Využití technologie Angular2 při vývoji webových aplikací

Bc. Juraj Štefan

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Juraj Štefan**
Osobní číslo:	**A14450**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**prezenční**
Téma práce:	**Využití technologie Angular2 při vývoji webových aplikací**
Téma anglicky:	**The Use of Angular2 in Web Application Development**

Zásady pro vypracování:

1. **Nastudujte a v teoretické části popište základní technologie používané při vývoji webových aplikací pomocí tzv. MEAN Stack. Zaměřte se zejména na vývojový framework Angular2 a jazyk TypeScript.**
2. **Stručně srovnejte MEAN Stack přístup s přístupem označovaným jako LAMP Stack.**
3. **Zdokumentujte požadavky zadavatele na webovou aplikaci a vytvořte návrh architektury.**
4. **Pomocí technologie Angular2 a jazyka TypeScript implementujte webovou aplikaci dle požadavků a klíčové části popište v rámci praktické části.**
5. **Zhodnoťte použití zvolených technologií z hlediska požadavků na vývojový tým, rychlosti implementace, udržitelnosti a rozšiřitelnosti projektu.**
6. **Popište bezpečnostní aspekty aplikace.**

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **DICKEY, Jeff. Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node.js. San Francisco: Peachit Press, 2015. Develop and design. ISBN 01-339-3015-7.**
2. **MURRAY, Nate, Ari LERNER, Felipe COURY a Carlos TABORDA. Ng-Book 2: The Complete Book on Angular 2. Fullstack.io, 2016. ISBN 978-0991344611.**
3. **ROZENTALS, Nathan. Mastering TypeScript. Packt Publishing, 2015. ISBN 978-1784399665.**
4. **ELROM, Elad. Pro mean stack development. Apress, 2016. ISBN 978-148-4220-436.**
5. **FAIN, Yakov a Anton MOISEEV. Angular 2 Development with TypeScript. Manning Publications, 2016. ISBN 9781617293122.**
6. **MCLAUGHLIN, Brett. PHP & MySQL the missing manual. Sebastopol,CA: O'Reilly Media, 2012. ISBN 9781449355548.**

Vedoucí diplomové práce:	**Ing. Radek Vala, Ph.D.** Ústav informatiky a umělé inteligence
Datum zadání diplomové práce:	**3. února 2017**
Termín odevzdání diplomové práce:	**16. května 2017**

Ve Zlíně dne 3. února 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Jméno, příjmení: JURAJ ŠTEFAN

Název bakalářské/diplomové práce: Využití technologie Angular 2 při vývoji webových aplikací

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/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;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/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 diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.05.2017

.....
podpis diplomanta

ABSTRAKT

Táto diplomová práca sa zaoberá návrhom a vývojom webovej aplikácie použitím prístupu MEAN stack. Aplikácia demonštruje hlavné možnosti vývoja aplikácií využívajúcich framework Angular 2.

V rámci teoretickej časti je popisovaný prístup MEAN stack, ako aj jeho jednotlivé časti, so zameraním sa hlavne na framework Angular 2. Na konci teoretickej časti je stručný popis prístupu LAMP stack ako aj jeho zrovnanie s balíčkom MEAN stack.

Praktická časť dokumentuje požiadavky zadávateľa a obsahuje popis architektúry aplikácie ako aj jednotlivých časti aplikácie. Ďalej v rámci praktickej časti je zhodnotené použitie zvolených technológii z hľadiska požiadaviek na vývojový tým, rýchlosť implementácie, udržateľnosť a rozšíriteľnosť.

Klíčová slova: MEAN stack, MongoDB, Express.js, Angular, LAMP stack, webová aplikácia, single-page aplikácia

ABSTRACT

This diploma thesis discusses design and development of the web application using MEAN stack. The application demonstrates the main possibilities of development of applications which use the Angular 2 framework.

Within the theoretical part is described MEAN stack approach as well as its individual parts, focusing mainly on framework Angular 2. At the end of the theoretical part is a brief description of LAMP stack approach as well as its comparison with the MEAN stack package.

The practical part illustrates the requirements of customer and contains a description of the application architecture as well as the individual parts of the application. Furthermore, the practical part reviews the use of selected technologies in terms of requirements on a development team, speed of implementation, sustainability and extensibility.

Keywords: MEAN stack, MongoDB, Express.js, Angular, LAMP stack, web application, single-page application

Poďakovanie

Veľkú vďaku si zaslúži pán Ing. Radek Vala, Ph.D. za všetky rady a pripomienky, ktoré mi boli nápomocné pri vypracovaní tejto práce. Poďakovanie patrí aj mojej rodine a priateľom, ktorí mi boli celý čas oporou.

