.1.3 Materiály pro výuku mediální gramotnosti	17
1.1.4 Deepfake	17
1.1.5 Perspekt	18
1.2 ZVOLENÝ PROTOTYP	19
2 VÝBĚR TECHNOLOGIE	20
2.1 WEBOVÉ TECHNOLOGIE	21
2.1.1 HTML	21
2.1.2 CSS	21
2.1.3 JavaScript	22
2.1.4 TypeScript	22
2.2 KOMPILAČNÍ METODY	23
2.2.1 Virtuální DOM	23
2.2.2 Přímá manipulace s DOM	24
2.3 FRONTEND	25
2.3.1 React	26
2.3.2 Angular	27
2.3.3 Vue.js	29
2.3.4 Svelte	29
2.4 BACKEND	31
2.4.1 Architektura systému	32
2.4.2 Model-View-Controller	32
2.4.3 Application Programmable Interface (API)	32
2.4.4 Django	33

2.4.5	Laravel.....	34
2.4.6	ASP.NET Core.....	35
II	PRAKTICKÁ ČÁST	37
3	NÁVRH APLIKACE	38
3.1	SPECIFIKACE POŽADAVKŮ SE SPOLKEM ZVOL SI INFO.....	38
3.2	REGISTRACE A PŘIHLAŠOVÁNÍ	38
3.3	PRŮBĚH HRY	38
3.3.1	Průvodní příběh.....	39
3.3.2	Úvodní obrazovka hry.....	39
3.3.3	Level 1 - První zpráva.....	40
3.3.4	Level 2 - Kdo a proč?	42
3.3.5	Level 3 - Důkazy pod lupou	44
3.3.6	Level 4 - Proč tomu lidé věří	45
3.3.7	Level 5 - Závěrečný kvíz	48
3.3.8	Zpětná vazba	50
3.3.9	Fun fact.....	51
4	IMPLEMENTACE BACKEND	53
4.1	VYBRANÁ PLATFORMA	53
4.2	ARCHITEKTURA APLIKACE	53
4.2.1	Využití principů v aplikaci.....	54
4.3	IMPLEMENTACE A NÁVRH DATABÁZE.....	55
4.4	KOMUNIKACE MEZI VRSTVAMI.....	57
4.5	MANUÁLNÍ OVĚŘOVÁNÍ FUNKČNOSTI.....	59
5	IMPLEMENTACE FRONTEND	60
5.1	VYBRANÁ PLATFORMA - SVELTE	60
5.2	VYTVOŘENÍ PROJEKTU	60
5.3	STRUKTURA SLOŽEK	61
5.3.1	Detailní popis složky Adres routes/	62
5.3.2	Práce se static/	62
5.3.3	Komponenty v src/lib/	62

5.4	ÚVODNÍ STRANA.....	63
5.5	HRA MARS GATE.....	64
5.6	KOMUNIKACE S BACKENDEM.....	65
5.6.1	Definice datových rozhraní	66
5.7	PERZISTENTNÍ PAMĚŤ	66
5.8	ADMINISTRATIVNÍ ČÁST	67
5.9	IMPLEMENTACE KONKRÉTNÍ HRY	68
5.9.1	Drag-and-drop - kognitivní zkreslení	68
6	BEZPEČNOST ŘEŠENÍ	72
6.1	ZABEZPEČENÍ DAT	72
6.1.1	JWT Token	72
6.1.2	Serverová konfigurace	72
6.1.3	Klientská konfigurace	73
6.2	CORS	73
6.3	POLITIKA HESLA A JEJÍ OVĚŘOVÁNÍ	74
7	MOŽNOSTI PUBLIKACE.....	76
7.1	BACKEND (.NET WEB API)	76
7.1.1	Platforma jako služba (PaaS)	76
7.1.2	Kontejnerizace	76
7.1.3	Vybrané řešení.....	77
7.2	SVELTEKIT FRONTEND	77
7.2.1	Režimy vykreslování frontendové aplikace	77
7.2.2	Vybrané řešení.....	78
8	BUDOUCÍ VÝVOJ	79
	ZÁVĚR.....	80
	SEZNAM POUŽITÉ LITERATURY	81
	SEZNAM OBRÁZKŮ	86
	SEZNAM TABULEK	88
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	90

SEZNAM PŘÍLOH.....	91
---------------------------	-----------

ÚVOD

V současné digitální éře je stále náročnější rozlišit ověřené informace od zavádějících či zcela nepravdivých sdělení. Ačkoli hlavní platformy sociálních sítí postupně implementují mechanismy detekce a označování potenciálně dezinformačního obsahu, dosud neexistuje univerzální nástroj, který by falešným zprávám zcela zabránil. Nicméně existují osvědčené metody a heuristiky, které uživatelům pomáhají v každodenním používání internetu kriticky ověřovat důvěryhodnost zdrojů.

Téma dezinformací, fake news a zodpovědnosti v digitálním světě je hlavním zaměřením neziskové organizace Zvol si info, s níž byla tato práce realizována. Ve spolupráci s touto organizací vznikla interaktivní příběhová webová hra, která v rámci čtyř úrovní systematicky provádí hráče principy kritického myšlení a ověřování online obsahu.

Teoretická část práce se nejprve věnuje analýze existujících prototypů interaktivních aplikací vyvinutých studenty Ateliéru digitálního designu Fakulty multimediálních komunikací Univerzity Tomáše Bati ve Zlíně. Následně přináší přehled klíčových technologií webového vývoje – standardů HTML5, CSS3 a jazyka JavaScript – a představuje moderní frameworky a nástroje využívané pro tvorbu frontendových a backendových komponent webových aplikací.

Praktická část popisuje návrh architektury navržené aplikace, implementaci uživatelského rozhraní ve frameworku Svelte a realizaci serverové logiky na platformě ASP.NET Core. Závěrečné kapitoly jsou věnovány bezpečnosti řešení, možnostem jeho nasazení v provozu a návrhům dalšího rozvoje.

I TEORETICKÁ ČÁST

1 VÝBĚR PROTOTYPU

Tato kapitola obsahuje analýzu prototypů navržených studenty Ateliéru Digitální design Fakulty multimediálních komunikací na Univerzitě Tomáše Bati ve Zlíně během akademického roku 2021/2022. Celkem bylo vytvořeno 14 prototypů, z nichž byl proveden užší výběr. Následující text představuje a popisuje vybrané prototypy.

1.1 Projekt Zvol si info

Zvol si info je spolek Masarykovy univerzity v Brně založený v roce 2016, v jehož řadách jsou zastoupeni zejména studentky a studenti žurnalistiky, pedagogiky, politologie, ale mají i členy z řad studentů informatiky.

Záměrem a posláním spolku je bojovat s dezinformacemi a jinými nástrahami digitálního světa. Tématem jejich přednášek a workshopů je zejména naučit lidi nakládat s informacemi a odhalovat manipulace, které jsou na internetu čím dál častější. [1]

Tabulka 1.1 Přehled prototypů

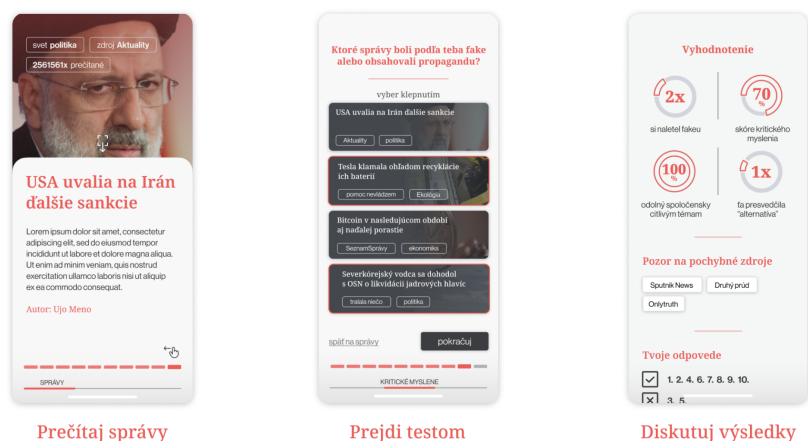
Název prototypu	Platforma	Popis
Ranné správy	Mobilní aplikace	Analýza zpráv
Conspirator	Webová aplikace	Rozpoznání konspiračních teorií
Materiály pro výuku mediální gramotnosti	Multiplatformní	Správa výukových materiálů
Deepfake	Webová aplikace	Rozpoznávání deepfake obsahu
Perspekt	Webová aplikace	Práce se zdroji

(Zdroj: vlastní)

1.1.1 Ranné správy

Cílem této mobilní aplikace je zvýšit povědomí studentů o důležitosti výběru a ověřování zdrojů informací. [2]

Studentům je vždy představena zpráva, u které je viditelný její zdroj, zařazení do kategorie a počet přečtení. Na základě obsahu zprávy a pomocných indicií má student za úkol určit, zda se jedná o fake news, nebo pravdivou zprávu. Po dokončení celého kvízu jsou zobrazeny výsledky, statistiky a správné odpovědi spolu s vysvětlením.



Obrázek 1.1 Prototyp Ranné správy

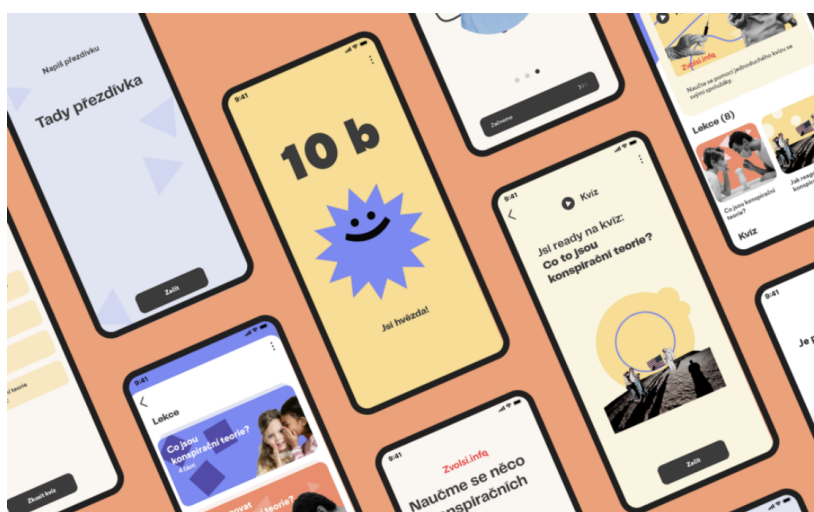
[2]

1.1.2 Conspirator

Prototyp *Conspirator* je interaktivní webová aplikace zaměřená na problematiku konspiračních teorií – jejich rozpoznávání a vyvracení. [3]

Aplikace je rozdělena na sekci s výukovými lekcemi a na část s kvízy. Kvízy fungují podobně jako v aplikaci *Kahoot*, kde je otázka promítána na tabuli a studenti odpovídají individuálně na svých zařízeních.

Učitel má možnost vytvářet vlastní kvízy nebo využívat již připravené materiály.



Obrázek 1.2 Prototyp Conspirator

[2]

1.1.3 Materiály pro výuku mediální gramotnosti

Tento prototyp představuje multiplatformní rozhraní pro správu výukových materiálů a podporu komunikace mezi studenty a učiteli. [4]

Desktopové rozhraní je určeno pro pedagogy, kteří si mohou prohlížet výukové materiály přímo z platformy *ZvolSi.Info*, nahrávat vlastní obsah a kontrolovat odevzdanou práci studentů.

Studenti využívají mobilní aplikaci, která jim umožňuje přístup k výukovým materiálům rozděleným do kategorií (např. zdroje, fake news). Učitelé mohou zadávat interaktivní úkoly, jako je vyplnění kvízů, dohledávání článků nebo zdrojů z internetu, či přidání fotografie novinového výstřižku. Úkoly studenti odevzdávají přímo skrze aplikaci. Přihlašování je realizováno prostřednictvím školního e-mailu, což eliminuje potřebu registrace.



Obrázek 1.3 Prototyp Materiály pro výuku mediální gramotnosti

[4]

1.1.4 Deepfake

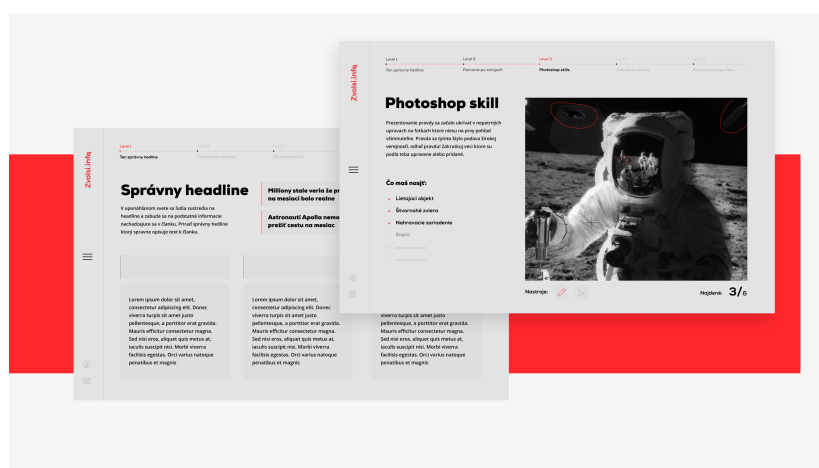
Tato webová aplikace se zaměřuje na aktuální problematiku rozpoznávání deepfake obsahu, kdy je čím dál obtížnější odlišit reálné fotografie a videa od manipulovaných. [5]

Aplikace nabízí dva typy kvízů – zaměřené na konspirační teorie a filmový deepfake. Obsah kvízů, jako jsou videa, fotografie, příběhy a otázky, se liší podle zvoleného tématu. Každý kvíz je strukturován jako interaktivní příběh, kde uživatel řeší různé úkoly a odpovídá

na otázky. Tyto úkoly testují schopnosti jako čtení s porozuměním, ověřování zdrojů, rozpoznávání fotomontáží a manipulovaných videí, nebo pochopení nástrojů, jako je Photoshop.

Uživatelé dostávají průběžnou zpětnou vazbu o svém pokroku, včetně tipů na zlepšení. Na závěr kvízu je zobrazeno celkové vyhodnocení s výsledky a shrnutím příběhu.

Aplikace zvyšuje mediální gramotnost a pomáhá uživatelům kriticky přemýšlet o informacích, se kterými se setkávají.



Obrázek 1.4 Prototyp Deepfake

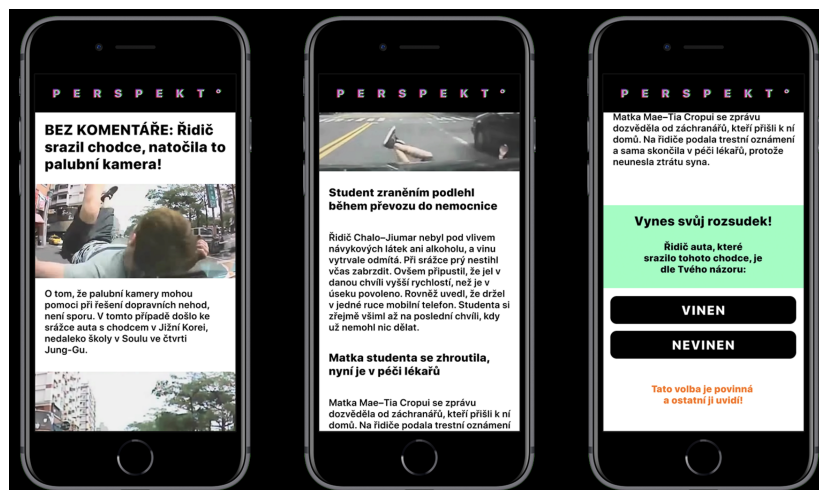
[5]

1.1.5 Perspekt

Prototyp *Perspekt* slouží jako informační cvičení, které studentům ukazuje, jak snadno mohou dojít k chybným závěrům při práci s neúplnými informacemi. [6]

Cvičení je koncipováno tak, aby dvě skupiny studentů pracovaly se záměrně omezenými zdroji. Po samostatné práci následuje debata, kde skupiny obhajují své názory. Až na konci diskuze zjistí, že obě skupiny pracovaly s odlišnými informacemi. Tato situace má za cíl vyvolat tzv. "aha efekt".

Aplikace umožňuje učitelům vkládat vlastní výukové materiály, které mohou být použity v rámci cvičení.



Obrázek 1.5 Prototyp Perspekt

[6]

1.2 Zvolený prototyp

Na základě analýzy a rozhovorů s několika studenty střední školy jsem se rozhodla pro prototyp *Deepfake* vytvořený Viktorií Kolesárovou. Tento prototyp je zajímavý obsahově, provedením a zaměřuje se na aktuálně rostoucí problematiku spjatou s rozmachem generativní umělé inteligence.

Prototyp je řešený jako hra o několika levelech, mezi něž patří přiřazení správného nového článku, rozpoznání upravených fotografií, nebo rozšifrování manipulačních technik a analýza deepfake videa.

2 VÝBĚR TECHNOLOGIE

Tato část se věnuje představení platforem a technologií vhodných pro realizaci vybraného prototypu. Vzhledem k tomu, že se jedná o hru s předpokládanou jednorázovou návštěvností, je optimální zvolit webovou aplikaci. Mobilní aplikace by v tomto případě nebyla ideální, a proto se dále zaměříme na možnosti vývoje webových aplikací.

Obecně lze vývoj aplikací rozdělit do dvou hlavních částí:

- **Backend:** Tato část se stará o aplikační logiku, zpracování dat, komunikaci s databázovými systémy a správu uživatelských účtů. Backend je zodpovědný za bezpečnost, autentizaci a celkovou správu datových operací. [7]
- **Frontend:** Frontendová část se zaměřuje na prezentaci aplikace a interakci s uživatelem. Zajišťuje vizuální rozhraní, zobrazení dat získaných z backendu a implementaci konceptů UX/UI (User Experience a User Interface), které optimalizují uživatelské prostředí a přívětivost rozhraní. [7]

Vhodný výběr technologie a přístupu v rámci backendu závisí na povaze požadovaného produktu, přičemž je třeba zohlednit potřeby budoucí škálovatelnosti a velikost projektu. Jedním z velmi populárních přístupů je vývoj API (Application Programming Interface), kdy je backend zcela oddělen od frontendové části. Frontend komunikuje s backendem pomocí HTTP/HTTPS požadavků a získává data prostřednictvím tzv. endpointů, které obvykle vracejí data ve formátu JSON. [8][9]

Alternativně lze využít architekturu Model-View-Controller (MVC), kde backend generuje dynamické pohledy (views) a vrací kompletní HTML stránky s integrovanými daty, typicky získanými z databáze. Tento přístup není založen na striktním oddělení frontendové a backendové logiky, ale spíše na společné správě dat a prezentace, což může být výhodné u některých typů aplikací. [10]

V rámci frontendové části je možné, jak bylo zmíněno výše, jít cestou MVC, kdy je klientská logika integrována přímo do serverové aplikace. Alternativním přístupem je kompletní oddělení frontendové a backendové části, kdy frontend komunikuje se serverem prostřednictvím API. Pro oba tyto přístupy jsou základem technologie HTML, CSS a JavaScript, případně TypeScript, které jsou detailněji popsány v této kapitole. [10]

Vzhledem k rostoucí komplexitě moderních aplikací by však samotné tyto technologie bylo obtížné udržovat a rozšiřovat. Proto se v současnosti hojně využívají moderní frontendové frameworky, které zjednodušují vývoj a správu rozsáhlých aplikací. Mezi nejvýznamnější frameworky patří React, Angular, Vue.js a nově také Svelte, který si díky svému efektivnímu a intuitivnímu přístupu získává na popularitě.

2.1 Webové technologie

V této podkapitole jsou představeny základní webové technologie, které jsou základem pro vytváření moderních webových aplikací. Patří zde HTML, CSS a JavaScript, které společně tvoří základní kostru pro vývoj uživatelského rozhraní.

2.1.1 HTML

HTML (HyperText Markup Language) je základní stavební prvek každé webové stránky. Definuje jednotlivé elementy pomocí tagů, které obalují požadovaný obsah a tím mu dodávají strukturu. Jednotlivé tagy reprezentují elementy jako nadpisy, bloky textu, odkazy, obrázky, tlačítka nebo seznamy. Soubor s definicí webové stránky má tradičně koncovku .html. [11]

V současném vývoji webových aplikací se často využívají moderní frameworky (např. React, Angular nebo Vue), které dynamicky generují HTML na základě komponent a šablon. To však neznamená, že by HTML bylo nahrazeno – výsledná aplikace stále obsahuje HTML, jen je jeho generování abstraktnější a automatizovanější. Vzhled webové stránky je pak dále upravován kaskádovými styly (CSS) a dynamická funkčnost je zajišťována pomocí JavaScriptu nebo jeho nadstavby, TypeScriptu, který přidává statické typování a další vývojová vylepšení. [11]

2.1.2 CSS

CSS (Cascading Style Sheets) je jazyk, který definuje, jak se dokumenty psané v HTML nebo XML zobrazují na obrazovce, v tisku či na jiných médiích. Je součástí základních technologií otevřeného webu a je standardizován napříč všemi moderními webovými prohlížeči. Styly lze definovat přímo v HTML pomocí tagu `<style>`, nebo v externích souborech s příponou .css, které se do HTML dokumentu includují (vkládají) pomocí tagu `<link>`. Díky CSS lze oddělit obsah dokumentu od jeho vizuální prezentace, což zvyšuje flexibilitu a usnadňuje údržbu kódu. [12]

2.1.3 JavaScript

JavaScript je skriptovací jazyk, který umožňuje interaktivitu a dynamické chování webových stránek. V moderním webovém vývoji hraje zásadní roli, neboť je nezbytný nejen pro vytváření interaktivních webových aplikací, ale i mobilních a desktopových aplikací. Jeho rozmach byl podpořen technologiemi, jako je AJAX, který umožňuje asynchronní načítání dat bez opětovného načítání celé stránky, a populárními knihovnami a frameworky, jako jsou React, Angular či Vue, které značně usnadňují vývoj komplexních uživatelských rozhraní. [13]

JavaScript se rozšířil také na serverovou stranu pomocí prostředí Node.js, což umožňuje vývojářům využívat jednotný jazykový stack jak na klientské, tak na serverové straně. Díky dynamické povaze jazyka a možnosti pracovat s funkcemi jako s prvotřídními objekty podporuje JavaScript různé programovací styly – objektově orientovaný, funkcionální i imperativní, což mu poskytuje značnou flexibilitu a adaptabilitu při řešení širokého spektra problémů. [13]

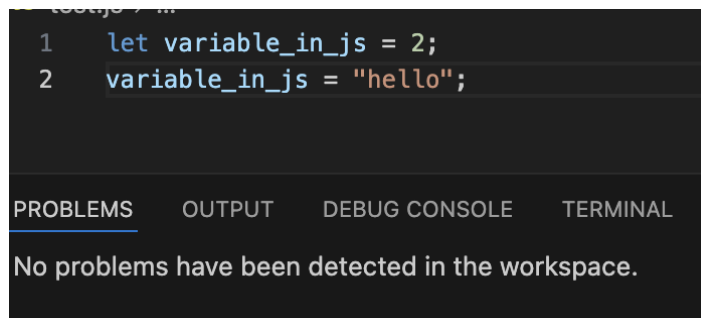
2.1.4 TypeScript

TypeScript je programovací jazyk vyvinutý společností Microsoft, který představuje nadmnožinu (superset) jazyka JavaScript. To znamená, že veškerý validní kód v JavaScriptu je zároveň validní v TypeScriptu, ale navíc TypeScript přináší statickou kontrolu typů, což umožňuje odhalit potenciální chyby již v době kompilace. Tím se výrazně zvyšuje robustnost a udržitelnost kódu, zejména u rozsáhlejších projektů. [14]

Klíčové rozdíly oproti JavaScriptu jsou:

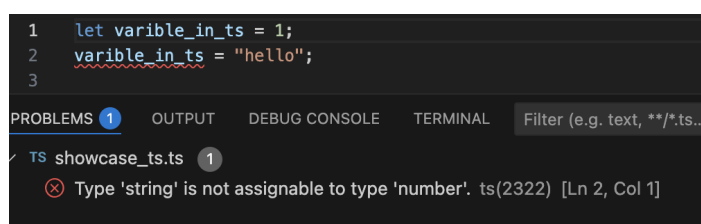
- **Statická typová kontrola:** Umožňuje explicitně definovat datové typy proměnných, funkcí a objektů, což pomáhá identifikovat chyby dříve, než se kód spustí.
- **Pokročilé jazykové konstrukce:** Podpora moderních konceptů, jako jsou rozhraní (interfaces), generické typy a třídy, usnadňuje psaní strukturovaného a přehledného kódu.
- **Kompilace do JavaScriptu:** Kód napsaný v TypeScriptu se kompiluje do standardního JavaScriptu. Díky tomu je čas běhu obou jazyků totožný, přičemž výhoda TypeScriptu spočívá pouze v dodatečné kontrole typů a lepší struktuře kódu. [13][14]

Hlavním přínosem TypeScriptu je tedy možnost včasné detekce chyb a zjednodušení refaktoringu, což je zásadní při vývoji složitých a škálovatelných aplikací. Díky těmto vlastnostem přináší TypeScript vyšší úroveň kontroly a konzistence kódu, což je velkou výhodou pro vývojové týmy.[14]



Obrázek 2.1 Ukázka JavaScript

(Zdroj: vlastní)



Obrázek 2.2 Ukázka TypeScript

(Zdroj: vlastní)

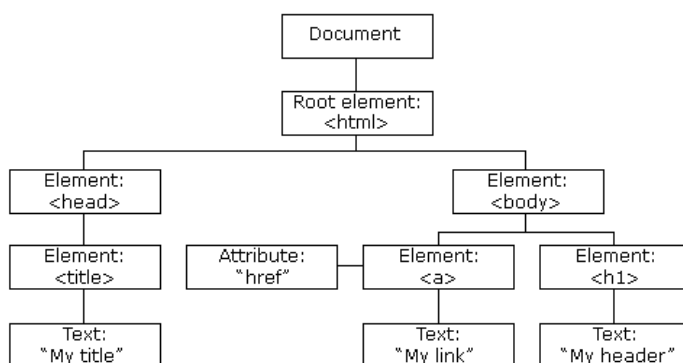
2.2 Kompilační metody

Kompilační metody se zaměřují na způsoby, jakými je kód zpracován a přenesen do finální podoby, kterou používá webová aplikace.

2.2.1 Virtuální DOM

Virtuální DOM je využíván například ve frameworkách jako React nebo Vue. Klasický DOM představuje strukturu vytvořenou při načtení stránky, která je reprezentována jako strom objektů. [15]

Virtuální DOM (VDOM) představuje koncept, při kterém je vytvořena ideální neboli "virtuální" reprezentace uživatelského rozhraní, uložená v paměti. Tato virtuální verze DOM je následně synchronizována s reálným DOM, což umožňuje efektivní manipulaci s uživatelským



Obrázek 2.3 DOM

[15]

ským rozhraním. Proces synchronizace mezi virtuálním a reálným DOM se nazývá *reconciliation* (rekonciliace).[16]

Při změně stavu aplikace se vytvoří nová verze virtuálního DOM, která je porovnána s předchozí verzí. Na základě zjištěných rozdílů se provádějí pouze nezbytné změny v reálném DOM, což výrazně zrychluje vykreslování a zlepšuje výkon aplikace. [16]

Tento přístup umožňuje deklarativní tvorbu uživatelského rozhraní, kdy vývojář pouze specifikuje, jak má UI vypadat na základě aktuálního stavu aplikace, a knihovna (např. React) se postará o to, aby byl reálný DOM vždy v souladu s těmito požadavky. [16]

2.2.2 Přímá manipulace s DOM

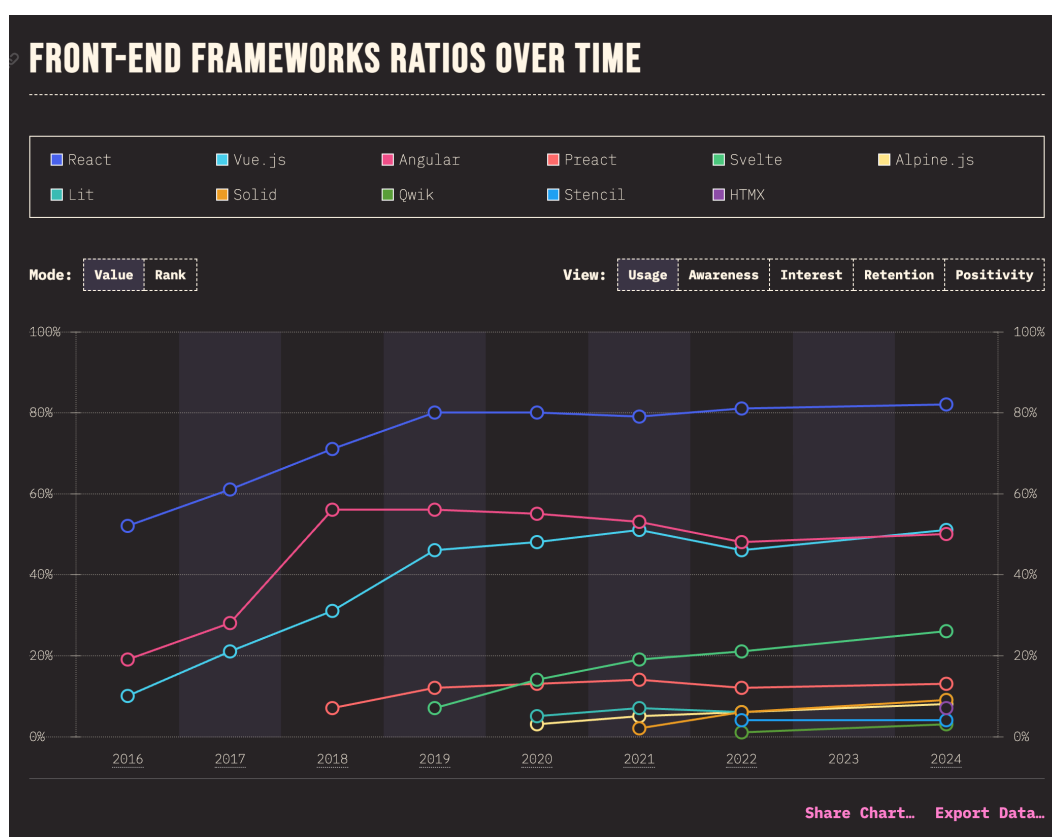
Další metodou je přímá manipulace s DOM, kterou využívá například framework Svelte. U tohoto přístupu se zdrojový kód již během sestavování projektu převede přímo do optimalizovaného JavaScriptu. Místo provádění složitých operací s virtuálním DOM výsledný kód manipuluje přímo s reálným DOM. To přináší několik výhod:

- Konečný kód je menší a rychlejší, protože obsahuje pouze nezbytné instrukce pro manipulaci s DOM, čímž se eliminuje potřeba rozsáhlého runtime prostředí;
- kompilace odstraňuje nepoužívaný kód (tzv. tree shaking) a provádí další optimalizace, což přispívá k rychlejšímu načítání a lepší odezvě aplikace;
- statická analýza během kompilace umožňuje odhalovat syntaktické a logické chyby ještě před nasazením do produkce, čímž se zvyšuje bezpečnost a stabilita výsledného produktu. [16]

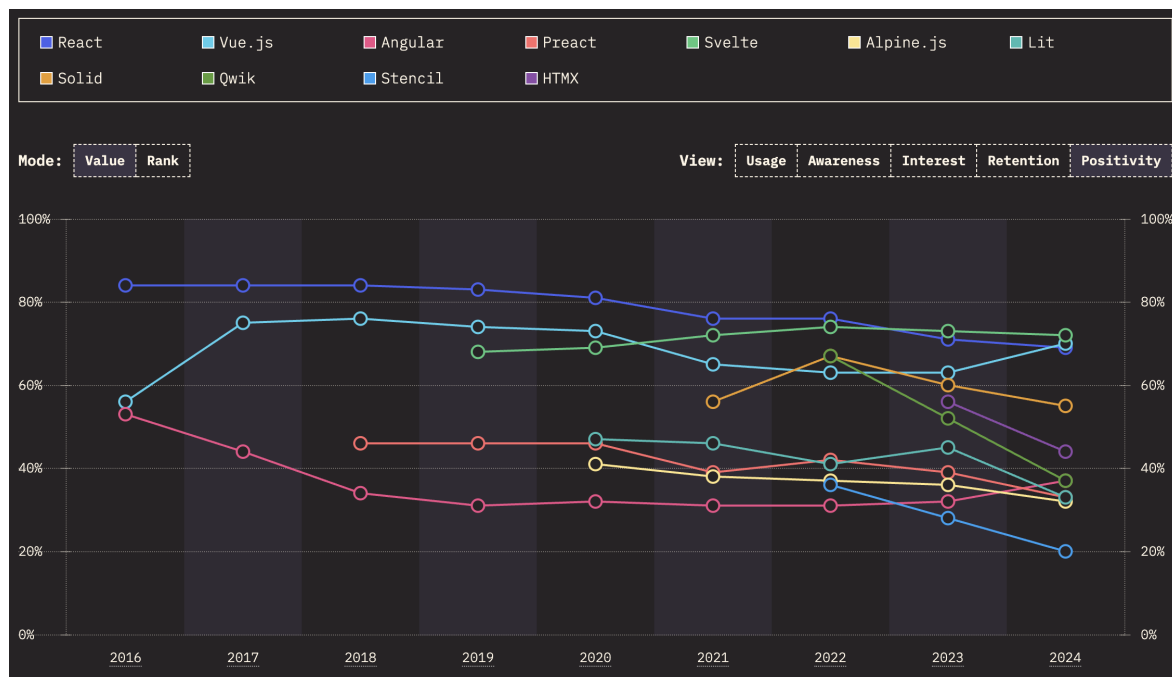
2.3 Frontend

V rychle se vyvíjejícím světě webového vývoje se staly JavaScriptové frameworky základem vytváření interaktivních a responzivních aplikací. Tyto frameworky jsou navrženy, aby urychlily a zjednodušily celý vývojový proces. Nabízí strukturovaný přístup, který velmi napomáhá údržbě a škálování komplexních moderních aplikací. React, Vue, nebo Angular, ale v posledních letech i rostoucí Svelte, jsou napříč frontendovým světem nepostradatelnou součástí.

State of JavaScript je každoroční průzkum, který od roku 2016 sbírá odpovědi od více než 20 tisíc vývojářů zaměřených na JavaScriptový ekosystém. Zkoumá, jaké technologie se aktuálně používají, jaké je povědomí o nových nástrojích a jak jsou vývojáři spokojeni s těmi stávajícími. V rámci průzkumu se sbírají i demografické údaje o respondentech (např. jejich země původu). Všechna sesbíraná data jsou dostupná na interaktivní webové stránce <https://2024.stateofjs.com>, kde si lze prohlížet podrobné statistiky a trendy z oblasti JavaScriptu. [17]



Obrázek 2.4 Vývoj užívání frontendových frameworků



Obrázek 2.5 Vývoj pozitivního hodnocení frontendových frameworků

[17]

Na grafu *Vývoj užívání frontendových frameworků* je bezkonkurenčně nejpoužívanějším frameworkem React, který si tuto pozici drží již od roku 2016. Ačkoli se mu v roce 2018 začal přibližovat Angular, v následujících letech se tento trend nepotvrdil a Angular společně s Vue.js zůstávají za Reactem. Nelze si však nevšimnout rostoucího využití technologie Svelte, která od roku 2019 výrazně nabrala na popularitě. Pokud bude tento trend pokračovat, může se z ní stát reálná konkurence pro aktuálně nejpoužívanější trojici.

Nárůst Svelte neukazuje jen uvedený graf, ale také pohled na kategorii Positivity (podíl pozitivních hodnocení), kde se Svelte umístilo na první příčce, a na kategorii Retention (pozitivní hodnocení těch, kteří technologii použili), kde je hned za technologií Solid.

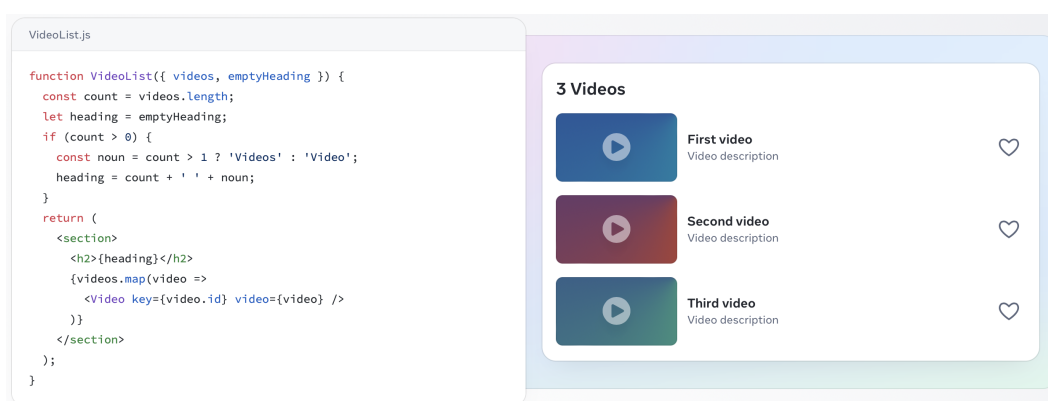
2.3.1 React

React je open-source JavaScript knihovna vyvinutá původně společností Facebook (nyní mateřskou společností Meta) pro vytváření uživatelských rozhraní. Základním stavebním kamenem Reactu jsou komponenty – samostatné, opakovaně použitelné bloky kódu, které lze začlenit na více místech aplikace. I když React není kompletním programovacím jazykem, ale spíše knihovnou zaměřenou na UI, od roku 2013, kdy byl široké veřejnosti zpřístupněn

(přes počáteční smíšené ohlasy), se stal standardem pro webový vývoj. [18]

React komponenty jsou znovupoužitelné HTML elementy, jejich skládáním vzniká komplexní aplikace. Mezi takové komponenty může patřit např. přihlašovací formulář nebo přednastavené galerie. Tím, že se aplikace rozdělí na menší komponenty, aplikace je přehlednější. Způsob práce v Reactu je tzn. deklarativní. Každý komponent může a většinou má své vlastnosti a vnitřní stav. React následně dokáže automaticky reagovat na změny těchto stavů a podle přednastavených akcí aplikaci překreslovat. [19]

Dalším významným konceptem je JSX (zkratka pro JavaScript + XML), který umožňuje psát HTML kód přímo v JavaScriptu. Díky tomu lze z funkce vrátit skutečný HTML element, nikoli pouze řetězec s jeho obsahem. Mimo jiné to také usnadňuje podmíněné renderování pomocí příkazů if–else. [20]



Obrázek 2.6 Ukázka kódu pomocí React

[19]

Aktuálně je React používaný aplikacemi Facebook, Netflix, Yahoo, Slack, Microsoft a mnoha dalšími.[19]

2.3.2 Angular

Angular je kompletní open-source vývojový framework vyvinutý společností Google postavený na TypeScriptu. Angular jako platforma obsahuje sbírku dobře integrovaných knihoven, které pokrývají širokou škálu funkcí, včetně routování, správy formulářů, komunikace mezi klientem a serverem a dalších.[21]

Dokumentace Angularu uvádí, že pro pochopení možností frameworku je nutné si osvojit následující koncepty:

- **Komponenty**

Komponenty jsou základními stavebními kameny každé Angular aplikace. Podobně jako v Reactu představují samostatné, znovupoužitelné bloky kódu, které mají vlastní logiku a odpovídají za určitý prvek uživatelského rozhraní (například formulář nebo navigační lištu). Každá komponenta je definována třídou (typicky v TypeScriptu), doplněnou o dekorátor `@Component`, a je svázána s HTML šablonou a styly (CSS, SCSS, atd.). Díky tomuto přístupu lze aplikaci rozdělit na menší, lépe spravovatelné části.[21]

- **Šablony (Templates)**

Šablony slouží k deklarativnímu popisu uživatelského rozhraní, které komponenta vykresluje. Angular poskytuje speciální syntaxi (tzv. Angular templating), která zahrnuje:

- *Interpolaci*: Vkládání proměnných z logiky komponenty do HTML (např. `{{ myVariable }}`).
- *Datové vazby* (Data binding): Propojení vlastností komponenty s elementy v šabloně (např. `event binding (event)="handler()"`).
- *Strukturální direktivy*: Umožňují podmíněné vykreslování nebo opakování prvků (např. `*ngIf`, `*ngFor`).
- *Atributové direktivy*: Ovlivňují vzhled či chování jednotlivých elementů (např. `[class.active]="isActive"`).