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

OBSAH

ÚVOD

V dnešnej dobe zažívajú single-page webové aplikácie vysokú popularitu. Z tohto dôvodu je vyvíjaných viacero JavaScript frameworkou pre tvorbu webových aplikácií tohto typu. Medzi najznámejšie patria ReactJS, Vue.js, Ember.js, Meteor.js a v neposlednom rade Angular. V septembri roku 2016 bola predstavená nová verzia posledného spomínaného frameworku, presnejšie verzia 2. A práve Angularom vo verzii 2 sa zaoberá táto diplomová práca, keďže táto verzia priniesla mnohé zásadné zmeny oproti pôvodnému AngularJS, čim bolo spôsobné že spätná kompatibilita nebola možná. Framework Angular je tiež súčasťou prístupu MEAN stack, ktorý je používaný pre vývoj webových aplikácie.

Cieľom práce je vývoj ukážkovej aplikácie, ktorá demonštruje použitie frameworku Angular2, ako aj z časti využite technológií v balíku MEAN stack. Práca rozoberá prístup MEAN stack, ktorého hlavné časti MongoDB, Express.js, Angular a Node.js sú tiež opísané v rámci práce. Podrobnejšie je rozoberaný Angular2, použitý jazyk TypeScript v tomto frameworku, jeho architektúra a používanie konzoly Angular-CLI, ktorá zjednodušuje vývoj. V práci je spomínaný aj prístup LAMP stack a jeho zrovnanie s vyššie spomínaným balíčkom MEAN stack. V rámci práce je podrobne skúmaná ukážková aplikácia. Popísané sú tu najprv funkčné a nefunkčné požiadavky zadávateľa na danú aplikáciu, ďalej návrh architektúry a popis hlavných častí aplikácie. Ďalej je tu kapitola zameraná na zhodnotenie technológií, ktoré boli použité v rámci aplikácie, z viacerých hľadísk. Medzi tieto hľadiská patria požiadavky na vývojový tým, rýchlosť implementácie, udržateľnosti a rozšíriteľnosti celého projektu. V poslednej kapitole sú prezentované bezpečnostné aspekty aplikácie.

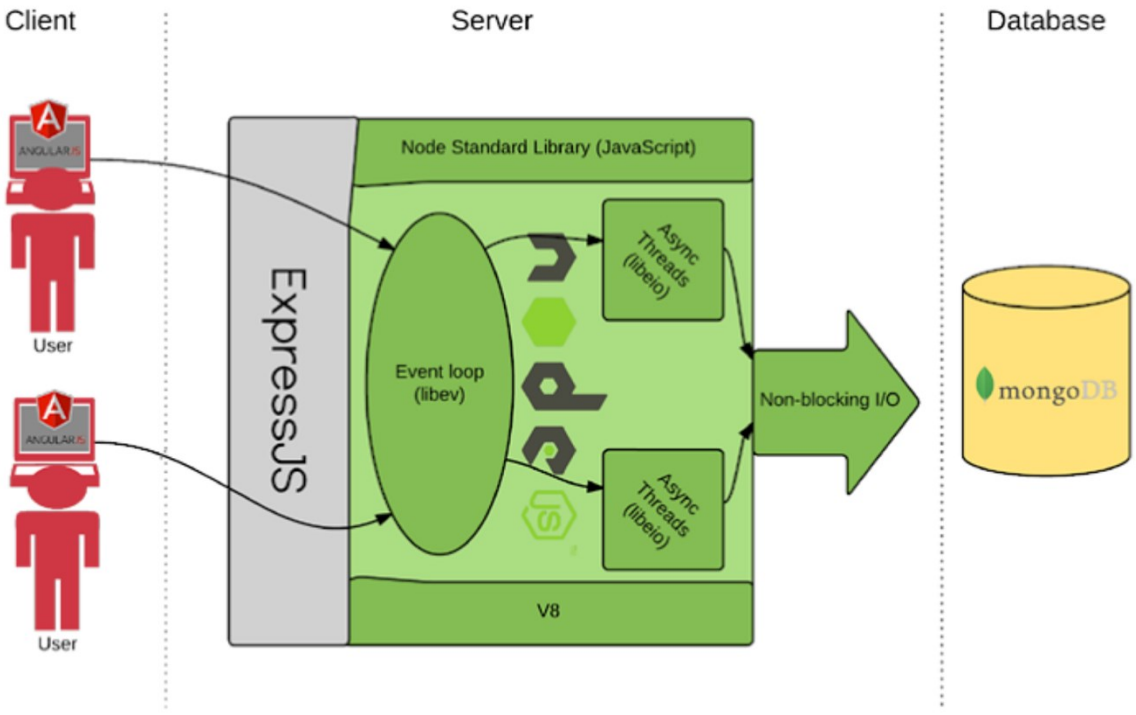
I. TEORETICKÁ ČÁST

1 MEAN STACK

Pre zjednodušenie programovania je navrhnutých viacero softvérových riešení vo forme stack-u alebo inak povedané zásobníku, prípadne balíčku. Softvérový zásobník môže byť opísaný ako balíček softvéru, ktorý spolupracuje na vytvorení jedného plne funkčného softvérového riešenia, v tomto prípade webovej aplikácie. Táto kapitola sa bude zaoberať balíčkom MEAN a jeho časťami.