Šablona tak tvoří „most“ mezi logikou komponenty (ve třídě) a výsledným HTML zobrazením v prohlížeči. [21]

- **Direktivy**

Direktivy jsou třídy, které přidávají nebo upravují chování a vlastnosti elementů v aplikaci. Angular rozlišuje tři základní typy direktiv:

- *Komponenty*: Specifický druh direktiv, který kromě logiky obsahuje i šablonu a styly (viz výše).

- *Atributové direktivy* (Attribute Directives): Mění vzhled či chování existujícího elementu (např. zvýraznění textu po najetí myší).
- *Strukturální direktivy* (Structural Directives): Ovlivňují strukturu DOM, tedy přidávají, odebírají nebo mění celé prvky (např. `*ngIf`, `*ngFor`). [21]

- **Dependency Injection (DI)**

Dependency Injection je mechanismus, který umožňuje přehledně spravovat závislosti mezi jednotlivými částmi aplikace (např. služby a komponenty). Angular poskytuje vlastní kontejner pro DI, který dokáže automaticky vyhledat a dodat potřebné služby (nebo jiné závislosti) do komponenty či jiné služby. Tento přístup usnadňuje testování, znovupoužitelnost kódu a celkovou udržitelnost aplikace. [21]

Na Angularu jsou přirozeně postavené webové aplikace společnosti Google jako Gmail, Google Docs nebo YouTube, mimo to například časopis Forbes nebo PayPal.[22]

2.3.3 Vue.js

Posledním z trojice nejpoužívanějších technologií pro tvorbu front-endových částí aplikací je *Vue.js*, často zkráceně jen *Vue*. Jedná se o moderní, progresivní JavaScriptový framework, který vznikl v roce 2014 jako odlehčená alternativa k robustnějším řešením, jako jsou Angular nebo React. V porovnání s jinými frameworky nabízí Vue možnost postupné integrace do již existujících HTML stránek – proto se o něm často mluví jako o „progresivním“ frameworku. Tato vlastnost mimo jiné usnadňuje přechod z knihoven typu jQuery, neboť Vue lze aplikovat pouze na vybranou část projektu, aniž by bylo nutné předělávat celou aplikaci.[23]

Vue je primárně zaměřeno na tvorbu uživatelského rozhraní (UI) a lze je bez větších obtíží integrovat téměř do libovolného projektu. Kromě interaktivních prvků a designu však Vue dokáže pohánět i kompletní *Single-page aplikace* (SPAs), a to díky oficiálním nástrojům, jako je *Vue Router* pro směrování a *Vuex* či *Pinia* pro správu stavu. [23]

2.3.4 Svelte

Svelte.js je frontendový framework, který – podobně jako jiné technologie – využívá HTML, CSS, JavaScript a případně TypeScript. Hlavním technologickým rozdílem oproti ostatním

```
import { createApp, ref } from 'vue'

createApp({
  setup() {
    return {
      count: ref(0)
    }
  }
}).mount('#app')
```

```
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Result



Obrázek 2.7 Ukázka kódu pomocí Vue

[24]

frameworkům je, že Svelte nepracuje s virtuálním DOM, ale během kompilace generuje optimalizovaný JavaScriptový kód. Tento přístup vede k rychlejší odezvě aplikace a ke snížení velikosti výsledného kódu, což zlepšuje uživatelský zážitek. Svelte tak umožňuje vývoj jak jednoduchých komponent, tak i komplexnějších full-stack aplikací.[25]

Kód se typicky píše do souboru *+page.svelte*, který může být rozdělen do tří částí:

- `<script lang="ts">...</script>` – obsahuje logiku a funkcionalitu, psanou v JavaScriptu nebo TypeScriptu,
- HTML – definuje strukturu stránky,
- `<style>...</style>` – určuje vzhled komponenty/stránky.

Všechny tyto části jsou umístěny v jednom souboru, což usnadňuje správu a vývoj jednotlivých komponent. Samotné Svelte stránky jsou routovány na základě hierarchie a názvů složek. Vše začíná kořenovou složkou *routes*, která je dále členěna do podadresářů, například s dynamickými názvy (např. *[productId]*) či statickými (např. *products*). [25]

V rámci .svelte dokumentů je také velmi snadné používat proměnné přímo v HTML kódu – stačí je vložit do složených závorek (např. {nazevPromenne}), což přispívá k intuitivnímu a přehlednému zápisu. [26]



Obrázek 2.8 Ukázka kódu pomocí Svelte

(Zdroj: vlastní)

2.4 Backend

Backend, neboli serverová část aplikace, je základ aplikací, který není na první pohled vidět. Stará se o ukládání a správu dat, komunikaci s databází, má na starost autorizaci a autentizaci uživatelů a v neposlední řadě ovlivňuje i celkový výkon. Při vývoji serverové části je třeba rozhodnout, zda bude aplikace založena na architektuře API, kde jsou logika a prezentační vrstva odděleny, nebo na MVC/MVT modelu, kde jsou tyto vrstvy více propojené. Při přístupu založeném na API je serverová část zcela oddělena od prezentační vrstvy, přičemž komunikace probíhá prostřednictvím protokolů HTTP nebo HTTPS.[27]

Většina webových aplikací má několik společných rysů, jako autentizace, získávání informací z databáze, nebo správu cookies. Při vývoji by pak programátoři museli vyvíjet vesměs stejné funkce stále dokola. I proto, stejně jako pro frontend, máme i pro backend k dispozici frameworky. Mezi ty nejpoužívanější patří Django, Spring, PHP nebo ASP .NET Core. Backendové frameworky jsou tedy jistý ekosystém, který pomáhá automatizovat jednotlivé kroky vývoje. [27]

2.4.1 Architektura systému

Jedním z klíčových aspektů při vývoji software je zvolení architektury systému. Existuje celá řada přístupů, ale mezi ty nejčastěji zvažované patří přístupy založené na model-view-controller (MVC) a na aplikačních programovatelných rozhraních (API).

2.4.2 Model-View-Controller

Model-View-Controller (MVC) je vzor, který rozděluje logiku aplikace do tří základních vrstev:

- **Model** – Vrstva zodpovědná za správu a logiku dat. Zajišťuje ukládání, načítání a validaci dat a komunikuje s databázemi.
- **View** – Vrstva, která prezentuje data uživateli. Obsahuje uživatelské rozhraní (UI) ve formě formulářů, tabulek, tlačítek apod., čímž umožňuje vizuální interakci s aplikací.
- **Controller** – Vrstva, která zprostředkovává komunikaci mezi modelem a pohledem. Řídí zpracování uživatelských vstupů, vyhodnocuje akce a předává data mezi ostatními vrstvami.

[28]

Obdobně funguje i architektura MVT využívaná pro framework Django, která jen pro stejný princip využívá pozměněné názvosloví, a to Model-View-Template. V tomto případě definice Model zůstává stejná, View se stará o aplikační logiku a Template o layout a prezentaci dat uživateli. [29]

2.4.3 Application Programmable Interface (API)

API je mechanismus, který umožňuje komunikaci mezi softwarovými komponentami pomocí předem definovaných rozhraní a protokolů. (Zdroj: [9])

Architektura API je často vysvětlována pomocí pojmů klient a server. Aplikace, která odesílá požadavek, se označuje jako klient, zatímco aplikace, která poskytuje odpověď, funguje jako server. V současnosti je velmi rozšířený protokol REST, který umožňuje flexibilní a jednoduché předávání dat mezi klientem a serverem. V REST architektuře klient odesílá požadavky obsahující data, server je zpracuje, vykoná potřebné operace a následně vrátí výsledek zpět klientovi. [9]

REST definuje sadu funkcí, jako jsou GET, PUT, DELETE, POST a další, která pak klient využívá k získávání nebo modifikaci dat na serveru. Data se přenášejí pomocí protokolu HTTP, přičemž odpovědi často přicházejí ve formátu JSON nebo jednoduchého textu. [30]

Hlavním rysem REST API je jeho bezstavovost (statelessness). To znamená, že server neuchovává stav mezi jednotlivými požadavky. Každý požadavek (request) je samostatnou *transakcí*, podobně jako URL adresa, kterou uživatel zadá do webového prohlížeče. Odpověď na požadavek je typicky převedena do formy textu či JSON. [9]

API Endpointy představují konkrétní adresy URL, na které klient odesílá požadavky. Každý požadavek se skládá z následujících částí:

- **Adresa** – Kombinace základní URL a specifického endpointu, např. `http://www.youapi.cz/api/getUser/user_id`
- **Headers** – Hlavičky obsahující metadata požadavku, jako jsou informace o autentizaci, typu obsahu (např. `Content-Type: application/json`) a další potřebné informace.
- **Body** – Tělo požadavku, které nese data k odeslání (u metod jako POST nebo PUT). U GET požadavků se tělo obvykle nevyužívá.

[30]

2.4.4 Django

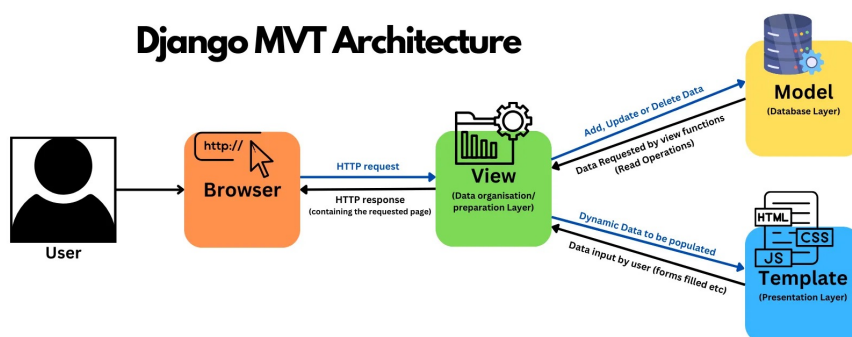
Django je vysokoúrovňový webový framework napsaný v jazyce Python, který výrazně usnadňuje rychlý vývoj a údržbu webových aplikací. Framework se řídí principem "*Don't Repeat Yourself*" (DRY, česky "Neopakuj se"), což podporuje minimalizaci duplicitního kódu a zvyšuje přehlednost celého projektu. [31]

Django využívá architekturu podobnou modelu MVC, avšak s mírně odlišnou terminologií – místo Controller se používá Template, což vede k modelu MTV (Model, Template, View).

- **Model** představuje datovou vrstvu, která definuje strukturu dat a zajišťuje komunikaci s databází.

- **View** obsahuje aplikační logiku, zpracovává HTTP požadavky a určuje, jaká data budou předána do šablony.
- **Template** slouží jako prezentační vrstva, která definuje, jak budou data zobrazena uživateli.

Toto oddělení zajišťuje, že logika, data a prezentace jsou v aplikaci jasně rozděleny, což přispívá k lepší škálovatelnosti, udržitelnosti a bezpečnosti vývoje webových aplikací.



Obrázek 2.9 MVT architektura v Django

(Zdroj: [32])

```

1 from django.http import JsonResponse
2
3 def example_view(request):
4     data = {'message': 'Hello from Django View!'}
5     return JsonResponse(data)

```

Listing 1 Příklad Django

2.4.5 Laravel

PHP framework Laravel nabízí kompletní ekosystém pro webový vývoj. Pro rychlý začátek vývoje nabízí tzv. *Starter kits* s integrovanými frontend frameworky React nebo Vue. Tyto balíčky obsahují mimo jiné routování, kontrolery i verifikaci uživatelů. [33]

Mimo vytvoření fullstack MVC aplikace, lze vytvářet i standalone APIs. Pro ukládání do databází využívá technologie *Eloquent ORM*, která nahrazuje přímou aplikaci SQL příkazů v kódu a podporuje tak čistý a přehledný kód. [34]

```
1 namespace App\Http\Controllers;
2 use Illuminate\Http\Request;
3
4 class ExampleController extends Controller
5 {
6     // Handles GET requests to the /example route.
7     public function index()
8     {
9         $data = ['message' => 'Hello from Laravel Controller!'];
10        return response()->json($data);
11    }
12 }
13 }
```

Listing 2 Perzistentní paměť

Kromě výše uvedených funkcí nabízí Laravel řadu dalších nástrojů a vlastností, které výrazně usnadňují vývoj a zvyšují produktivitu:

- **Blade templating engine:** Umožňuje vytvářet dynamické šablony s jednoduchou a přehlednou syntaxí, což usnadňuje oddělení logiky od prezentace.
- **Artisan CLI:** Výkonný příkazový nástroj, který automatizuje běžné úlohy, jako je generování kódu, správa databázových migrací nebo čištění cache.
- **Middleware:** Poskytuje možnost filtrovat a zpracovávat HTTP požadavky, čímž zajišťuje bezpečnostní kontroly, například autentizaci a autorizaci.
- **Podpora testování:** Integrovaná podpora unit a feature testů usnadňuje zajištění kvality aplikace a umožňuje snadnější refaktoring kódu.
- **Real-time komunikace:** Pomocí nástrojů jako Laravel Echo lze implementovat real-time funkce, například notifikace nebo chatovací systémy.
- **Komunitní podpora a dokumentace:** Rozsáhlá dokumentace a aktivní komunita přispívají k rychlému řešení problémů a rozvoji frameworku prostřednictvím řady doplňkových balíčků.[34]

[34]

2.4.6 ASP.NET Core

ASP.NET Core (.NET) je open-source multiplatformní framework pro vytváření aplikací pomocí technologie .NET, který umožňuje vytvářet webové aplikace, API servery, microservices nebo mobilní aplikace. Podporuje mnoho programovacích jazyků jako C#, F#, Visual Basic, nebo NodeJS. [27]

```
1 using Microsoft.AspNetCore.Mvc;
2 namespace MyApp.Controllers
3 {
4     [ApiController]
5     [Route("api/[controller]")]
6     public class ExampleController : ControllerBase
7     {
8         // GET: api/example
9         [HttpGet]
10        public IActionResult Get()
11        {
12            var data = new { Message = "Hello from ASP.NET Core
13                ↪ Controller!" };
14            return Ok(data);
15        }
16    }
17 }
18 }
```

Listing 3 Ukázka ASP.NET Core

Technologie .NET jako taková představuje robustní a flexibilní platformu, která nabízí celou řadu klíčových výhod:

- **Vysoký výkon a škálovatelnost:** Optimalizovaný runtime a podpora asynchronního programování umožňují vytvářet rychlé a efektivně škálovatelné aplikace.
- **Multiplatformnost:** Aplikace postavené na ASP.NET Core lze spustit na operačních systémech Windows, Linux i macOS, což usnadňuje nasazení v různých prostředích.
- **Modularita a flexibilita:** Díky modulární architektuře je framework lehce přizpůsobitelný specifickým potřebám projektu a vývojáři tak mohou využívat pouze ty komponenty, které jsou nezbytné.
- **Integrovaná podpora dependency injection:** Umožňuje snadnou správu závislostí, což vede k čistšímu a testovatelnějšímu kódu.
- **Bohatý ekosystém a nástroje:** Platforma disponuje množstvím knihoven, nástrojů a rozšíření, které usnadňují vývoj moderních cloudových a serverových aplikací. [35]

[35]

ASP.NET Core rovněž umožňuje snadnou integraci s dalšími službami a technologiemi, jako jsou relační i NoSQL databáze, cloudové platformy nebo frontendové frameworky. Díky aktivní komunitě a otevřenému zdrojovému kódu se framework neustále vyvíjí a zdokonaľuje, čímž poskytuje stabilní a moderní prostředí pro vývojáře po celém světě. [35]

II PRAKTICKÁ ČÁST

3 NÁVRH APLIKACE

3.1 Specifikace požadavků se spolkem Zvol si info

Po vzájemném schválení vybraného prototypu Deepfake jsme zahájili práce na návrhu finální aplikace. Z úvodních schůzek vyplynulo, že výsledná podoba aplikace se bude lišit od původního prototypu. Reprezentanti spolku Zvol si info postupně komunikovali své požadavky a možnosti implementace. Ve všech klíčových oblastech jsme dosáhli shody a komunikace probíhala transparentně, organizovaně a v pravidelných intervalech.

Aplikace je koncipována jako interaktivní příběh, kterým je uživatel postupně provázen „levely“ (úrovněmi). V rámci této práce byl vyvinut příběh pro jedno téma, přičemž bylo třeba dojednat i technické detaily, jako jsou registrace a správa uživatelů, administrace úkolů, flexibilita příběhu (změna pořadí úkolů, editace obsahu úkolů).

3.2 Registrace a přihlašování

Hra je navržena jako jednorázová interakce, proto se nevyžaduje registrace ani přihlášení pro běžné hráče. Přesto byly z důvodu správy obsahu a přístupu do administrátorské sekce implementovány moduly pro registraci a přihlášení. Po nasazení aplikace budou však tyto funkce skryté a přístupné pouze členům spolku Zvol si info pomocí odkazu.

3.3 Průběh hry

V této kapitole je popsána struktura samotné hry, jejímž jádrem je příběh s názvem *Mars-Gate*. Příběh je cílen na dospívající a dospělé od 15 let a skládá se z pěti úrovní, které vedou hráče k rozvoji kompetence mediální gramotnosti. Postupně přechází od základních znalostí přes praktické dovednosti až k vytváření kritického postoje.

Každá úroveň obsahuje:

- **Příběhový posun:** Fáze, ve které se hráč nachází a kontext úkolu.
- **Interaktivní úkol:** Zapojení hráče – typ aktivity (kvíz, drag-and-drop, ověřování obrázku apod.).
- **Vzdělávací cíl:** Co se má hráč naučit (znalosti, dovednosti, postoje).
- **Fun fact:** zajímavost ke zvolenému tématu.

3.3.1 Průvodní příběh

Země je na pokraji ekologického kolapsu. Mezi lidmi kolují spekulace, že elity (politici, bohatí investoři, celebrity) připravují tajný útěk na Mars a zbytek lidstva tam nechají na umření. Konspirátor Ivan publikuje senzaci podloženou „důkazy“ (fotografie, videa):

„Vláda už na Marsu dávno přistála, začíná tam stavět a teď to tají!“

Úkolem hráče je prověřit, zda jde o pravdu, či podvrh, osvojit si techniky odhalování manipulace a pochopit, jak se podobné dezinformace šíří. Na konci hry aplikace ukáže, jak snadno lze manipulovat s dostupnými zdroji a lidskou psychikou (konfirmační zkreslení, heuristiky apod.).

3.3.2 Úvodní obrazovka hry

Na úvodní obrazovce je uživateli stručně představen samotný příběh spolu s ilustrační fotografií. Obrazovka je navržena s důrazem na přehlednost a intuitivní ovládání, aby se hráč mohl plně soustředit na obsah hry bez zbytečných rozptylovacích prvků. Barevné schéma a typografie pak pomáhají zdůraznit klíčové prvky a zachovat grafickou konzistenci celého projektu.

Ve spodní části stránky se nachází časová osa, na níž jsou znázorněny jednotlivé úrovně, které hráče čekají, avšak bez bližších podrobností. Tento přehled má v hráči vzbudit zvědavost a motivovat ho k postupnému objevování dalších částí příběhu.

Další důležitou součástí obrazovky je vysvětlení progress baru – tedy ukazatele, který se v průběhu hry dynamicky mění. Pro účely tohoto projektu byl zvolen tzv. reverzní progress bar, nazvaný *Míra chaosu ve společnosti*. Tento indikátor zobrazuje počet lidí ztracených v informačním chaosu. S každou úspěšně dokončenou úrovní se hodnota ukazatele snižuje, čímž symbolizuje, že hráč pomáhá ostatním lépe se orientovat ve světě dezinformací a obnovovat důvěru ve fakta.

Než může hráč hru skutečně spustit, je vyzván k zadání své přezdívky, pod níž bude ve hře vystupovat. Po potvrzení volby přezdívky se automaticky aktivuje tlačítko, kterým se dostane do hry.

3.3.3 Level 1 - První zpráva

Na stránce se objeví Facebookový příspěvek od uživatele *@MarsRebel*, kde autor s výraznou naléhavostí oznamuje „konec mlčení“ a tvrdí, že disponuje exkluzivními materiály získanými z kontaktů ve vládních kruzích. Použitím opakovaných vykřičníků a dramatických frází (např. „Pravda musí ven!!!!!!“) vytváří dojem bezprostředního odhalení velkého spiknutí. Hlavní sdělení spočívá v tom, že „elity už na Marsu budují nový svět“, zatímco běžné občany masivně ovlivňují dezinformace, aby skryly svou tajnou misi. Příspěvek uzavírá výzva ke zhlédnutí prvních unikátních fotografií, čímž autor motivuje čtenáře ke sdílení a šíření svého obsahu. V tomto příspěvku jsou zobrazeny i fotografie, které mají celému příspěvku přidat na věrohodnosti.

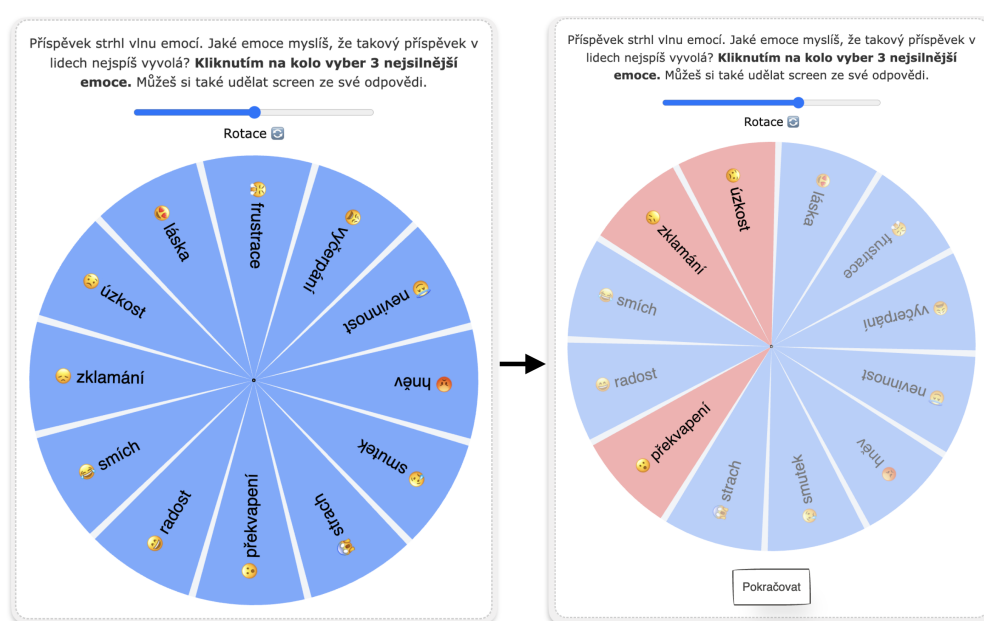


Obrázek 3.1 Facebookový příspěvek

(Zdroj: vlastní)

Na další stránku hráč přejde kliknutím na tlačítko *Pokračovat*. Po zobrazení příspěvku se tlačítko po dobu dvaceti sekund „načítá“ – objeví se jednoduchá animace, které zabraňuje okamžitému přechodu dále. Tento záměrný časový odstup pomáhá hráči věnovat zvýšenou pozornost právě zobrazenému textu a dát mu prostor k přemýšlení nad obsahem.

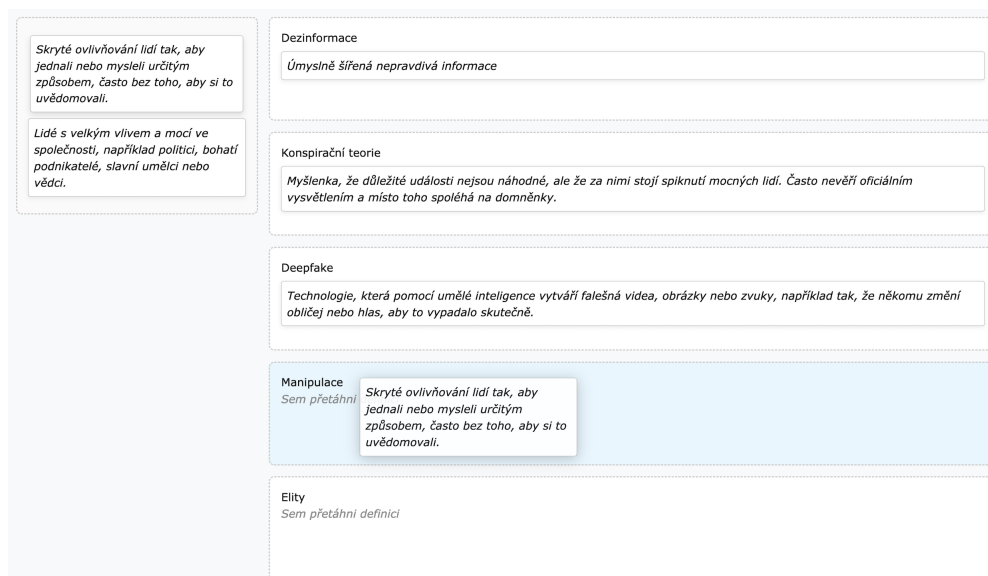
Jakmile uplyne stanovená doba a hráč tlačítko odklikne, dostane se na stránku, kde má možnost reflektovat své pocity. Aby to nebylo příliš složité, má k dispozici interaktivní "Kolo emocí" rozdělené do několika základních kategorií (radost, zklamání, znepokojení, zvědavost, strach aj.), na kterém má kliknutím vybrat tři nejsilnější emoce, které to v něm vyvolalo. Kolo lze libovolně otáčet a měnit výběr tak dlouho, dokud hráč nenajde přesně ty emoce, které v něm předchozí část vyvolala. Až ve chvíli, kdy jsou všechny tři sloty obsazeny, se aktivuje tlačítko *Pokračovat*, což hráče zároveň povzbudí k zamyšlení nad tím, proč právě tyto pocity vybral.



Obrázek 3.2 Kolo emocí

(Zdroj: vlastní)

V závěrečné části první kapitoly hráč obdrží soubor několika klíčových pojmů z oblasti dezinformační problematiky – například „dezinformace“, „deepfake“ nebo „konspirační teorie“. K jednotlivým termínům jsou připraveny definice, které se zobrazí ve levé části obrazovky. Pomocí drag-and-drop rozhraní musí hráč přetáhnout každou definici do správného políčka k odpovídajícímu pojmu. V případě chybných přiřazení se definice automaticky vrátí do výchozího seznamu, což hráče přirozenou cestou vede k opakování a upevnění znalostí. Tento přístup byl zvolen spolkem zvolsi.info z toho důvodu, že jsou definice jednoznačné a je žádoucí, aby se hráč nad významy jednotlivých výrazů zamyslel. Po úspěšném spárování všech pojmů aplikace nabídne možnost pokračovat do další kapitoly.



Obrázek 3.3 Drag-and-drop rozhraní na pojmy

(Zdroj: vlastní)

3.3.4 Level 2 - Kdo a proč?

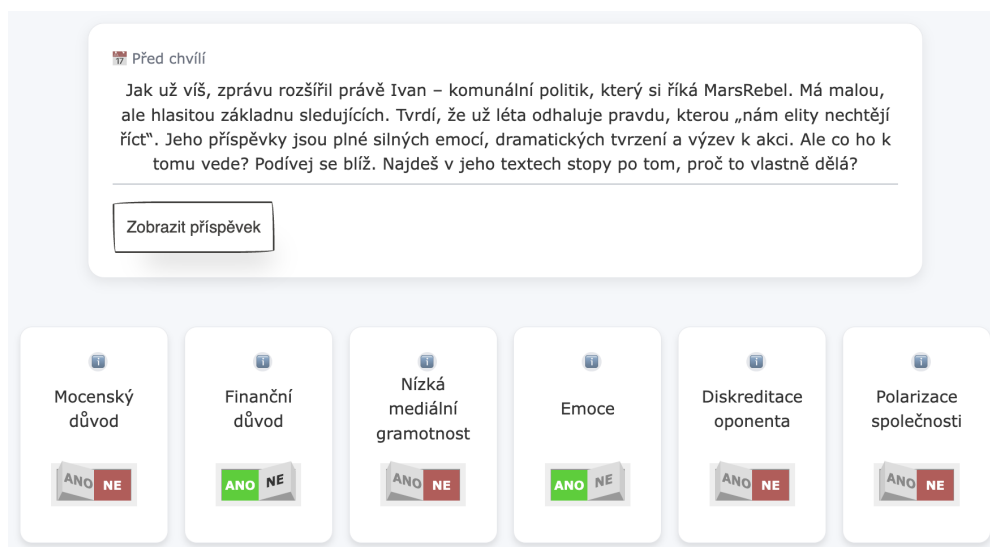
V této části se hráč podrobně zabývá rozбором Facebookového příspěvku a zkoumá, jaké motivace mohly vést jeho autora k jeho zveřejnění.

Po vstupu do úrovně se zobrazí zadání: hráč si má z nabídky vybrat dva nejsilnější důvody, které z textu příspěvku nejvýrazněji vyplývají. Mezi přednastavenými možnostmi jsou například:

- „Mocenský důvod“
- „Diskreditace oponenta“
- „Nízká mediální gramotnost“

U každé z nabízených možností si hráč může zobrazit definici najetím myši na tlačítko *i*.

U každé položky jsou interaktivní přepínače *Ano / Ne*. Hráč přepíná dva vybrané důvody do polohy *Ano*. Jakmile jsou označeny dva důvody a výběr je správný, aktivuje se tlačítko *Pokračovat*, které hráče posune dál. V případě, že odpověď správná není, hráč je vyzván, aby svůj výběr změnil. Během celé doby může hráč kdykoli otevřít původní příspěvek kliknutím na tlačítko *Zobrazit příspěvek*, aby si připomněl kontext a zkontroloval správnost své volby.

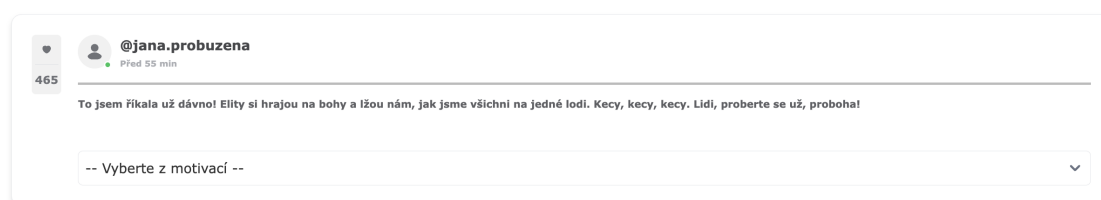


Obrázek 3.4 Motivace příspěvku

(Zdroj: vlastní)

Tento úkol rozvíjí schopnost kritického myšlení a pomáhá hráči odhalovat skryté záměry autorů dezinformačního obsahu.

V další fázi se hráči zobrazí čtyři komentáře, u nichž má pomocí rozbalovacího seznamu vybrat ten nejdominantnější motiv. Pojmy zůstávají stejné jako v předchozím kole (např. „Mocenský důvod“, „Diskreditace oponenta“, „Nízká mediální gramotnost“ apod.). Po do-



Obrázek 3.5 Motivace komentáře - příklad

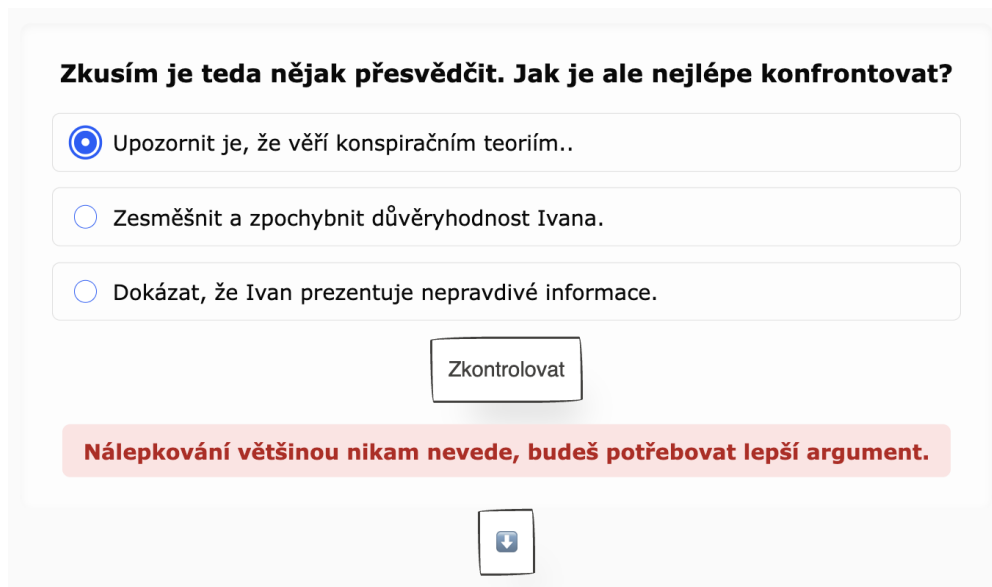
(Zdroj: vlastní)

končení výběru hráč klikne na tlačítko *Zkontrolovat výsledky*. Následně se:

- **zeleně** označí komentáře s správně vybranou motivací,
- **červeně** zvýrazní komentáře se špatnou volbou, u nichž se navíc zobrazí krátké odůvodnění správného motivu.

3.3.5 Level 3 - Důkazy pod lupou

Na přechodu do třetí úrovně hráče čeká krátký „můstkový“ kvíz, jehož cílem je seznámit ho s myšlením o původu a ověřování obrázků. Tento kvíz obsahuje čtyři zvědavé otázky, které hráče navádějí k uvědomění si různých možných zdrojů či úprav fotografií.



Obrázek 3.6 Příklad můstkového kvízu

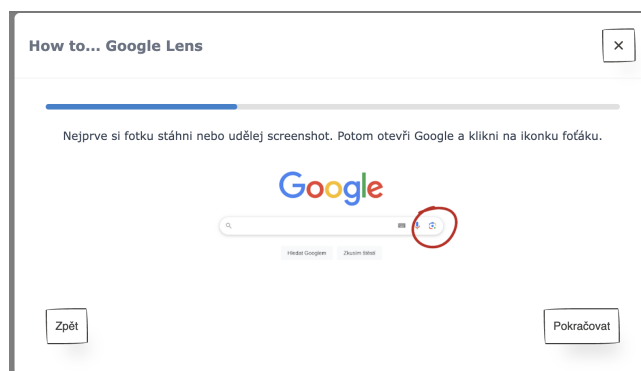
(Zdroj: vlastní)

Po úspěšném dokončení kvízu se zobrazí informační obrazovka s oznámením, že uživatel *@MarsRebel* publikoval nové snímky, které údajně dokládají přistání na Marsu. Kliknutím na tlačítko *Pokračovat* hráč spustí šestikrokový tutoriál k nástroji Google Lens. V něm se na příkladu z pohádky *Princové jsou na draka* naučí, jak:

1. otevřít aplikaci Google Lens
2. naimportovat fotografii do vyhledávání
3. porovnat výsledky s dostupnými zdroji
4. vyhodnotit spolehlivost nalezených informací

Hráč si může tutoriál kdykoli znovu zobrazit tlačítkem *Znovu zobrazit návod*.

Ve vlastním úkolu jsou obrázky zobrazeny ve dvou sloupcích: vlevo série fotografií, vpravo seznam hypotetických zdrojů. Pomocí kliknutí na fotografii a poté na zdroj vytvoří



Obrázek 3.7 Návod na používání Google Lens

(Zdroj: vlastní)

hráč dvojici, o níž se domnívá, že k sobě patří. V případě omylu může dvojici rozpojit a zkusit přiřazení znovu.

Jakmile hráč spáruje všechny fotografie, klikne na *Zkontrolovat odpovědi*. Aplikace pak zvýrazní zeleně ty dvojice, které hráč určil správně, a označí červeně chybné přiřazení. Současně zobrazí správná spojení a krátké vysvětlení správného zdroje. Kliknutím na tlačítko se dostane do další části.

V další fázi úrovně hráč nejprve seřadí jednotlivé kroky ověřování fotografií pomocí Google Lens. Kroky se zobrazují v *seřaditelném seznamu*, kde je hráč přetahuje myší do logického pořadí. Po kliknutí na tlačítko *Zkontrolovat pořadí* obdrží okamžitou zpětnou vazbu, zda je seřazení správné. Pokud není, hráč může pořadí upravovat, dokud není zcela správné.

Bezprostředně poté navazuje animace psaní komentáře pod přezdívkou, kterou si hráč zvolil na začátku hry. Text komentáře se postupně odhaluje a hráč v něm vyzývá ostatní uživatele k ověřování zdrojů a shrnuje kroky práce s Google Lens. Jakmile je komentář kompletní, aktivuje se tlačítko *Odeslat komentář*, kterým hráč tento level završuje a *publikuje* tím svůj komentář.

3.3.6 Level 4 - Proč tomu lidé věří

V závěrečné příběhové části hráč čelí reakci komunity na svůj komentář s návodem pro Google Lens. Místo podpory se objevily komentáře, které zcela odmítají kritické ověřování informací a zdrojů.

Jako interaktivní úkol se zobrazí otázka: „*Proč se ostatní lidé nepřidali na tvou stranu, i když jsi poskytl/a jasné důkazy?*“ Hráč vybírá ze tří přednastavených odpovědí, z nichž

Zpětné vyhledávání obrázků přes vyhledávač máš za sebou. Nebylo to tak těžké, co? Škoda, že tohle neví každý, pak by si mohl informace snadno ověřit...
Tak co kdybychom je to naučili!

Sestav správné pořadí jednotlivých kroků pro vyhledání obrázku přes Google Lens:

- :: Ve vyhledávací liště na Googlu klikni na ikonku foťáku
- :: Z nabídnutého seznamu vyber stránku, na které se fotka objevila
- :: Klikni na zobrazit přesné shody
- :: Fotku stáhni nebo vyscreenshotuj
- :: Vlož svůj soubor do Google Lens

Zkontrolovat pořadí

Obrázek 3.8 Úkol na používání Google Lens

(Zdroj: vlastní)

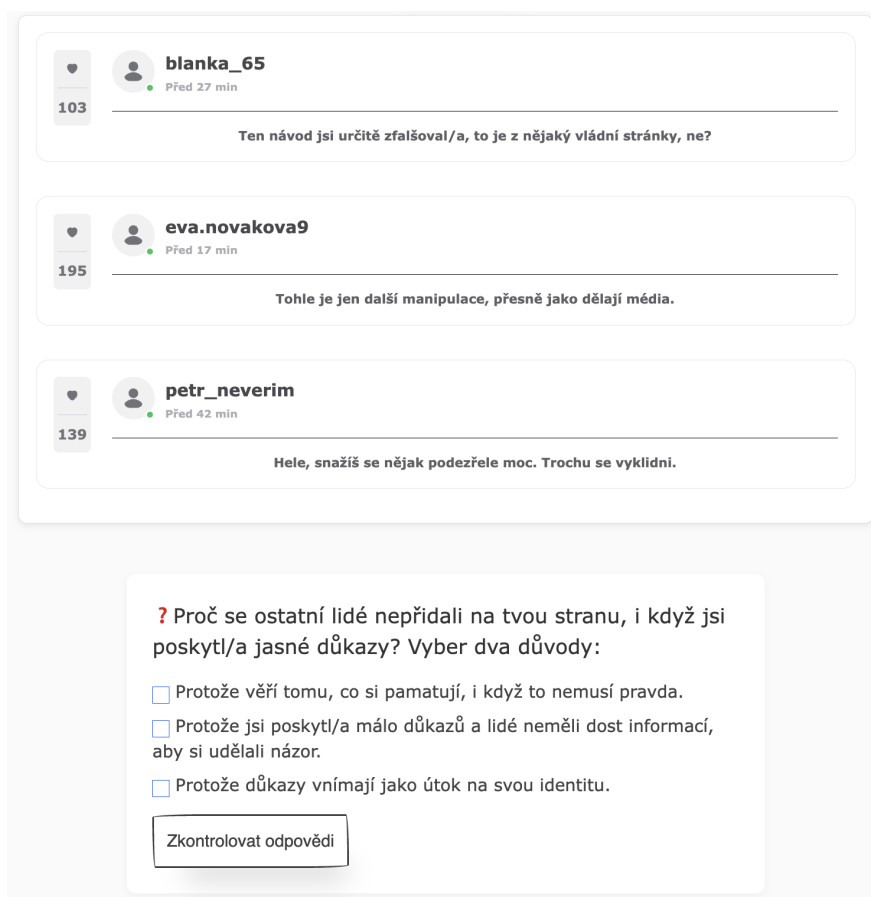
dvě jsou správné. Při označení chybné možnosti systém upozorní, že nějaká odpověď není správná, a vybízí k opětovnému výběru. Tento mechanismus procvičuje identifikaci překážek v komunikaci a pochopení psychologických bariér.

V další části hráči rozbalovací seznam nabízí termíny kognitivních zkreslení, u nichž je třeba vybrat odpovídající definici. Po vyplnění všech položek klikne na tlačítko *Zkontrolovat definice*, které zeleně označí správné a červeně chybná přiřazení. Hráč musí nesprávné definice opravit, aby mohl pokračovat.

Následně se zobrazí konkrétní příklady kognitivních zkreslení z průběhu příběhu. Například u *Spáčova efektu* se doplňuje situace:

„I když sledující později zjistili, že jde o dezinformaci, zůstal jim pocit: „něco o misi na Marsu jsem někde slyšel...““

Hráč přetažením příkladu k definici přiřadí každý příklad k příslušnému zkreslení a opět



Obrázek 3.9 Úkol v levelu 4

(Zdroj: vlastní)

klikne na *Zkontrolovat*. Zde zůstává stejný princip kontroly - hráči se buď zobrazí možnost pokračování ve hře, nebo ho hra upozorní, aby některé definice opravil.

Nyní se hra dostává k ukončení příběhové části před konečným kvízem. Na obrazovce se uživateli zobrazí notifikace s popiskem *Nová zpráva*. Po rozkliknutí tohoto tlačítka se otevře chat, kde je vidět zpráva od uživatele @*realjan47*, který v první zprávě uvádí, že podle hráčova komentáře skutečně ověřil několik fotografií. Bezprostředně poté se zobrazí výzva zdůrazňující význam kritického myšlení a tlačítko *Začít hru*, kterým hráč spustí závěrečný úkol.

V následném drag-and-drop rozhraní musí hráč roztřídit tipy a praktiky pro rozvoj kritického myšlení. Ze seznamu přetahuje „správné“ rady do pole s doporučeními a „nevhodné“ tipy do koše. Mezi doporučené patří například:

- Podívat se na názor, se kterým nesouhlasím – a zkusit ho pochopit, ne rovnou odmít-

Kotvení

Vyberte správnou definici:

Při rozhodování spoléháme na první informaci, kterou dostaneme. ▼

Heuristika dostupnosti

Vyberte správnou definici:

Časem zapomeneme, odkud informace pochází, ale zůstane nám dojem, že je pravdivá. ▼

Spáčův efekt

Vyberte správnou definici:

Časem zapomeneme, odkud informace pochází, ale zůstane nám dojem, že je pravdivá. ▼

Obrázek 3.10 Přiřazení definic

(Zdroj: vlastní)

nout.

- Zastavit se, když cítím silnou emoci – nadechnout se, a chvíli neklikat nebo nesdílet dál.

Mezi nevhodné postupy pak:

- Věřit hlavně tomu, co sdílí moji přátelé – ti by mi přece nelhali.
- Pokud má zpráva hodně lajků, nemusím ji již ověřovat, určitě je pravdivá.

Po správném rozřazení se opět otevře chat, ve kterém se animuje odpověď @realjan47. Hráč navrhne kolegovi další tipy na budování kritického myšlení a povzbudí ho, aby se nebál ověřovat informace – každý krok k ověření se počítá. Na závěr se objeví další zpráva od @realjan47, který hráči poděkuje za užitečné rady a potvrdí, že si díky nim vybudoval lepší návyk kritického přístupu.

3.3.7 Level 5 - Závěrečný kvíz

V závěrečné části se ověřují znalosti, které hráč během hry nasbíral. Pro každý předchozí level je připravena minimálně jedna otázka:



Obrázek 3.11 Přiřazení příkladů

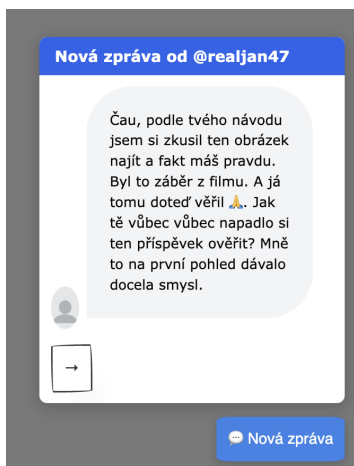
(Zdroj: vlastní)

- **Level 1:** Co je to dezinformace?
- **Level 2:** Když někdo používá silné výrazy jako „vláda nás zradila“ nebo „pravda, kterou vám neřeknou“, jde pravděpodobně o: *manipulační techniky*.
- **Level 3:** Jak můžeš spolehlivě ověřit pravost obrázku, který vidíš na sociálních sítích? (Například pomocí Google Lens.)
- **Level 4:** Uved' příklad konfirmačního zkreslení.

Po zodpovězení všech otázek hráč klikne na tlačítko *Zkontrolovat odpovědi*.

- Správné odpovědi se zvýrazní zeleně,
- chybné odpovědi červeně a současně se zobrazí krátké vysvětlení správného řešení.

Na základě zpětné vazby může hráč své znalosti zopakovat nebo přejít k závěrečnému shrnutí a vyhodnocení celé hry.



Obrázek 3.12 Zpráva od uživatele @realjan47

(Zdroj: vlastní)

REAKČNÍ PŘÍSPĚVEK <i>Co pomáhá</i> Podívat se na názor, se kterým nesouhlasím – a zkusit ho pochopit, ne rovnou odmítnout. Ověřit si zprávu, která potvrzuje, co si myslím – právě ty bývají nejzrádnější.	KOŠ 🗑️ <i>Čemu se vyvarovat</i> Říct si: „Nevím. Zjistím si víc.“ – a necítit se kvůli tomu hloupě.
Zastavit se, když cítím silnou emoci – nadechnout se, a chvíli neklikat nebo nesdílet dál.	
Věřit hlavně tomu, co sdílí moji přátelé – ti by mi přece nelhali.	Pokud má zpráva hodně lajků, určitě je pravdivá.
Odeslat	

Obrázek 3.13 Rozřazení tipů pro kritické myšlení

(Zdroj: vlastní)

3.3.8 Zpětná vazba

Po ukončení celé hry vyskočí hráči okénko, kde má hráč možnost hvězdičkami vyjádřit názor na hru a navíc i prostor rozepsat své hodnocení v textovém poli. Po vyplnění těchto polí se hráči zobrazí výběr jeho role - tedy zda je student, nebo školitel. Následně nehledě na vybrané roli se aplikace zeptá na jméno, věk a pohlaví. Pokud hráč vybral roli studenta, je mu na konci zobrazena stránka s poděkováním a s odkazem na sociální síť. Pro školitele pak možnost zanechat svůj email pro zasílání newsletteru. Zároveň je zde zaklikávací pole pro udělení souhlasu se zpracováním osobních údajů.

ZÁVĚREČNÝ KVÍZ

Jste připraveni zopakovat si, co jste se naučili?

Co je to dezinformace?

- ☐ Pravdivá informace podaná zábavnou formou
- ☒ Neověřený názor zveřejněný na internetu
- ☐ Úmyslně šířená nepravdivá nebo zavádějící informace

Tvrzení: "Přistání na Marsu nemá vědecký účel, jde o spiknutí elit, které se tam chystají žít," odpovídá nejvíce:

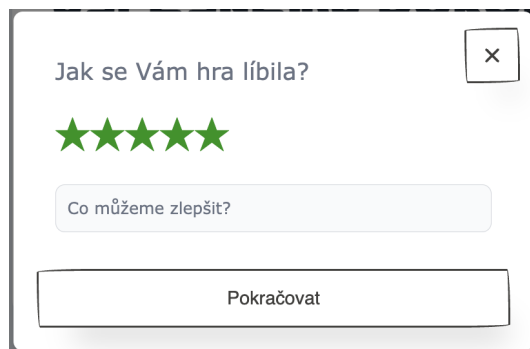
- ☐ konspirační teorii
- ☒ dezinformaci
- ☐ omylu

Obrázek 3.14 Konečný kvíz

(Zdroj: vlastní)

3.3.9 Fun fact

Po dokončení každého levelu se uživateli zobrazí vyskakovací okno (pop-up) s nějakým zajímavým faktem z oblasti dezinformací. Tyto fakty jsou generovány náhodně z interní databáze. Každé vyskakovací okno obsahuje samotný fakt a zároveň i odkaz. Přidáním přímého odkazu podporuje důvěru hráče ve věrohodné informační zdroje a motivuje ho k dalšímu samostatnému ověřování.



Obrázek 3.15 Zpětná vazba - hodnocení

(Zdroj: vlastní)

Obrázek 3.16 Zpětná vazba - role

(Zdroj: vlastní)

Obrázek 3.17 Fun-fact vyskakovací okno

(Zdroj: vlastní)

4 IMPLEMENTACE BACKEND

4.1 Vybraná platforma

Pro implementaci serverové části byla na základě předem definovaných požadavků vybrána technologie ASP .NET Web API. V praktické části byly popsány časté technologie k vývoji webového API. ASP .NET byl vybrán zejména z důvodu kladných předchozích zkušeností s vývojem.

4.2 Architektura aplikace

Vícevrstvá (multilayer) architektura rozděluje aplikaci na několik logicky oddělených vrstev, z nichž každá plní jasně definovanou roli. Typicky se rozlišuje prezentační vrstva, která přijímá HTTP požadavky a zajišťuje komunikaci s uživatelem; aplikační vrstva, která orchestruje business scénáře a spravuje transakce; doménová vrstva, v níž jsou definovány entity, hodnotové objekty a obchodní pravidla; a vrstva infrastructure, která realizuje přístup k databázi (prostřednictvím Entity Framework DbContext, migrací či Identity) a zprostředkovává volání externích služeb.

Toto rozčlenění umožňuje udržet kód přehledný a modularizovaný, protože každá vrstva řeší pouze svou specifickou odpovědnost (princip Separation of Concerns). Nízkourovněvé funkce, jako je validace vstupů nebo mapování dat, lze díky tomu sdílet napříč všemi částmi aplikace a vyhnout se duplicitám, což odpovídá zásadě DRY (Don't Repeat Yourself). Komunikace mezi vrstvami probíhá výhradně přes definovaná rozhraní, což zachovává zapouzdření (encapsulation) – vnitřní implementace zůstávají skryté a lze je měnit bez dopadu na okolní kód. [36]

K dalším výhodám patří snadná evoluce a testovatelnost: při přechodu na jinou databázovou technologii nebo upravení obchodní logiky stačí upravit danou vrstvu, aniž by to vyžadovalo zásahy do ostatních částí systému. Ve vícevrstvé architektuře platí, že vyšší vrstva může volat pouze definovaná rozhraní vrstvy nižší – opačná závislost je zakázána. Tento princip zaručuje, že změna implementace podkladové vrstvy nezpůsobí kaskádové úpravy v navazujících vrstvách. [36]



Obrázek 4.1 Blokové schéma

(Zdroj: vlastní)

4.2.1 Využití principů v aplikaci

Při návrhu a implementaci backendu byly uplatňovány osvědčené softwarové principy, které zajišťují přehlednost, testovatelnost a snadnou údržbu kódu. Aplikace je rozdělena do jasně vymezených vrstev: controller zajišťuje komunikaci s klientem a přijímá HTTP požadavky, aplikační služba (AppService) obsahuje veškerou obchodní logiku a DeepFakeDbContext spravuje přímý přístup k databázi. Toto rozdělení odpovídá principu Separation of Concerns – každá vrstva řeší pouze svou doménu a změna v jedné vrstvě nevyžaduje zásah v ostatních. [36]

V rámci tříd byly všechny závislosti předávány konstruktorům, čímž byl umožněn průběh Dependency Injection. Kontroller tak pracuje pouze s rozhraním AppService a nezná detaily jeho implementace, což usnadňuje případnou výměnu konkrétní implementace bez nutnosti úprav v kontroleru. Rozhraní AppService sama definuje pouze metody relevantní pro konkrétní scénář (např. CreateAsync, UpdateAsync, DeleteAsync), čímž je zajištěna Interface Segregation – žádná třída není nucena implementovat metody, které nepotřebuje. [36]

Metody pro přístup k databázi jsou implementovány asynchronně pomocí klíčových slov `async/await`. Tento přístup zamezuje blokování vláken při I/O operacích, zvyšuje propustnost serveru a zajišťuje lepší odezvu při větším zatížení. Každá třída navíc nese jedinou odpovědnost – controller pouze přijímá a vrací data, AppService zpracovává obchodní pravidla a DbContext pečlivě spravuje entitní model. Tím je naplněn Single Responsibility Principle, kdy se zjednodušuje ladění a minimalizují nečekané vedlejší efekty. [36]

AppService vrstva zároveň funguje jako repozitář – konsoliduje všechny operace nad entitami a poskytuje centrální místo pro přístup k datům. Tento Repository-Like Pattern podporuje opakované použití a přehlednost, neboť logika dotazů a úprav není rozptýlena mezi mnoho částí kódu. Veškeré podrobnosti práce s databází jsou přitom skryty za rozhraním služby, což zachovává Encapsulation a chrání integritu dat. [36]

Dále díky principu Convention over Configuration je směrování a zpracování HTTP požadavků řízeno atributy jako [HttpGet], [HttpPost] či [Route], bez potřeby rozsáhlých konfiguračních souborů. Díky tomu zůstává kód čistý, konzistentní a snadno se v něm orientuje. [36]

4.3 Implementace a návrh databáze

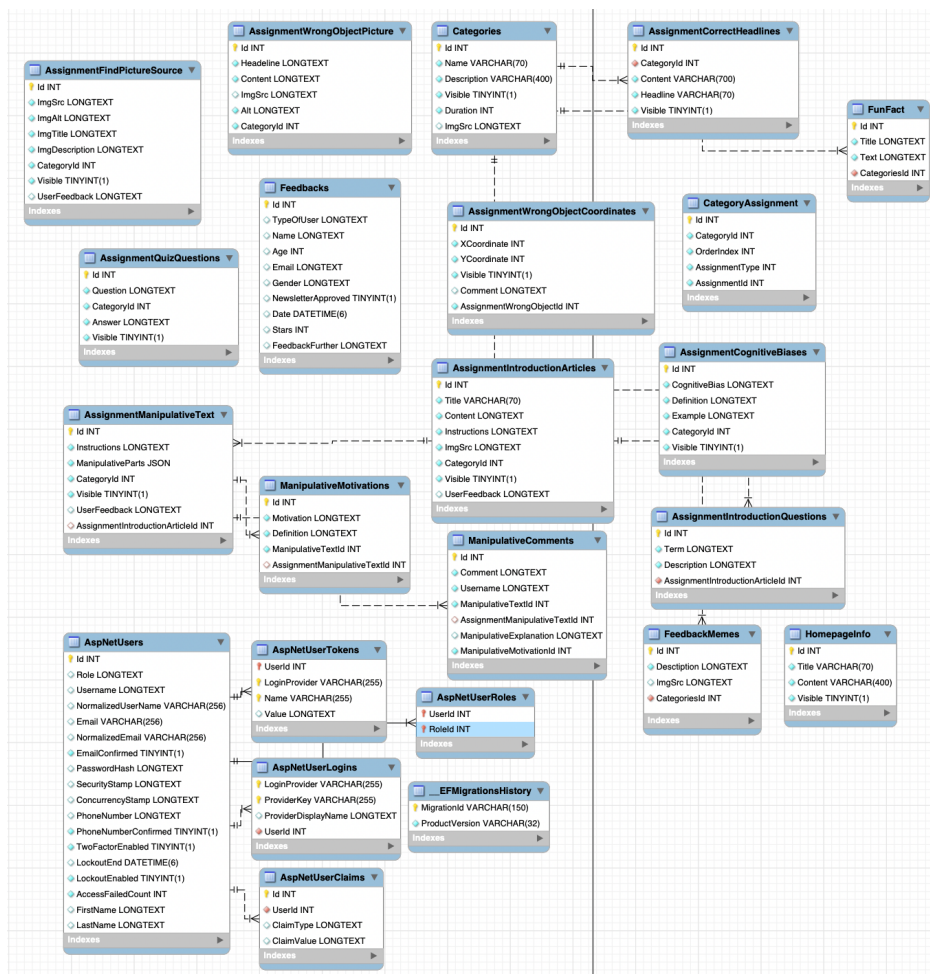
Jako databáze byla zvolena MySQL databáze. Databázový model se během vývoje výrazně proměňoval, a proto bylo nezbytné navrhovat ho od začátku tak, aby byl co nejvíce přizpůsobivý (častým) změnám. Z tohoto důvodu byl zvolen přístup *Code First*, který umožňuje definovat strukturu databáze prostřednictvím C# tříd. Díky EF Core migracím pak odpadá nutnost ručních zásahů přímo do databáze.

Aktuálně je vytvořeno celkem 25 tabulek s tím, že 6 z nich je vygenerováno Identity a 1 obsahuje informace o migracích. Databáze je aktuálně hostovaná pomocí AWS.

Vytvoření nové tabulky začíná přidáním souboru s entitou do doménové vrstvy, konkrétně do adresáře Entities. Zde se definují jednotlivé atributy, jejich datové typy a vlastnosti jako například maximální délka textové řetězce [StringLength(70)], povinné pole [Required] nebo nastavení závislostí mezi tabulkami pomocí primárních a cizích klíčů [ForeignKey(nameof(Categories))].