Názov MEAN je vlastne akronym skomponovaný zo štyroch komponentov, ktorými sú MongoDB ako databáza, Express ako framework webového servera, Angular ako framework pre webového klienta a Node.js ako serverová platforma. V MEAN stacku sa používa iba jeden hlavný programovací jazyk na strane klienta, ako aj na strane servera a týmto jazykom je jazyk JavaScript. [1]

Architektúra MEAN stack-u je ukázaná na obrázku (*Obr. 1*) a funguje na jednoduchom princípe. Keď Node server príjme požiadavku od klienta, Express najskôr spracuje túto požiadavku. Express beží nad platformou Node a pripája sa k databáze MongoDB podľa potreby. Nakoniec žiadosť klienta odpovedá aplikáciou Angular. Aby nedochádzalo k neustálemu aktualizovaniu sa celých stránok, Angular je zodpovedný za okamžité vykresľovanie nových údajov. [2]



Obr. 1 MEAN stack komponenty [2]

1.1 MongoDB

MongoDB je open-source multiplatformový, dokumentovo orientovaný databázový systém. Namiesto tabuliek MangoDB používa dokumenty, ktoré majú schému podobnú JSON. [11] Tieto dokumenty sú seba-popisujúce, hierarchické stromové dátové štruktúry, ktoré môžu pozostávať z máp, zbierok a skalárnych hodnôt. Uložené dokumenty sú navzájom podobné, ale nemusia byť úplne rovnaké. Dáta dokumentov v databáze sú ukladané na základe vzťahu kľúč-hodnota. [3]

1.1.1 Hlavné vlastnosti

Hlavnou úlohou MongoDB je prioritne slúžiť ako databáza, no okrem vytvárania, čítania, aktualizácie a odstraňovania údajov poskytuje MongoDB neustále rastúci zoznam unikátnych funkcií:

Ad-hoc dotazy

MongoDB podporuje hľadanie podľa poľa, rozsahové hľadanie a hľadanie podľa regulárnych výrazov. Dotaz tiež môže obsahovať používateľsky definované JavaScriptové funkcie. [4]

Indexovanie

MongoDB podporuje generické sekundárne indexy, ktoré umožňujú rôzne rýchle dotazy (queries) a poskytujú jedinečné, zložité, geopriestorové a fulltextové indexovanie. [3]

Replikácia

MongoDB poskytuje vysokú dostupnosť pomocou sady replík. Sada replík obsahuje dve alebo viac kópií údajov. Každý člen replík môže vždy zastávať funkciu primárnej alebo sekundárnej repliky. Primárna replika implicitne vykonáva všetky čítacie a zapisovacie operácie. Sekundárne repliky udržiavajú kópiu dát primárnych replík s využitím vstavanej replikácie. Keď primárna replika zlyhá, sada replík automaticky vykoná proces výberu náhrady zo sekundárnych replík. Sekundárne repliky môžu tiež vykonávať čítaciu operáciu, ale údaje sú nakoniec vždy konzistentné. [5]

Agregácia

MongoDB podporuje "agregačnú pipeline", ktorá umožňuje vytvárať komplexné agregácie z jednoduchých dokumentov. [3]

Vyvažovanie záťaže

MongoDB je možné škálovať horizontálne pomocou shardingu. [5] Používateľ si vyberie shard kľúč, ktorý určuje spôsob distribúcie údajov v kolekcii. Údaje sú rozdelené na rozsahy (na základe shard kľúča) a sú rozdelené medzi viaceré shardy. (Shard je master s jedným alebo viacerými slaves.). MongoDB môže bežať na viacerých serveroch, vyvažovať zaťaženie alebo duplikovať dáta, aby systém v prípade zlyhania hardvéru zostal v prevádzke. [4]

Špeciálne typy kolekcii

MongoDB podporuje time-to-live kolekcie pre dáta, ktoré by mali expirovať v určitom čase, napríklad sessions. Podporuje tiež kolekcie s pevnou veľkosťou, ktoré sú užitočné na uchovávanie nedávnych údajov, ako sú logovanie. [3]

Ukladanie súborov

MongoDB taktiež podporuje ľahko použiteľný protokol na ukladanie veľkých súborov a súborových metadát.

Niektoré funkcie, ktoré sú bežné pre relačné databázy, nie sú v MongoDB prítomné, predovšetkým JOIN a komplexné viacriadkové transakcie. Vynechanie týchto rozhodnutí bolo architektonickým rozhodnutím umožňujúcim väčšiu škálovateľnosť, keďže obidve tieto funkcie je ťažké efektívne poskytovať v distribuovanom systéme. [3]

1.1.2 Dokumenty MongoDB

Ako je vidieť z JSON kódu nižšie (*Zdrojový kód 1*), je jasné, že dokumenty v MongoDB nie sú vždy rovnaké. V dokumentových databázach je povolené mať rozdiely v atribútoch v porovnaní s databázou relačnou, kde každý riadok v tabuľke musí sledovať rovnakú schému. Schéma dokumentov v MongoDB sa môže líšiť a napriek tomu, tieto dokumenty stále môžu patriť do tej istej kolekcie.

```
1. [
2.   {
3.     "_id": "1",
4.     "name": "Karl",
5.     "children": [
6.       "Susan, Lisa"
7.     ],
8.     "birthplace": "London",
9.     "working experiences": [
```

```
10.         {
11.           "company": "Zlin Industries",
12.           "position": "Scientist"
13.         },
14.         {
15.           "company": "ZlinCorp",
16.           "position": "cleankeeper"
17.         }
18.       ]
19.     },
20.     {
21.       "_id": "2",
22.       "name": "Lucy",
23.       "birthName": "Liu",
24.       "birthplace": "Beijing"
25.     }
26.   ]
```

Zdrojový kód 1 MongoDB dokument

Ďalej v kóde vidíme, že dokumenty môžu obsahovať ako hodnotu nielen reťazce, čísla a iné typy ale aj zoznam hodnôt, ako v tomto prípade `children` alebo ďalšie dokumenty a dokonca aj zoznam dokumentov `workingExperiences`. Dokumenty majú podobné atribúty ako `_id`, `name` a `birthPlace`, ale majú aj rôzne atribúty, napríklad `children` v prvom dokumente a `birthName` v druhom dokumente. Toto nie je povolené v relačných databázach, kde je potrebné definovať hodnotu v každom stĺpci alebo ju nastaviť ako prázdnu alebo nulovú.

1.2 Express

Express.js je súčasťou backen-u MEAN stack-u. Je to flexibilný framework založený na Express môže pomôcť zjednodušiť a urýchliť vývoj samostatného Node.js servera. Jedná sa o web-framework, postavený na moduloch Node.js http a Connect, nazývané dokopy middleware. Sú základným kameňom filozofie configuration over convention, čo znamená že vývojári môžu použiť akékoľvek knižnice, ktoré potrebujú pre určitý projekt, čo im poskytuje flexibilitu a vysokú prispôsobivosť. [6]

Ak vývojár používa len hlavné moduly Node.js, častokrát sa stretáva so situáciou, kedy opakovane píše rovnaký kód pre podobne úlohy, ako napríklad:

- Parsovanie tela HTTP požiadavku
- Parsovanie cookies
- Správa sessions
- Routing s mnohými if podmienkami závislých na URL cestách a HTTP metódach HTTP požiadavku

- Určenie vhodnej hlavičky odpoveď na základe dátového typu
- Vytiahnutie parametrov z URL

Express.js rieši tieto a mnohé ďalšie problémy. Poskytuje tiež štruktúru podobnú MVC pre webové aplikácie. Tieto aplikácie sa môžu byť v rozsahu primitívnych back-endových REST API až po plnohodnotné vysoko škálovateľné webové aplikácie bežiace v reálnom čase .[7]

Express pozostáva z troch často používaných hlavných objektov. Objekt aplikácie (application object) je inštancia aplikácie Express a zvyčajne sa používa na konfiguráciu aplikácie. Objekt požiadavky (request object) je wrapper HTTP požiadavku a slúži na extrahovanie informácií o aktuálne spracovanej žiadosti HTTP. Objekt odozvy (response object) je wrapper objektu HTTP odozvy Node a slúži na spracovanie žiadosť odoslanej na server a bude na tieto žiadosti odpovedať použitím metód. [8] .

1.2.1 Ako Express.js funguje

Express.js je npm modul, tým pádom sa inštaluje rovnako ako každý iný npm modul. [11] Ak je Express.js aplikácia v jednom súbore, napríklad server.js a bude spustená pomocou `node` príkazu v konzole , musí byť v tomto súbore vyžiadaný a nakonfigurovaný Express.js. Tento súbor zväčša musí obsahovať:

1. Závislosť tretích strán, ako aj vlastné moduly, ako sú napríklad kontrolery (controllers).

2. Inštancie objektu Express.js

3. Pripojenie sa k databáze ako napríklad MongoDB

4. Konfigurácie aplikácie Express.js, ako napríklad template engine

5. Definícia middleware, ako sú obsluha chyby (error handler), priečinky statických súborov, cookies súbory a iné obslužné nástroje

6. Definícia ciest (routes) a ich obsluhy

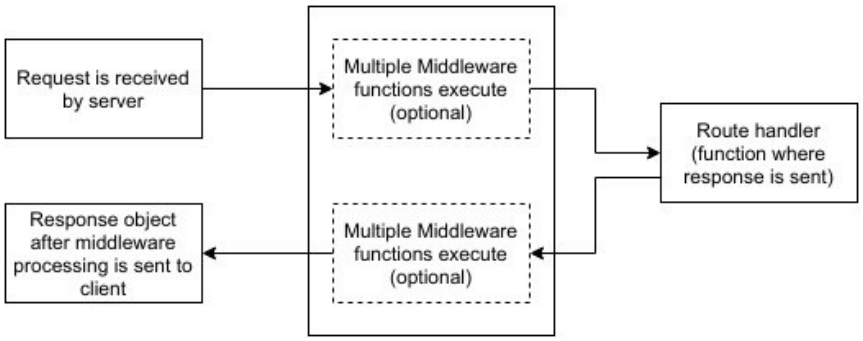
7. Štart aplikácie, ktorý spustí server na konkrétnom hostovi a porte [6]