```
1 public class Category : Entity<int>
2 {
3     [Required]
4     [StringLength(70)]
5     public string Name { get; set; } = null!;
6
7     [Required]
8     [StringLength(400)]
9     public string Description { get; set; } = null!;
10
11     [Required]
12     public bool Visible { get; set; }
13
14     [Required]
15     public int Duration { get; set; }
16
17     public string? ImgSrc { get; set; }
18
19     [NotMapped]
20     [FileContentValidation("image")]
21     public IFormFile? Image { get; set; }
22
23     public ICollection<FunFact> FunFacts { get; set; } = new
24         ↳ List<FunFact>();
25     public ICollection<FeedbackMeme> FeedbackMemes { get; set; } = new
26         ↳ List<FeedbackMeme>();
27 }
```

Listing 4 Definice entity Category



Obrázek 4.2 ER diagram

(Zdroj: vlastní)

Pro zajištění komunikace mezi databází a entitami slouží kontext odvozený ze třídy `IdentityDbContext`, kde jsou deklarovány jednotlivé `DbSet` vlastnosti:

```

1 public class DeepFakeDbContext : IdentityDbContext<User, Role, int>
2 {
3     public DbSet<Category> Categories { get; set; }
4     public DbSet<AssignmentCognitiveBias> AssignmentCognitiveBiases { get;
5         ↪ set; }
6     public DbSet<FunFact> FunFacts { get; set; }
7     public DbSet<FeedbackMeme> FeedbackMemes { get; set; }
8     public DeepFakeDbContext(DbContextOptions<DeepFakeDbContext> options)
9         : base(options)
10    {
11    }
12    protected override void OnModelCreating(ModelBuilder modelBuilder)
13    {
14        base.OnModelCreating(modelBuilder);
15        // další konfigurace modelu lze přidat zde
16    }
17 }
18

```

Listing 5 Definice kontextu DeepFakeDbContext

Migrace se vytvářejí příkazem: `dotnet ef migrations add <NázevMigrace>` a aplikují do databáze pomocí: `dotnet ef database update`

4.4 Komunikace mezi vrstvami

V aplikační vrstvě byly definovány rozhraní, která vymezují akce určené k implementaci.