Počas behu aplikácie Express.js sleduje žiadosti. Každá prichádzajúca žiadosť sa spracováva podľa definovaného middleware reťazca od hora dole. Toto umožňuje kontrolovať

vykonávané kroky. Napríklad, môžeme mať viaceré funkcie na spracovanie každej žiadosti. Niektoré z týchto funkcií budú uprostred (odtiaľ názov middleware):

1. Parsovanie cookie súborov a prechod na ďalší krok po dokončení
2. Parsovanie parametrov z adresy URL a pokračovanie v ďalšom kroku
3. Získanie informácie z databázy na základe hodnoty parametru
4. Zobrazenie dát a ukončenie odpovede. [7]

Middleware funkcie sú funkcie, ktoré majú prístup k objektom požiadavky, objektom odpovede a k ďalším middleware funkciám v aplikačnom request-response cykle. Tieto funkcie sú používane na modifikovanie týchto objektov pre účely parsovania požiadaviek, pridávanie hlavičiek odpovedí atď. Ako vidieť na obrázku používajú sa pred a po spracovaní požiadavku. [9]



Obr. 2 Middleware [9]

1.3 Angular

AngularJS je JavaScriptový štrukturálny framework pre vývoj dynamických webových aplikácií. Prvý krát ho predstavili Miško Hevery a Adam Abrons v roku 2009. Teraz je projekt Angular pod záštitou Google. Umožňuje použitie HTML ako šablónovacieho jazyka pre statické dokumenty. [10]

1.3.1 História Angular

Vývojári v minulosti používali HTML kód pre vytváranie statických dokumentov. No zatiaľ čo pre tento účel bol dostačujúci, pre účel dynamických stránok bol nevyhovujúci. Aj keď už existovali frameworky ktoré riešili nedostatky HTML spojením HTML, CSS a JavaScriptu alebo príkazovým spôsobom na manipulovanie s DOM. Tieto riešenia pra-

covali v určitom rozsahu ale neriešili hlavný problém HTML a teda, že nie je určený na zobrazenie dynamického obsahu. [39]

Z tohto dôvodu sa viacerí vývojári snažili vytvoriť riešenie, ktoré by riešilo tento problém. Jedným s nich boli aj Miško Hevery, ktorý v roku 2009 vyvinul technológiu AngularJS. Vo svojom blogu uvádza, že dôvodom prečo začal pracovať na Angular bolo, že väčšina webových aplikácii boli len UI nad nejakou databázou CRUD(create, read, upadte, delete). Cieľom <angular/>, ako sa pôvodne projekt menoval, bolo spraviť CRUD aplikácie jednoduchšími na vývoj. Jeho hlavným cieľom nebol generický framework, kde sa dá vytvoriť akýkoľvek typ aplikácie, ale je hlavným zameraním bol vývoj CRUD webových aplikácií. Veril, že mnoho aplikácií nebolo vyvinutých pretože by ich vývoj bol časovo veľmi náročný a s Angular by bola táto časová náročnosť efektívne redukovaná. [12]
K Heverymu sa pridal aj Adam Abrons a spolu pracovali na frameworku AngularJS a vydali ho pod open-source licenciou. Myšlienkou bolo spopularizovať vývoj single-page aplikácií. V tom čase obaja vývojári Hevery aj Abrons pracovali pre spoločnosť Google, z tohto dôvodu je projekt AngularJS ďalej vyvíjaný pod záštitou tejto spoločnosti.
V septembri roku 2014 bol na konferencii ng-Europe predstavený Angular 2. Mnoho vývojárov nebolo spokojných s absenciou akejkoľvek cesty migrácie z Angular 1.x na 2.0. Angular priniesol novú syntax, architektúru a odstránil niektoré vlastnosti, ktoré poskytoval Angular 1.x. [13]
Ako doposiaľ posledná verzia bol predstavený Angular 4, ktorý preskočil verziu 3 aby nedošlo k mýleniu z dôvodu router balíčkov, ktoré boli vo verzii 3.x. Angular 4 je spätne kompatibilný s verziou Angular 2 a prináša Ahead-of-time kompiláciu, teda kompiluje šablóny počas buildovania a generuje JavaScript kód. Ďalšími zmenami sú pridanie else do ngIf direktívy, vytvorenie vlastného balíčku pre animácie a ďalšie iné. [14]

1.3.2 AngularJS

AngularJS je frontend JavaScript framework navrhnutý pre vývoj single-page aplikácií (SPA)použitím MVC architektúry. SPA je web aplikácia, ktorá je raz načítaná a ďalej nevyžaduje obnovenie celej stránky, keď používateľ vykoná nejakú akciu na tejto stránke. [15] Úlohou AngularJS je rozšíriť funkcionalitu HTML použitím špeciálnych atribútov ktoré bindujú JavaScript business logiku s HTML elementami. Schopnosť AngularJS rozšíriť HTML dovoľuje čistejšiu DOM manipuláciu cez clien-side šablónovanie a two-way data binding, ktoré bezproblémovo synchronizuje model a výstup na obrazovke (view). Ďalej AngularJS vylepšuje štruktúru kódu aplikácie a testovateľnosť použitím MVC

a vkladaním závislostí (dependency injection). Aj keď začiatky s AngularJS sú ľahké, písanie väčších aplikácie je komplexnejšie, čo vyžaduje široké chápanie kľúčového konceptu frameworku. [1]

1.3.3 Angular 2

Počas vývoja AngularJS bola predstavená nová verzia Angular 2, ktorá priniesla zásadné zmeny. Medzi tieto zmeny patria prechod na TypeScript a stretávame sa tu ECMASript 6 špecifikáciou. Angular 2 je v porovnaní s Angular 1.x kompletne založený na komponentoch a je viacej objektovo orientovaný. Ďalšie funkcionality ako scopes a controllery sú úplne zrušené. [16]

Ďalšie zmeny Angular 2 oproti Angular 1.x:

- Je 2 až 5 krát rýchlejší ako jeho predchodca
- Mobile-first prístup
- Zmena štruktúry a syntaxi
- Zmena dependency injection
- Viacero možností voľby programovacieho jazyka – Typescript, Dart alebo JavaScript. [16]

1.4 Node.js

Node.js je platforma postavená na jadre JavaScriptového V8 engine-u pre jednoduché vytváranie rýchlych a škálovateľných sieťových aplikácií. Node.js využíva model I/O s neblokujúcim prístupom založeným na udalostiach. Node.js aplikácie sú písané v jazyku JavaScript a môžu byť spustené na operačných systémoch OS X, Microsoft Windows a Linux. Node.js tiež poskytuje bohatú knižnicu rôznych JavaScript modulov, ktorá zjednodušuje vývoj webových aplikácií pomocou programu Node.js do značnej miery.

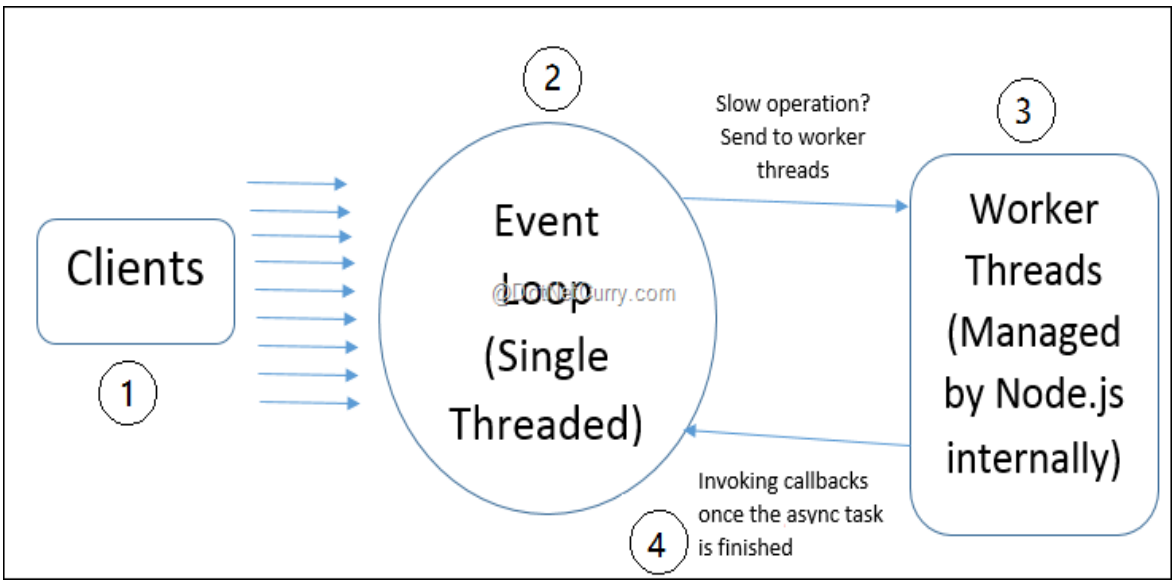
Node.js bol založený Ryanom Dahlom a prvýkrát bol ohlásený v roku 2009. Jedným z prvých problémov, ktoré sa Ryan Dahl pokúšať vyriešiť, bolo ako informovať prehliadač o tom, koľko času zostáva do konca procesu, ktorý nahráva používateľský súbor na serveri. Druhým problémom bolo, ako správne spracovať viacero žiadostí naraz. V tom čase spoločnosť Google predstavila V8 JavaScript engine, ako súčasť prehliadača Google Chrome. Jazyk JavaScript je interpretovaný jazyk a z tohto dôvodu potrebuje nejaký interpreter, ktorý kompiluje JavaScript do natívneho strojového kódu. Jedným z nich je aj V8 engine, ktorý je efektívne napísaný v jazyku C ++. V8 engine dokázal, že JavaScript by mohol

bežať oveľa rýchlejšie ako predtým. Vysoká rýchlosť interpretera V8 je možná najmä vďaka tomu, že je schopný skompilovať JavaScript kód na natívny strojový kód pred jeho vykonaním. Táto optimalizácia spôsobila, že JavaScript sa stal životaschopným programovacím jazykom. Dahl si toho všimol a vzal V8 engine a rozšíril ho o kód , ktorý mal podporu TCP / IP a prístupu k súborovému systému, keďže JavaScript nebol predtým navrhnutý na spustenie na strane servera. [17]