```
1 using DeepFake.Domain.Entities;
2 namespace DeepFake.Application.Abstraction;
3
4 public interface ICategoriesAppService
5 {
6     IList<Categories> GetCategories();
7     Task<Categories> GetCategoryById(int id);
8     Task<bool> Update(Categories category);
9     Task<bool> Create(Categories categories);
10    Task<bool> Delete(int id);
11    IList<Categories> GetActiveCategories();
12 }
13
```

Listing 6 ICategoriesAppService.cs

Následně byla ve stejné vrstvě, v adresáři Implementation, provedena implementace rozhraní pomocí třídy `CategoriesAppService : ICategoriesAppService`. Při neúplné implementaci všech členů rozhraní je hlášena kompilátorová chyba, např.: `Interface member 'Task<bool> ICategoriesAppService.Delete(int)' is not implemented.`

```

1  using DeepFake.Application.Abstraction;
2  using DeepFake.Infrastructure.Database;
3  using DeepFake.Domain.Entities;
4  using Microsoft.EntityFrameworkCore;
5
6  namespace DeepFake.Application.Implementation;
7
8  public class CategoriesAppService : ICategoriesAppService
9  {
10     DeepFakeDbContext _deepFakeDbContext;
11     private readonly IFileUploadService _fileUploadService;
12     private readonly string _destinationFolder = "findsource-pictures";
13
14     public CategoriesAppService(DeepFakeDbContext deepFakeDbContext,
15         ↪ IFileUploadService fileUploadService)
16     {
17         _deepFakeDbContext = deepFakeDbContext;
18         _fileUploadService = fileUploadService;
19     }
20
21     public IList<Categories> GetCategories()
22     {
23         return _deepFakeDbContext.Categories
24             .Include(x => x.FeedbackMemes)
25             .Include(x=>x.FunFacts)
26             .ToList();
27     }
28
29     public async Task<bool> Create(Categories category)
30     {
31         if (category.Image != null)
32         {
33             string filePath = await
34                 _fileUploadService.UploadFileAsync(category.Image,
35                 ↪ _destinationFolder);
36             category.ImgSrc = filePath;
37         }
38         _deepFakeDbContext.Categories.Add(category);
39         await _deepFakeDbContext.SaveChangesAsync();
40         return true;
41     }
42 }
43
44

```

Listing 7 CategoriesAppService.cs

Toto rozdělení je obecně považováno za přínosné z hlediska přehlednosti a škálovatelnosti kódu a podporuje princip Dependency Injection. Pro reálné použití v kontroleru je pak injektováno pouze rozhraní a využívány jsou služby definované v implementaci. Tím je zajištěno, že všechny metody deklarované v rozhraní jsou skutečně implementovány.

```
1 [Area("Admin")]
2 [Route("api/[area]/[controller]/[action]")]
3 [ApiController]
4
5 public class CategoriesController : Controller
6 {
7     private readonly ICategoriesAppService _categoriesAppService;
8     public CategoriesController(ICategoriesAppService
9         ↪ categoriesAppService)
10     {
11         _categoriesAppService = categoriesAppService;
12     }
13
14     [HttpGet]
15     public ActionResult<IEnumerable<Categories>> GetCategories()
16     {
17         var categories = _categoriesAppService.GetCategories();
18         return Ok(categories);
19     }
20
21     [Authorize(Roles = nameof(Roles.Editor) + "," + nameof(Roles.Admin))]
22     [HttpPost]
23     public async Task<ActionResult> Update([FromForm] Categories category)
24     {
25         bool result = await _categoriesAppService.UpdateAsync(category);
26         if (!result) return NotFound("Category not found");
27         return Ok(new { message = "Category updated successfully" });
28     }
29
30     [Authorize(Roles = nameof(Roles.Editor) + "," + nameof(Roles.Admin))]
31     [HttpPost]
32     public async Task<IActionResult> Create([FromForm] Categories
33         ↪ category)
34     {
35         await _categoriesAppService.CreateAsync(category);
36         return Ok(new { message = "Category created successfully",
37             ↪ category });
38     }
39 }
40
41 }
42
```

Listing 8 CategoriesController.cs

4.5 Manuální ověřování funkčnosti

V rámci kvality implementace bylo provedeno ruční testování následujících scénářů:

- Ověření CRUD operací nad všemi entitami pomocí softwaru Postman a rozhraní Swagger.
- Zpracování validace vstupních dat (null, příliš dlouhé řetězce) a kontrola návratových HTTP kódů.
- Funkčnost nahrávání a mazání obrázků
- Uživatelské akce ve frontendové hře – základní herní scénáře byly ověřeny v prohlížeči Chrome a Safari.

5 IMPLEMENTACE FRONTEND

Výběr vhodné platformy pro implementaci klientské části představuje klíčový krok, který vyžaduje pečlivé zhodnocení požadavků. U této konkrétní webové aplikace bylo nezbytné zvolit framework s podporou integrace herních prvků. V teoretické části byly analyzovány tři nejrozšířenější frameworky současnosti – React, Vue a Angular. Dále byla popsána i technologie Svelte, která nakonec byla pro implementaci zvolena. Svelte je rychle rostoucí inovativní framework. Mezi jeho nevýhody patří menší rozšířenost a omezená podpora na programátorských fórech. Implementace této hry však představovala ideální příležitost framework vyzkoušet a osvojit si jeho možnosti

5.1 Vybraná platforma - Svelte

Platforma Svelte je ve své nejnovější verzi 5.0 vybavena řadou zásadních změn ve srovnání s předchozími verzemi. Dokumentace a příklady dostupné online byly většinou vytvářeny pro verze starší než 5.0, takže došlo k ukončení podpory některých API, například `createEventDispatcher` nebo vybraných animačních přechodů. V případě, že se do kódu dostanou nepodporované funkce, jsou jejich výskyty ve vývojovém prostředí Visual Studio Code automaticky označeny přeškrtnutím.

5.2 Vytvoření projektu

Nejrychlejší a zároveň doporučovanou cestou dle oficiální dokumentace je vytvoření nového projektu pomocí příkazu

```
1 npx sv create nazev-aplikace
```

Tímto příkazem se vytvoří adresář *nazev-aplikace* s přednastavenou strukturou projektu. V adresáři je následně instalace závislostí provedena příkazem

```
1 npm install
```

Pro spuštění aplikace ve vývojovém režimu je použit příkaz

```
1 npm run dev
```

Po jeho zadání je aplikace dostupná na adrese `localhost:5173`.

5.3 Struktura složek

Typická struktura projektu SvelteKit je navržena tak, aby oddělovala zdrojové soubory, statické prostředky a konfigurační soubory do logických celků, které usnadňují vývoj a údržbu aplikace. Níže je uveden popis této struktury:

- **Kořenová složka projektu** (pojmenovaná při vytváření projektu) obsahuje:
 - `src/` – hlavní adresář se zdrojovými kódy aplikace.
 - `static/` – adresář pro statické prostředky (obrázky, fonty, ikony, manifesty apod.).
 - Konfigurační soubory:
 - * `package.json` – metadata projektu, skripty (`start`, `build`, `preview`) a seznam závislostí.
 - * `svelte.config.js` – konfigurace SvelteKit (adaptery, preprocesory, pluginy).
 - * `tsconfig.json` – konfigurace TypeScriptu (nastavení transpile, cesty aliasů, cílové prostředí).
 - * `vite.config.js` – konfigurace bundleru Vite (aliasy, pluginy, servery proxy, optimalizace).
- **Adresář `src`** obsahuje hlavní logiku a strukturu aplikace:
 - `lib/` – knihovna znovupoužitelných modulů a komponent, které nejsou přímo navázány na konkrétní routu.
 - `routes/` – zde se vytváří kompletní URL struktura aplikace pomocí složek a souborů (viz níže).
 - `stores/` – složka pro sdílení stavových dat napříč komponentami. Například `auth.ts` spravuje přihlášení uživatele a kontrolu relace, zatímco `app.ts` uchovává herní data jako jméno hráče, aktuální úroveň a stav progress baru.
 - `types/` – obsahuje definice TypeScript rozhraní ve souboru `interfaces.ts`, které popisují strukturu dat používaných v aplikaci a usnadňují komunikaci s API.

- Stavební skripty:

- `app.html` – základní HTML šablona, kam SvelteKit vkládá vykreslené komponenty a skripty.
- `hooks.client.js` a `hooks.server.js` – háčky (hooks) pro předzpracování požadavků a autentizaci.

5.3.1 Detailní popis složky `Adres routes/`

Adresář `routes/` je jádrem SvelteKit aplikace. Struktura složek a názvy souborů uvnitř `routes/` přímo určují, jaké URL adresy budou dostupné:

- `+page.svelte` – výchozí komponenta pro danou cestu (page component), vykreslí se při navigaci na odpovídající URL.
- `+layout.svelte` – sdílený layout pro všechny pod-mountované stránky, do kterého se vkládá `<slot />`. Napojení globálních stylů lze provést importem `global.css` právě zde.
- `+error.svelte` – komponenta pro zobrazení chybových stavů (404, 500 apod.) automaticky zpracovávaných SvelteKit.
- Dynamické routy – podsložky či soubory s hranatými závorkami (např. `[id]/+page.svelte`) umožňují definovat parametrizované cesty (např. `/products/3`).

5.3.2 Práce se `static/`

Veškeré soubory, které se nemění za běhu aplikace, např. obrázky, SVG ikony, fonty či externí JSON soubory, patří do složky `static/`. Při nasazení se její obsah zkopíruje přímo do kořenové URL, takže k nim odkazuje absolutně jako `/nazev-souboru.ext`.

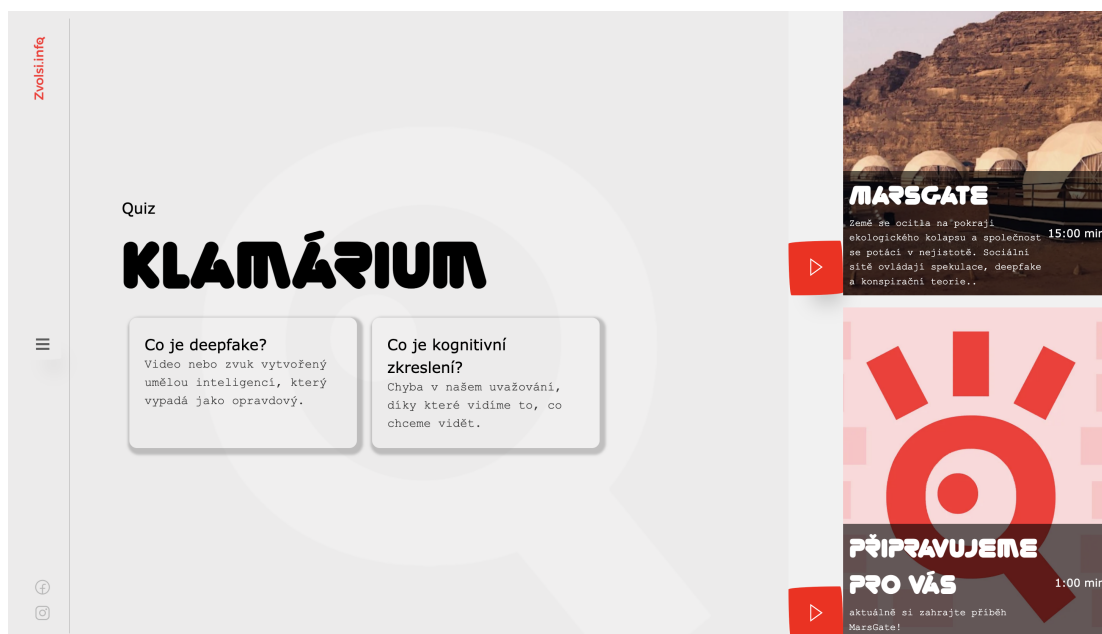
5.3.3 Komponenty v `src/lib/`

Pro zlepšení organizace kódu a opětovné použití se doporučuje vytvářet samostatné komponenty v `src/lib/`:

- `lib/assessments/` – uloženy jednotlivé kvízové úrovně (levely), sestavované do větších celků a importované do odpovídajících rout.

- lib/components/ – univerzální UI prvky (tlačítka, formuláře, karty, navigace apod.), které lze používat napříč všemi stránkami.

5.4 Úvodní strana



Obrázek 5.1 Úvodní strana

(Zdroj: vlastní)

Na úvodní straně je hráči zobrazen název hry o pojmy, které jsou s tématem spojené. V levé části hry má rozevřací menu a na pravé straně dynamický seznam dostupných her. Aktuálně je implementována pouze jedna hra a to MarsGate.

Z hlediska kódu je tato hlavní strana sestavena ze tří komponent - levé menu, hlavní obsah a výběr her/kategorií. Boční panel je samostatná komponenta, která je zahrnuta v již v šabloně aplikace - tedy souboru `+layout.svelte` a je automaticky zobrazena na každé stránce aplikace. Zbylé dvě komponenty `MainContent.svelte` a `VideoSection.svelte` jsou pak inkludovány a zobrazeny v souboru `+page.svelte` v kořenovém adresáři `routes`.

```

1  <script>
2    import MainContent from '$components/MainContent.svelte';
3    import VideoSection from '$components/VideoSection.svelte';
4    import './global.css';
5  </script>
6
7  <div class="container">
8    <MainContent marginRight="30%" />
9    <VideoSection/>
10  </div>
11

```

Listing 9 routes/+page.svelte

V rámci komponenty VideoSection jsou jednotlivé kategorie získány z databáze pomocí `categories.set()` jsou zapsány do lokální proměnné `categories`.

```

1  <script lang="ts">
2
3
4  import { onMount } from 'svelte';
5  import type { Category } from '$types/interfaces';
6  import { PUBLIC_VITE_API_ROOT } from '$env/static/public';
7  import { writable } from 'svelte/store';
8
9  const request_get = '/api/Admin/Categories/GetActiveCategories';
10
11  let categories = writable<Category[]>([]);
12
13  onMount(async () => {
14    const response = await fetch(PUBLIC_VITE_API_ROOT + request_get);
15    const data = await response.json();
16    categories.set(data);
17  });
18 </script>
19
20
21 <div class="scrollable-container">
22   {#each $categories as video (video.id)}
23     <div class="selection">
24       <button onclick={() => navigate("/marsGate")}
25         class="play-btn"></button>
26       <div
27         class="video-card"
28         style="background-image: url({video.imgSrc});"
29       >
30         <div class="video-info">
31           <div class="video-text">
32             <h2>{video.name}</h2>
33             <p>{video.description}</p>
34           </div>
35           <h3>{video.duration}:00 min</h3>
36         </div>
37       </div>
38     </div>
39   {/each}
40 </div>

```

Listing 10 VideoSection.svelte

Styl hlavní stránky je dále pozměněn i pro mobilní zobrazení pomocí CSS stylů pomocí `@media screen and (max-width: 768px)`

5.5 Hra MarsGate

Logika a posloupnost hry jsou definovány v souboru `routes/marsGate/+page.svelte`. V tomto hlavním souboru jsou jednotlivé levely importovány z oddělených komponent a má na starost následující funkcionality:

- Generování náhodného fun-factu
- Nastavení a zobrazení názvu aktuálního levelu nad progress-barem
- Řízení animace a hodnoty progress-baru

- Zprostředkování komunikace mezi jednotlivými levely

```

1  async function handleLevelComplete() {
2    if ($state_game !== 7 && $state_game !== 6) {
3      getRandomFunFact();
4    }
5    state_game.update((n) => (n < 7 ? ((n + 1) as 1 | 2 | 3 | 4 | 5 | 6 |
6      ↪ 7) : n));
7    await tick();
8    progress.set($progress - 0.25 * $progress);
9    setLevelName();
10  }
11
12  {#if $state_game === 4}
13    <FindSource_dnd onLevelComplete={handleLevelComplete} />
14  {/if}
15
16  {#if $state_game === 5}
17    <CognitiveBias onLevelComplete={handleLevelComplete} />
18  {/if}
19
20  {#if $state_game === 6}
21    <EndQuiz onLevelComplete={handleLevelComplete} />
22  {/if}

```

Listing 11 Navigace mezi levely

V každé komponentě levelu je využita interní asynchronní funkce `handleLevelComplete()`, která je vyvolána událostí `onLevelComplete`. Při jejím spuštění je:

1. vygenerován nový *fun fact* (pokud ještě nebyl dosažen konec hry),
2. inkrementována hodnota `state_game`,
3. snížena hodnota progress-baru,
4. aktualizován název levelu.

Aktualizované hodnoty progress-baru a čísla levelu jsou zároveň perzistentně uloženy podle logiky popsané v sekci „Perzistentní paměť“. Toto je zejména z důvodu, aby při nechtěném opuštění hry se hráč mohl vrátit na stejné místo.

5.6 Komunikace s backendem

Komunikace s REST API je v aplikaci realizována prostřednictvím nativní funkce `fetch`. Volání API se rozlišují na:

- *veřejné* (nevyžadující autorizaci),

- *chráněné* (vyžadující autorizaci – použité výhradně pro administrativní operace, které jsou od herní části odděleny).

Pro sjednocení základního URL všech endpointů byla v konfiguračním souboru `.env` definována proměnná `PUBLIC_VITE_API_ROOT`. Tato proměnná je importována do komponent vyžadujících data a slouží k sestavení adresy požadavku na konkrétní endpoint.