1.4.1 Princíp Node.js

JavaScript je programovací jazyk riadený udalosťami, čo znamená, že kód reaguje na konkrétne udalosti. Node.js plne využíva túto funkcionalitu, ktorá umožňuje naprogramovať jednoduché spustenie asynchrónneho kódu bez zablokovania zostávajúceho programu, čo v príklade môže znamenať, že nie je potrebné čakať na dokončenie databázového procesu, aby sa mohla vykonávať iná práca. [18]

Node.js používa cyklus udalostí, ktorý mu dáva možnosť zvládnuť súbežné požiadavky, pričom stále beží jedno-vláknovo. Takže má len jedno vlákno, ktoré je schopné postarať sa o veľký počet požiadaviek paralelne a reagovať na tieto požiadavky. [19]



Obr. 3 Cyklus udalostí [19]

Nasledujúci zoznam opisuje cyklus požiadavky a odpovede, ktorý je znázornený na obrázku (*Obr. 3*) vyššie:

1. Klienti pošle požiadavku na server Node.js.

2. Požiadavky sú prebraté v slučke udalostí alebo môžu byť požiadané, aby čakali v zásobníku, až kým cyklus udalostí sa neuvoľní.

3. Ak požadovaná klientska operácia musí byť vykonaná asynchrónne, úloha je delegovaná na pracovné vlákna a slučka udalostí vyberie ďalšiu úlohu zo zásobníku.

4. Keď pracujúce vlákno ukončí svoju úlohu, pošle oznámenie o tom do slučky udalostí spätným volaním a prenesie výsledok operácie na túto funkciu. [19]

1.4.2 Správca balíkov v Node.js

Ako býva zvykom, ku programovaciemu jazyku patrí aj nejaký balíkový systém. Node.js je spojený so správcom balíkov pre jazyk JavaScript (npm). Npm bol vytvorený rovnako ako Node.js, v roku 2009 za cieľom zjednodušiť zdieľanie kódu pre vývojárov vo forme modulov alebo balíkov. Npm prináša spôsob, ako znova použiť kód od iných vývojárov a tiež spôsob, ako s nimi zdieľať kód, a umožňuje jednoduché spravovanie verzií kódu. Ďalej si npm možno predstaviť aj ako register balíčkov, v ktorom je možné prehliadať, sťahovať a spravovať moduly tretích strán. [19]

Modul alebo balíček je priečinok s jedným alebo viacerými súbormi, ktorý obsahuje aj súbor nazvaný "package.json" s niektorými metadátami o tomto balíku, ako napríklad informácie o názve, verzii, spôsobe spustenia a tiež o závislostiach na rôznych balíkoch. Typická aplikácia, napríklad webová stránka, bude závisieť od desiatok alebo stoviek balíkov. Tieto balíky sú často malé. Všeobecnou myšlienkou je, že vytvoríte malý stavebný blok, ktorý rieši jeden problém a rieši ho dobre. To vám umožní vytvoriť vlastné riešenia z týchto malých, zdieľaných stavebných blokov. [20]

2 ANGULAR S POUŽITÍM JAZYKA TYPESCRIPT

Nasledujúca kapitola sa zaoberá architektúrou frameworku Angular2, v ktorej sa nachádzajú ukážky kódov. Tie sú v jazyku TypeScript, v ktorom sú Angular2 aplikácie písané.

2.1 TypeScript

Jazyk JavaScript je jazyk, ktorý je síce ľahký na naučenie, ale predstavuje problémy pri písaní veľkých a komplexných programov. Ako interpretovaný jazyk nemá JavaScript žiadny kompilačný krok, a tak je vykonávaný za behu. Pre programátorov, ktorí sú zvyknutý na písanie kódu v tradičnejšej forme pomocou kompilátorov, silného typovania a zaužívaných programovacích vzorov, môže JavaScript byť úplne cudzie prostredie.

TypeScript sa snaží potlačiť tieto úskalia. Je to silno typovaný, objektovo orientovaný, kompilovaný jazyk, ktorý programátorom umožňuje používanie konceptov a myšlienok zavedených v objektovo orientovaných jazykoch v jazyku JavaScript. TypeScript kompilátor generuje JavaScript, ktorý dodržiava tieto silno typové, objektovo orientované princípy - ale súčasne je to len čistý JavaScript. TypeScript bude fungovať spoľahlivo, kdekoľvek, kde môže byť spustený jazyk JavaScript, teda v prehliadači, na serveri alebo na mobilných zariadeniach.