```

1  const response = await fetch(PUBLIC_VITE_API_ROOT + get_category, {
2      method: 'GET',
3      headers: {
4          'Content-Type': 'application/json',
5          Accept: 'application/json'
6      }
7  });

```

Listing 12 Komunikace s API ve Svelte

5.6.1 Definice datových rozhraní

Data získané pomocí API jsou ve formátu JSON. Aby však mohly být dále použity, je potřeba předem definovat objekt, který očekáváme. Pro zjednodušení a zajištění konzistence dat nejen napříč celou aplikací, ale i mezi backendem a frontendem je vhodné definovat tyto objekty na jednom místě. V rámci této aplikace je tím místem soubor `interfaces.ts` uložený v adresáři `src/types`. V tomto souboru jsou obsaženy veškeré objekty, které jsou v rámci aplikace použity. Objekty mohou mít jako atribut jiný objekt. Například pro hru s přiřazováním manipulačních technik ke komentářům je interface definován takto:

```

1  export interface ManipulativeTextComments{
2      id:number;
3      comment:string;
4      username:string;
5      manipulativeMotivationId:number;
6      manipulativeMotivation: ManipulativeMotivation;
7      manipulativeExplanation:string;
8      manipulativeTextId:number;
9  }
10
11 export interface ManipulativeMotivation{
12     id:number;
13     motivation:string;
14     definition:string;
15     manipulativeTextId:number;
16 }

```

Listing 13 Datová rozhraní

5.7 Perzistentní paměť

Pro ukládání informací o hráči – konkrétně zvolené přezdívky, aktuální úrovně hry a hodnoty progress baru – je využíváno perzistentní úložiště založené na `localStorage`. Pro jednot-

nou správu napříč celou aplikací je definována generická funkce `persisted<T>`, umístěná v souboru `src/stores/app.ts`:

```

1  import { writable, type Writable } from 'svelte/store';
2
3  function persisted<T>(key: string, init: T): Writable<T> {
4      let initial: T;
5      try {
6          const raw = localStorage.getItem(key);
7          initial = raw ? JSON.parse(raw) as T : init;
8      } catch {
9          initial = init;
10     }
11     const store = writable<T>(initial);
12     store.subscribe((v) => {
13         try {
14             localStorage.setItem(key, JSON.stringify(v));
15         } catch {
16         }
17     });
18     return store;
19 }
20
21 export const nickname = persisted<string>('nickname', '');
22 export const state_game = persisted<1 | 2 | 3 | 4 | 5 | 6 | 7
23   ↳ >('state_game', 1);
24 export const progress = persisted<number>('progress', 100);

```

Listing 14 Perzistentní paměť - definice

Při spuštění aplikace se nejprve provádí pokus o načtení existujících hodnot z `localStorage`; pokud nejsou nalezeny nebo nastane chyba, použijí se přednastavené hodnoty. Každá změna store je okamžitě serializována a uložena zpět.

Pro čtení či zápis je nutné importovat příslušný store a využít jeho metody:

```

1
2  import {progress } from '$stores/app';
3
4  // Nastavení hodnoty progress
5  progress.set($progress - 0.25 * $progress);
6
7  // Vymazání hodnoty z paměti
8  localStorage.removeItem('progress');

```

Listing 15 Perzistentní paměť

5.8 Administrativní část

Pro správu dat a obsahu jednotlivých her bylo vytvořeno administrátorské rozhraní, do kterého mají přístup pouze přihlášení uživatelé s rolí Admin, nebo Editor. V této části je možné dílčí části jednotlivých úkolů mazat, upravovat nebo je přidat. Konfigurace je dále popsána v kapitole *Bezpečnost řešení*.

Zpět na seznam

VYTVOŘIT OTÁZKU

Vyplňte prosím níže uvedený formulář pro vytvoření nové otázky s odpověďmi.

Otázka

Jaká může být jedna z hlavních motivací pro šíření dezinformací?

Kategorie

MarsGate

Odpovědi

Lidé chtějí šířit dobro	Správně? ☐	Odstranit
Pomoc lidem lépe se orientovat ve vědě	Správně? ☐	Odstranit
Pro vytvoření chaosu a strachu	Správně? ☒	Odstranit

Přidat odpověď

Vytvořit otázku

Obrázek 5.2 Administrátorský pohled

(Zdroj: vlastní)

5.9 Implementace konkrétní hry

5.9.1 Drag-and-drop - kognitivní zkreslení

Komponenta `dnd_board.svelte` byla vytvořena pro implementaci drag & drop hry zaměřené na kognitivní zkreslení. Jako vstup přijímá pole `columnItems`, které obsahuje metadata o sloupcích a položkách (viz obr. 3.11 „Přiřazení příkladů“).

Funkce `handleDndConsiderCards` a `handleDndFinalizeCards` zajišťují správnou aktualizaci pořadí položek během přetahování. Ověření správnosti přiřazení probíhá ve funkci `checkDropZones`, která je vyvolána stiskem tlačítka „Zkontrolovat“. Pokud jsou všechny položky přiřazeny správně, aktivuje se callback `onAllCorrect`, kterým je rodičovská komponenta informována o úspěšném dokončení úkolu. V kódu se odlišuje logika pro sloupec s id *examples*, to je právě pro sloupec, který slouží hráči jako výběr - na začátku hry jsou zde umístěny všechny příklady, které mají být přiřazeny. V obrázku níže je tento sloupec *examples* pojmenovaný *Doplň příklad k definici*.



Obrázek 5.3 Konkrétní příklad implementace drag-and-drop

(Zdroj: vlastní)

```

1 <script lang="ts">
2   import { flip } from 'svelte/animate';
3   import { dndzone } from 'svelte-dnd-action';
4   import { onMount } from 'svelte';
5   import '../styles/04_dnd.css';
6
7   export let columnItems: ColumnItem[] = [];
8   export let onAllCorrect = (msg: string) => {};
9
10  const flipDurationMs = 200;
11
12  function checkDropZones() {
13    let allCorrect = true;
14
15    columnItems = columnItems.map((column) => {
16      if (column.id !== 'examples') {
17        if (
18          column.items.length !== 1 ||
19          column.items[0].name.trim() !== column.correct_example.trim()
20        ) {
21          allCorrect = false;
22          return { ...column, card_error: true };
23        } else {
24          return { ...column, card_error: false };
25        }
26      }
27      return { ...column, card_error: false };
28    });
29
30    if (allCorrect) {
31      onAllCorrect(
32        'Všechny odpovědi jsou správné! Jsi připraven pokračovat dál?'
33      );
34    }
35  }
36
37  function handleDndConsiderCards(cid: number | string, e) {
38    const colIdx = columnItems.findIndex((c) => c.id === cid);
39    columnItems[colIdx].items = e.detail.items;
40    columnItems = [...columnItems];
41  }
42
43  function handleDndFinalizeCards(cid: number | string, e) {
44    const colIdx = columnItems.findIndex((c) => c.id === cid);
45    columnItems[colIdx].items = e.detail.items;
46    columnItems = [...columnItems];
47  }
48 </script>
49

```

Listing 16 Drag-and-drop implementace

V HTML části komponenty se pomocí bloku

`{#each columnItems as column (column.id)}` postupně generují jednotlivé karty `<div class="card">`. Na každou kartu je aplikována animace

`animate:flip= duration: flipDurationMs` a – v případě, že `column.card_error` má hodnotu `true` – také třída `card-error`, která vizuálně zvýrazní chybná přiřazení.

V hlavičce karty se zobrazuje název `column.name`.

V těle karty následuje element `<div class="dropzone">`, do kterého direktiva `use:dndzone= items: column.items, flipDurationMs, type: 'cards'` vkládá položky z pole `column.items`. Události `on:consider` a `on:finalize` volají funkce

`handleDndConsiderCards` a

`handleDndFinalizeCards`, jež upravují pole `columnItems` a tím zajišťují reaktivní překreslení. Uvnitř této zóny je pak další blok `{each column.items as item (item.id)}`, který vykreslí každou položku jako `<div class="card-item">` s animací `flip`.

Za všemi kartami je umístěno tlačítko

`<button on:click={checkDropZones}>Zkontrolovat</button>`, které při kliknutí spustí finální ověření správnosti přiřazení ve funkci `checkDropZones()`.

```

1 {each columnItems as column (column.id)}
2   <div class="card"
3     animate:flip={{ duration: flipDurationMs }}
4     class:card-error={column.card_error}>
5     <div class="card-header">
6       {column.name}
7       {if column.definition}
8         <p class="definition-tooltip"><strong>zobrazit
9           ↪ definici</strong></p>
10        <p class="form-select" id="hidden">
11          {column.definition}
12        </p>
13      {/if}
14    </div>
15    <div class="card-body">
16      <div
17        class="dropzone"
18        class:incorrect={column.card_error}
19        use:dndzone={{ items: column.items, flipDurationMs, type:
20          ↪ 'cards' }}
21        on:consider={{(e) => handleDndConsiderCards(column.id, e)}}
22        on:finalize={{(e) => handleDndFinalizeCards(column.id, e)}}>
23        {each column.items as item (item.id)}
24          <div class="card-item" animate:flip={{ duration:
25            ↪ flipDurationMs }}>
26            {item.name}
27          </div>
28        {/each}
29      </div>
30    </div>
31  {/each}
32  <button class="check-btn" on:click={checkDropZones}> Zkontrolovat
  ↪ </button>

```

Listing 17 Drag-and-drop zobrazení

6 BEZPEČNOST ŘEŠENÍ

V kontextu této aplikace je nutné zajistit, aby neoprávněné požadavky nebyly schopné měnit obsah hry ani přistupovat k chráněným datům. Veškerá hra i administrativní rozhraní komunikují se serverem prostřednictvím REST API, proto musí být endpointy pro zápis, úpravu i mazání dat adekvátně zabezpečeny.

6.1 Zabezpečení dat

Pro zabezpečení REST API bylo zvoleno ověřování na bázi JWT (JSON Web Token) a výchozí middleware ASP.NET Core pro autentizaci a autorizaci.

6.1.1 JWT Token

JWT token je v HTTP hlavičce `Authorization: Bearer <token>` zasílán při každém požadavku na chráněné endpointy. Token obsahuje:

- **Header** – identifikuje použitý algoritmus (HS256) a typ tokenu (JWT).
- **Payload** – obsahuje *claims*, například
 - sub (uživatelské ID),
 - role (uživatelská role: Admin, Editor),
 - exp (čas vypršení platnosti tokenu).
- **Signature** – HMAC SHA-256 podpis nad Base64Url(header) a Base64Url(payload) společně se sdíleným tajným klíčem. [37]

Platnost tokenu je omezena na limitovaný počet minut (konfigurováno na straně serveru). Po vypršení platnosti je vyžadováno nové přihlášení.

6.1.2 Serverová konfigurace

V souboru `Program.cs` byla autentizace a autorizace zprovozněna pomocí následující konfigurace:

```

1  builder.Services.AddAuthentication(x =>
2  {
3      x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
4      x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
5      x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
6  }).AddJwtBearer(x =>
7  {
8      var jwtKey = builder.Configuration["Jwt:Key"];
9      if (string.IsNullOrEmpty(jwtKey))
10     {
11         throw new ArgumentNullException(nameof(jwtKey),
12             "JWT secret key is not configured.");
13     }
14     x.TokenValidationParameters = new TokenValidationParameters
15     {
16         ValidIssuer = builder.Configuration["Jwt:Issuer"],
17         ValidAudience = builder.Configuration["Jwt:Audience"],
18         IssuerSigningKey = new SymmetricSecurityKey(
19             Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"])
20         ),
21         ValidateIssuer = true,
22         ValidateAudience = true,
23         ValidateLifetime = true,
24         ValidateIssuerSigningKey = true
25     };
26 });
27
28 builder.Services.AddAuthorization();
29

```

Listing 18 Autentizace a autorizace

Po této konfiguraci se požadavky na chráněné endpointy označí atributem `[Authorize]` nebo konkrétní rolí `[Authorize(Roles = "Admin,Editor")]`. [38]

6.1.3 Klientská konfigurace

Pro správné přidání autorizačního tokenu do HTTP požadavků ve SvelteKit byl využit soubor `hooks.server.ts`, který umožňuje globální manipulaci s příchozími požadavky a odchovými odpověďmi.

Pokud aplikace komunikuje s API vyžadujícím autentizaci pomocí tokenu uloženého v cookies, je nutné zajistit, aby byly cookies zahrnuty v každém požadavku. Toho bylo dosaženo nastavením možnosti `credentials: 'include'` při volání `fetch`.

Při každém načtení stránky – resp. při průchodu souborem `+layout.server.ts` – se volá endpoint `GetUser`. Ten na základě přiloženého JWT tokenu vrací informace o uživateli, zejména jeho přezdívkou a roli.

6.2 CORS

Pro bezpečnou komunikaci mezi frontendem (běžícím na `localhost:5173` nebo později reálné doméně) a backendem byl nakonfigurován CORS tak, aby byly povoleny pouze předem definované původy:

```
1     builder.Services.AddCors(options =>
2 {
3     options.AddDefaultPolicy(builder =>
4     {
5         builder.WithOrigins( "")
6             .AllowAnyHeader()
7             .AllowAnyMethod();
8     });
9 });
```

Listing 19 CORS

- **WithOrigins** – povoleny pouze konkrétní domény front-endu specifikované v uvozovkách.
- **AllowAnyHeader** – front-end může odesílat libovolné HTTP hlavičky (včetně ‘Authorization’).
- **AllowAnyMethod** – podporovány všechny HTTP metody (GET, POST, PUT, DELETE).
- **AllowCredentials** – povoleno zasílání cookies a autorizace přes hlavičku. [39]

Tímto postupem je zajištěno, že k API mají přístup pouze autorizovaní klienti a že každý pokus o zápis či mazání dat musí být proveden s platným JWT tokenem.

6.3 Politika hesla a její ověřování

Pro zajištění bezpečnosti uživatelských účtů byla definována minimální politika složitosti hesla. I když je registrace v současné verzi dostupná pouze pro administrátory, bylo nutné zajistit, aby zvolená hesla odpovídala stanoveným bezpečnostním kritériím. Politika je v aplikaci prosazena dvojím způsobem – na serveru i na klientovi.

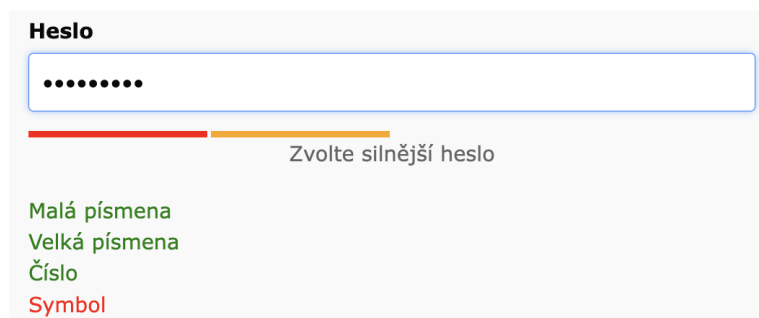
Na serverové straně je použita standardní konfigurace ASP.NET Identity, která vyžaduje:

- alespoň 10 znaků (`RequiredLength = 10`),
- minimálně jednu číslici (`RequireDigit = true`),
- velké a malé písmeno (`RequireUppercase` a `RequireLowercase`),
- speciální znak (`RequireNonAlphanumeric`).

```
1 builder.Services.Configure<IdentityOptions>(options =>
2 {
3     options.Password.RequireDigit = true;
4     options.Password.RequiredLength = 10;
5     options.Password.RequireNonAlphanumeric = true;
6     options.Password.RequireUppercase = true;
7     options.Password.RequireLowercase = true;
8     options.Password.RequiredUniqueChars = 1;
9     options.User.RequireUniqueEmail = true;
10 });
```

Listing 20 Konfigurace politiky hesla na serveru (ASP.NET Identity)

Současně je na klientské straně implementován modul průběžné kontroly síly hesla. Uživatel je během zadávání informován o tom, jaké podmínky ještě musí splnit (počty znaků, typy znaků). Tento okamžitý feedback pomáhá zabránit situacím, kdy by server heslo kvůli nedostatečné složitosti odmítl až při odeslání formuláře.



Heslo

.....

Zvolte silnější heslo

- Malá písmena
- Velká písmena
- Číslo
- Symbol

Obrázek 6.1 Průběžná kontrola síly hesla na klientovi (zdroj: vlastní)

Díky této kombinaci serverové politiky a klientské vizuální zpětné vazby je zajištěno, že do systému jsou přijímána pouze silná a bezpečná hesla.

7 MOŽNOSTI PUBLIKACE

Správa verzí obou částí (frontend i backend) byla realizována v repozitářích na GitHubu. Pro zajištění kontinuální integrace bylo využito GitHub Actions, kde je nakonfigurován workflow, které při každém „pushi“ do hlavní větve provádí:

- instalaci a ověření závislostí,
- kompilaci a build aplikace,
- v budoucnu i nasazení na cílové prostředí.

7.1 Backend (.NET Web API)

Pro nasazení serverové části byly identifikovány dvě hlavní možnosti:

7.1.1 Platforma jako služba (PaaS)

Výsledný build aplikace se jednoduše publikuje na hostitelskou PaaS platformu (např. Azure App Service, AWS Elastic Beanstalk nebo Google App Engine), která:

- zajistí automatické vytvoření a správu serverových instancí,
- poskytne předkonfigurované HTTPS certifikáty, monitoring a logování,
- umožní horizontální i vertikální škálování bez další konfigurace.

7.1.2 Kontejnerizace

Aplikace může být zabalena do Docker kontejneru a nasazena buď na vlastní VPS, nebo do orchestrátoru (např. Kubernetes), kde:

- je zajištěna konzistence prostředí napříč vývojem a produkcí,
- jsou automaticky prováděny rolling updates, health-checky a autoscaling,
- vyžaduje se však správa registrů, sítí a údržba clusteru. [40]

7.1.3 Vybrané řešení

Pro tento projekt byla zvolena varianta PaaS s AWS Elastic Beanstalk, která nabízí nejrychlejší spuštění a nejmenší režii správy infrastruktury. Toto řešení již bylo zkušebně implementováno a je funkční.

Krátký postup nasazení:

1. **Nastavení účtu:** Nejprve je potřeba v AWS konzoli vytvořit účet, aplikaci, přidat správná práva a napojit na databázi.
2. **Build:** `dotnet publish -c Release -o ./publish`
3. **Inicializace Elastic Beanstalk:** `eb init` (volba platformy .NET Core, region)
4. **Vytvoření a deploy:** `eb create <env>` následované `eb deploy`

Výsledná aplikace je zpřístupněna na veřejné URL, přičemž AWS zajišťuje provisioning, load balancing i automatické škálování. [41]

7.2 SvelteKit frontend

7.2.1 Režimy vykreslování frontendové aplikace

Frontendová aplikace založená na SvelteKit může být nasazena ve dvou základních režimech: jako čisté SPA (Single-Page Application) nebo s podporou SSR (Server-Side Rendering). Volba režimu závisí na konkrétních požadavcích na SEO, bezpečnost práce s cookie/jwt a typu routování. [42] [43]

Single-Page Application (SPA) V režimu SPA jsou všechny stránky předávány klientovi jako statické soubory a veškerá navigace i načítání dat probíhá na straně prohlížeče. Tento režim je ideální, pokud:

- velká část UI používá klient-side fetch v onMount,
- není vyžadována primární optimalizace pro vyhledávače (SEO),
- neprobíhá validace cookie nebo JWT na serveru před prvním vykreslením.

Konfigurace SPA v `svelte.config.js`:

```
1 import adapter from '@sveltejs/adapter-static';
2
3 export default {
4   kit: {
5     adapter: adapter({
6       fallback: 'index.html',
7       strict: false
8     })
9   }
10 };
```

Aplikace je pak nasazena jako statický web (Netlify, GitHub Pages, Vercel apod.), kde poskytovatel zajišťuje CDN, HTTPS a fallback na `index.html` pro neznámé cesty.

Server-Side Rendering (SSR) Ve SSR režimu je HTML generováno na serveru pro každý požadavek, což umožňuje:

- načítání a validaci uživatele v `+layout.server.ts` před prvním vykreslením,
- zabezpečené čtení cookie/JWT na serveru,
- podporu dynamických rout s přímým voláním server-side load funkcí nebo endpoints (např. `src/routes/admin/[...]`).

Pro SSR může být použit např. adapter Node.js:

```
1 import adapter from '@sveltejs/adapter-node';
2
3 export default {
4   kit: {
5     adapter: adapter()
6   }
7 };
```

Nasazení probíhá jako běžná Node aplikace na Vercel, Render, Heroku nebo jako edge-funkce, přičemž server zajišťuje jak statické assets, tak API endpointy. [44][42]

7.2.2 Vybrané řešení

Vzhledem k tomu, že je v `+layout.server.ts` prováděno čtení cookie a volání API pro načtení uživatele a navíc jsou požadovány dynamické administrační routy s server-side loadingem, je pro tuto implementaci vhodnější SSR režim. Zvolením např. `@sveltejs/adapter-vercel` (nebo `@sveltejs/adapter-node`) by měla být zajištěna bezpečná inicializace session, optimalizace pro SEO a možnost přímé obsluhy serverových endpointů z jednoho procesu.

8 BUDOUCÍ VÝVOJ

Aplikace byla prezentována zástupcům spolku, kteří ji přijali s velmi pozitivní zpětnou vazbou. V první fázi po zveřejnění bude provedeno interní testování, jehož cílem je odhalit a odstranit případné nedostatky. Zároveň bude doladěno UI pro různé platformy a velikosti obrazovek. Pokud nebudou zjištěny kritické chyby, bude testování rozšířeno na účastníky připravovaných workshopů, čímž se získá širší spektrum uživatelských poznatků. Na základě vyhodnocení sesbíraných podnětů a implementaci potřebných úprav bude aplikace uvolněna jako finální produkt.

Pro zlepšování herního zážitku je ve hře implementován mechanismus zpětné vazby, který umožňuje hráčům po dokončení úrovně ohodnotit obsah, obtížnost a typ úkolů. Tato data poslouží k úpravě scénářů, přidání nových herních prvků nebo snížení, případně zvýšení složitosti.

Z hlediska kvality kódu je plánováno doplnit jednotkové a integrační testy jak na straně serveru (xUnit, Moq pro služby a kontrolery), tak na straně klienta (Svelte Testing Library, případně Playwright pro end-to-end testy). CI/CD pipeline v GitHub Actions bude v takovém případě rozšířena o fázi spouštění testů, aby bylo možné včas odhalit chyby.

V případě zpřístupnění registrace i běžným uživatelům bude potřeba implementovat ověření e-mailové adresy zasláním potvrzovacího e-mailu, funkci změny hesla v profilu a obnovu hesla pomocí jednorázového tokenu.

Dalšími možnými rozšířeními jsou zavedení monitoringu a analýzy chování uživatelů a postupné rozšiřování herního obsahu o nové úrovně.

ZÁVĚR

V této práci se podařilo implementovat výukovou aplikaci zaměřenou na podporu mediální gramotnosti. Díky intenzivní spolupráci s neziskovou organizací Zvol si info, která se podílela na vývoji příběhu i odborném ověření obsahu, se podařilo vytvořit nástroj, který uživatelům zprostředkovává osvědčené metody ověřování zdrojů a informací online.

V teoretické části byly popsány a analyzovány prototypy, ze kterých byl vybrán prototyp vytvořený Viktorií Kolesárovou na fakultě mediální komunikace Univerzity Tomáše Bati. Následně práce představila přehled základních technologií webového vývoje – HTML, CSS a JavaScript – a popisovala moderní frameworky, z nichž pro frontend byl zvolen Svelte a pro backend ASP.NET Core Web API.

Praktická část přináší detailní popis návrhu a implementace aplikace. Jsou zde specifikovány herní mechanismy a čtyři výukové úrovně příběhu MarsGate, způsoby řízení herního toku, interakce s hráčem a integrace s API pro ukládání dat. Vývoj backendu zahrnoval návrh RESTful rozhraní a autentizačního modulu.

Aplikace Klamárium v současné fázi obsahuje úvodní modul MarsGate, který na základě analýzy falešného facebookového příspěvku seznamuje hráče s principy kritického myšlení. Před finálním nasazením bude nutné provést rozsáhlejší uživatelské testování a na základě získané zpětné vazby případně upravit herní mechaniky i obsah scénářů. Rovněž je v plánu rozšíření o další příběhové linie a implementace víceúrovňového monitoringu pokroku uživatele.

Věřím, že díky další spolupráci se spolkem Zvol si info a plánovanému pilotovému ověření v reálném provozu bude Klamárium publikováno jako edukační nástroj, který přispěje ke zvýšení mediální gramotnosti širší veřejnosti.

SEZNAM POUŽITÉ LITERATURY

- [1] INFO, Zvol, 2023. *Zvol si info*. Online. Dostupné z: <https://www.muni.cz/studenti/studentske-spolky/zvol-si-info>. Online. [cit. 2024-11-29].
- [2] ŠRAGA, Radoslav, 2021. *Zvol si info*. Online. Dostupné z: <https://showcase.fmk.utb.cz/digitalni-design/zvol-si-info/>. Online. [cit. 2024-11-29].
- [3] KORNASOVÁ, Veronika, 2021. *Design-formace/Conspirator*. Online. Dostupné z: <https://showcase.fmk.utb.cz/digitalni-design/design-formace-conspirator/>. Online. [cit. 2024-11-29].
- [4] BÁROVÁ, Klára, 2021. *Materiály pro výuku mediální gramotnosti*. Online. Dostupné z: <https://showcase.fmk.utb.cz/digitalni-design/materialy-pro-vyuku-medialni-gramotnosti/>. Online. [cit. 2024-11-29].
- [5] KOLESÁROVÁ, Viktória, 2021. *Zvol' si info—Deepfake*. Online. Dostupné z: <https://showcase.fmk.utb.cz/digitalni-design/zvol-si-info-deepfake/>. Online. [cit. 2024-11-29].
- [6] WEIS, Albert, 2021. *Perspekto – informační cvičení pro SŠ*. Online. Dostupné z: <https://showcase.fmk.utb.cz/digitalni-design/perspekto-informacni-cviceni-pro-ss/>. Online. [cit. 2024-11-29].
- [7] FERGUSON, Nicole, 2024. *What's the Difference Between Frontend vs Backend Web Development?* Online. Dostupné z: <https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/>. Online. [cit. 2025-04-27].
- [8] KOĐOUSKOVÁ, Barbora, 2024. *Co je to API a jaké jsou možnosti jeho využití?* Online. Dostupné z: <https://www.rascasone.com/cs/blog/co-je-api>. Online. [cit. 2025-04-27].
- [9] AWS, 2025. *What is an API (Application Programming Interface)?* Online. Dostupné z: <https://aws.amazon.com/what-is/api/>. Online. [cit. 2025-03-27].

- [10] KOŘOUSKOVÁ, Barbora, 2024. *Architektura MVC: definice, struktura, frameworky*. Online. Dostupné z: <https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky>. Online. [cit. 2025-04-27].
- [11] MOZILLA, 2025. *HTML: HyperText Markup Language*. Online. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTML>. Online. [cit. 2025-02-15].
- [12] MOZILLA, 2025. *CSS: Cascading Style Sheets*. Online. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/CSS>. Online. [cit. 2025-02-15].
- [13] MOZILLA, 2025. *JavaScript*. Online. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Online. [cit. 2025-02-15].
- [14] MOZILLA, 2025. *TypeScript*. Online. Dostupné z: <https://developer.mozilla.org/en-US/docs/Glossary/TypeScript>. Online. [cit. 2025-02-15].
- [15] W3SCHOOLS, 2024. *JavaScript HTML DOM*. Online. Dostupné z: https://www.w3schools.com/js/js_htmlDOM.asp. Online. [cit. 2024-11-29].
- [16] HARRIS, RICH, 2018. *Virtual DOM is pure overhead*. Online. Dostupné z: <https://svelte.dev/blog/virtual-dom-is-pure-overhead>. Online. [cit. 2025-03-27].
- [17] JAVASCRIPT, State, 2024. *Front-end Frameworks Ratios Over Time*. Online. Dostupné z: <https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/>. Online. [cit. 2025-03-15].
- [18] HÁMORI, Ferenc, 2024. *The History of React.js on a Timeline*. Online. Dostupné z: <https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/>. Online. [cit. 2025-03-15].
- [19] REACT, 2025. *Learn React*. Online. Dostupné z: <https://react.dev/learn>. Online. [cit. 2025-03-20].
- [20] SAINI, Kusum, 2025. *The History of React.js on a Timeline*. Online. Dostupné z: https://www.simplilearn.com/tutorials/reactjs-tutorial#what_is_react. Online. [cit. 2025-03-20].

- [21] ANGULAR, 2025. *What is Angular?* Online. Dostupné z: <https://angular.dev/overview>. Online. [cit. 2025-03-20].
- [22] ANGULAR, Made, 2025. *A showcase of web apps made with Angular*. Online. Dostupné z: <https://www.madewithangular.com/sites>. Online. [cit. 2025-03-27].
- [23] VUE, 2025. *What is Angular?* Online. Dostupné z: <https://vuejs.org/guide/introduction>. Online. [cit. 2025-03-20].
- [24] VUE, 2025. *What is Vue?* Online. Dostupné z: <https://vuejs.org/guide/introduction>. Online. [cit. 2025-03-20].
- [25] SVELTE, 2025. *web development for the rest of us*. Online. Dostupné z: <https://svelte.dev>. Online. [cit. 2025-03-20].
- [26] MOZILLA, 2025. *Getting started with Svelte*. Online. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Svelte_getting_started. Online. [cit. 2025-03-20].
- [27] GADHAVI, Maitray, 2024. *Most Popular Backend Frameworks for Web Development in 2025*. Online. Dostupné z: <https://radixweb.com/blog/best-backend-frameworks>. Online. [cit. 2025-03-20].
- [28] SHELDON, Robert, 2025. *Definition: model-view-controller (MVC)*. Online. Dostupné z: <https://www.techtarget.com/whatis/definition/model-view-controller-MVC>. Online. [cit. 2025-03-27].
- [29] GENTILE, Angelo, 2023. *Basics of Django: Model-View-Template (MVT) Architecture*. Online. Dostupné z: <https://angelogentileiii.medium.com/basics-of-django-model-view-template-mvt-architecture-8585aecffbf6>. Online. [cit. 2025-03-27].
- [30] POSTMAN, 2025. *What is an API?* Online. Dostupné z: <https://www.postman.com/what-is-an-api>. Online. [cit. 2025-03-27].
- [31] PYTHON, Wiki, 2025. *Web Frameworks for Python*. Online. Dostupné z: <https://wiki.python.org/moin/WebFrameworks>. Online. [cit. 2025-03-25].

- [32] MAPLE, Code, 2023. *Understanding Django MVT architecture and view functions*. Online. Dostupné z: <https://medium.com/@CodeMaple/understanding-django-mvt-architecture-and-view-functions-django-full-course-for-beginners-lesson-39c8da093b44>. Online. [cit. 2025-03-25].
- [33] LARAVEL, 2025. *The fastest way to start your next application*. Online. Dostupné z: <https://laravel.com/starter-kits>. Online. [cit. 2025-03-20].
- [34] LARAVEL, 2025. *Meet Laravel*. Online. Dostupné z: <https://laravel.com/docs/12.x#meet-laravel>. Online. [cit. 2025-03-20].
- [35] MICROSOFT, 2025. *What is .NET?* Online. Dostupné z: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>. Online. [cit. 2025-03-20].
- [36] SMITH, Steve, 2023. *Architecting modern web applications with ASP.NET Core and Microsoft Azure*. Online, PDF. Microsoft Developer Division, .NET, and Visual Studio product teams. Dostupné z: <https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf>.
- [37] JWT, 2025. *Introduction to JSON Web Tokens*. Online. Dostupné z: <https://jwt.io/introduction>. Online. [cit. 2025-04-27].
- [38] MICROSOFT, 2024. *Configure JWT bearer authentication in ASP.NET Core*. Online. Dostupné z: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication>. Online. [cit. 2025-04-27].
- [39] MICROSOFT, 2024. *Enable Cross-Origin Requests (CORS) in ASP.NET Core*. Online. Dostupné z: <https://learn.microsoft.com/en-us/aspnet/core/security/cors>. Online. [cit. 2025-04-27].
- [40] VOONG, Cain, 2021. *Dockerising .NET Core and Svelte*. Online. Dostupné z: <https://dev.to/cainux/net-core-and-svelte-f8o>. Online. [cit. 2025-04-27].
- [41] AWS, 2025. *Deploying .NET Windows applications with Elastic Beanstalk*. Online. Dostupné z: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_NET.html. Online. [cit. 2025-04-27].

- [42] SVELTE, 2024. *Building your app*. Online. Dostupné z: <https://svelte.dev/docs/kit/building-your-app>. Online. [cit. 2025-04-27].
- [43] OLLILA, Risto, 2021. *A performance comparison of rendering strategies in open source web frontend frameworks*. Online, diplomová práce. University of Helsinki. Dostupné z: https://helda.helsinki.fi/bitstream/10138/329793/2/Ollila_Risto_tutkielma_2021.pdf. [cit. 2025-04-27].
- [44] VERCEL, 2025. *SvelteKit on Vercel p.* Online. Dostupné z: <https://vercel.com/docs/frameworks/sveltekit>. Online. [cit. 2025-05-07].

SEZNAM OBRÁZKŮ

Obrázek 1.1	Prototyp Ranné správy	16
Obrázek 1.2	Prototyp Conspirator	16
Obrázek 1.3	Prototyp Materiály pro výuku mediální gramotnosti	17
Obrázek 1.4	Prototyp Deepfake	18
Obrázek 1.5	Prototyp Perspekt	19
Obrázek 2.1	Ukázka JavaScript	23
Obrázek 2.2	Ukázka TypeScript	23
Obrázek 2.3	DOM	24
Obrázek 2.4	Vývoj užívání frontendových frameworků	25
Obrázek 2.5	Vývoj pozitivního hodnocení frontendových frameworků	26
Obrázek 2.6	Ukázka kódu pomocí React	27
Obrázek 2.7	Ukázka kódu pomocí Vue	30
Obrázek 2.8	Ukázka kódu pomocí Svelte	31
Obrázek 2.9	MVT architektura v Django	34
Obrázek 3.1	Level 1: Facebookový příspěvek	40
Obrázek 3.2	Level 1: Kolo emocí	41
Obrázek 3.3	Level 1: Drag-and-drop rozhraní na pojmy	42
Obrázek 3.4	Level 2: Motivace příspěvku	43
Obrázek 3.5	Level 2: Motivace komentáře	43
Obrázek 3.6	Level 3: Můstkový kvíz	44
Obrázek 3.7	Level 3: Návod na používání Google Lens	45
Obrázek 3.8	Level 3: Úkol na používání Google Lens	46
Obrázek 3.9	Level 4: Úkol	47
Obrázek 3.10	Level 4: Přiřazení definic	48
Obrázek 3.11	Level 4: Přiřazení příkladů	49
Obrázek 3.12	Level 4: Chat s @realjan47	50
Obrázek 3.13	Level 4: Rozřazení tipů pro kritické myšlení	50
Obrázek 3.14	Level 5: Kvíz	51
Obrázek 3.15	Zpětná vazba - hodnocení	52
Obrázek 3.16	Zpětná vazba - role	52

Obrázek 3.17	Fun-fact vyskakovací okno	52
Obrázek 4.1	Blokové schéma.....	54
Obrázek 4.2	ER diagram.....	56
Obrázek 5.1	Klamárium - úvodní strana	63
Obrázek 5.2	Administrátorský pohled	68
Obrázek 5.3	Konkrétní příklad implementace drag-and-drop	69
Obrázek 6.1	Průběžná kontrola síly hesla na klientovi (zdroj: vlastní).....	75

SEZNAM TABULEK

Tabulka 1.1	Přehled prototypů	15
-------------	-------------------------	----

SEZNAM VÝPISŮ

1	Příklad Django.....	34
2	Perzistentní paměť	35
3	Ukázka ASP.NET Core.....	36
4	Definice entity Category.....	55
5	Definice kontextu DeepFakeDbContext	56
6	ICategoriesAppService.cs	57
7	CategoriesAppService.cs.....	58
8	CategoriesController.cs	59
9	routes/+page.svelte.....	63
10	VideoSection.svelte	64
11	Navigace mezi levely	65
12	Komunikace s API ve Svelte.....	66
13	Datová rozhraní	66
14	Perzistentní paměť - definice	67
15	Perzistentní paměť	67
16	Drag-and-drop implementace	69
17	Drag-and-drop zobrazení.....	71
18	Autentizace a autorizace	73
19	CORS	74
20	Konfigurace politiky hesla na serveru (ASP.NET Identity)	75

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AI	umělá inteligence
API	Application Programming Interface – aplikační programové rozhraní
ASP.NET	Active Server Pages .NET (platforma pro vývoj webových aplikací)
AJAX	Asynchronous JavaScript and XML – technika pro asynchronní načítání dat
CSS	Cascading Style Sheets – kaskádové styly pro definici vzhledu webu
CORS	Cross-Origin Resource Sharing – pravidla pro sdílení zdrojů mezi doménami
CRUD	Create, Read, Update, Delete – základní operace nad daty
DI	Dependency Injection – princip vkládání závislostí do tříd
DOM	Document Object Model – model dokumentu pro manipulaci s HTML
DRY	Don't Repeat Yourself – princip neopakování kódu
EF Core	Entity Framework Core – ORM framework pro .NET
HTML	HyperText Markup Language – značkovací jazyk pro webové stránky
HTTP	HyperText Transfer Protocol – protokol pro přenos webových stránek
HTTPS	HTTP Secure – zabezpečený HTTP
JWT	JSON Web Token – standard pro přenos autentizačních údajů
JSON	JavaScript Object Notation – formát pro výměnu dat
MVC	Model-View-Controller – architektonický vzor rozdělení aplikace do tří vrstev
MVT	Model-View-Template – architektonický vzor používaný v Django
PaaS	Platform as a Service – cloudový model, poskytovatel spravuje infrastrukturu i runtime
REST	REpresentational State Transfer – architektura pro návrh HTTP API
SSR	Server-Side Rendering – vykreslování stránek na serveru
SPA	Single-Page Application – aplikace běžící jako jedna stránka
UI	User Interface – uživatelské rozhraní
UX	User Experience – uživatelská zkušenost
VDOM	Virtuální DOM

SEZNAM PŘÍLOH

PŘÍLOHA P I: Frontend aplikace Klamárium

PŘÍLOHA P II: Backend aplikace Klamárium

PŘÍLOHA P I: FRONTEND APLIKACE KLAMÁRIUM

Složka *Deepfake_frontend* obsahuje Svelte projekt s implementací klientské části aplikace.

,

PŘÍLOHA P II: BACKEND APLIKACE KLAMÁRIUM

Složka *Deepfake_backend* obsahuje ASP.NET Web API projekt s implementací serverové části aplikace.