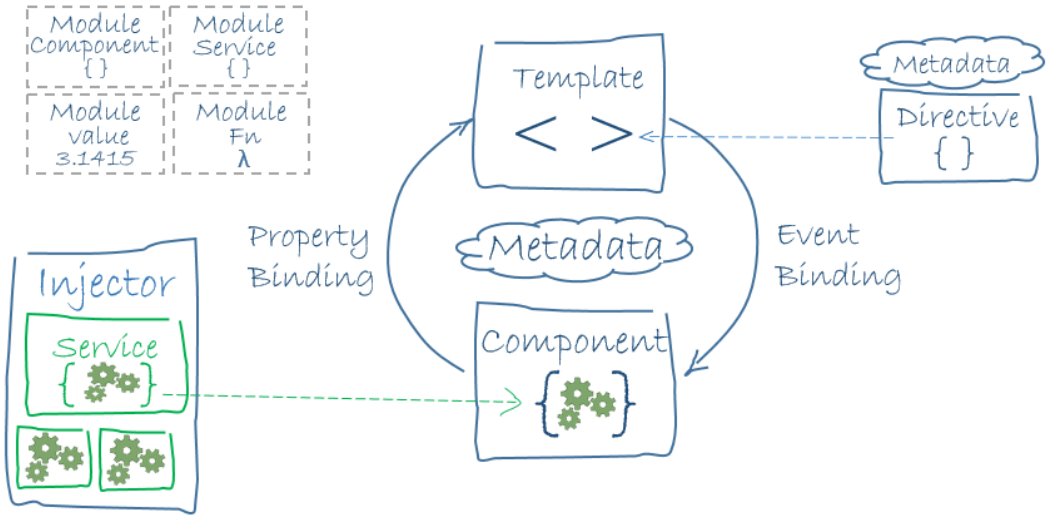
TypeScript je oboje, jazyk aj sada nástrojov na generovanie kódu JavaScript. Navrhol ho vývojár Anders Hejlsberg zo spoločnosti Microsoft (dizajnér C #) ako open-source projekt, ktorý pomáha vývojárom napísať JavaScript kód bez predošlých skúseností s interpretovaným, slabo typovým jazykom. JavaScript sa stal široko obľúbeným programátormi na celom svete, pretože môže fungovať v každom prehliadači na ľubovoľnom operačnom systéme. Príchodom Node.js sa môže jazyk JavaScript spustiť aj na serveri, na pracovnej ploche alebo mobilnom telefóne.

TypeScript generuje JavaScript. Namiesto vyžadovania úplne nového prostredia runtime môže JavaScript generovaný TypeScriptom opätovne použiť všetky existujúce nástroje, frameworky a knižnice, ktoré sú k dispozícii pre jazyk JavaScript. Jazyk TypeScript a kompilátor však prináša vývoj jazyka JavaScript bližšie k tradičnejšiemu objektovo orientovanému prístupu. Najväčšími výhodami TypeScriptu sú:

- Kompilačný krok
- Silné typovnie
- Definovanie dátových typov

- Zapuzdrenie
- Možnosť priradiť funkciám a premenným v triedach dekoratér private alebo public [21]

2.2 Angular 2 architektúra



Obr. 4 Angular2 architektúra [24]

Ako je možné vidieť z diagramu (*Obr. 4*), Angular framework pozostáva z ôsmych hlavných blokov, ktorými sú:

- Moduly
- Komponenty
- Šablóny
- Metadáta
- Data binding
- Direktívy
- Služby
- Dependency injection

2.2.1 Moduly

Angular aplikácie sú označované ako modulárne aplikácie. To znamená, že Angular má vlastný modulárny systém nazývaný Angular modules alebo NgModules. Každá Angular aplikácia má aspoň jednu Angular module triedu, teda koreňový modul nazývaný AppModule. Modul je mechanizmus na zoskupovanie komponentov, direktív, pipes a služieb, a je

možné ich kombinovať s inými modulmi na vytvorenie aplikácie. Angular aplikácia môže byť považovaná za puzzle, kde je potrebný každý kus (alebo každý modul), aby bolo možné vidieť celý obrázok v puzzle. Angular modul môže byť knižnicou pre iný modul. Napríklad knižnica angular2/core, ktorá je primárnym modulom Angular, bude importovaná iným komponentom.[25] Angular modul, či už koreňový alebo vlastný modul, je trieda s dekoratérom `@NgModule`. [23] Dekoratéri sú funkcie ktoré modifikujú JavaScript triedy. Dekoratéri v Angular 2 aplikujú metadáta na triedy. Keď je trieda v aplikácii použitá, Angular 2 preberie tieto metadáta pre nastavenie očakávaného chovania. [22]

```
1. import { NgModule }      from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. @NgModule({
4.   imports:      [ BrowserModule ],
5.   providers:    [ Logger ],
6.   declarations: [ AppComponent ],
7.   exports:      [ AppComponent ],
8.   bootstrap:    [ AppComponent ]
9. })
10. export class AppModule { }
```

Zdrojový kód 2 Ukážka NgModule [24]

NgModule (*Zdrojový kód 2 Ukážka NgModule [24]Zdrojový kód 2*) je dekoračná funkcia, ktorá prevezme metadáta objektu, ktorého nastavenia popisujú module. Najdôležitejšie nastavenia sú:

- **declarations** – view triedy, ktoré patria do tohto modulu. Angular má tri typy view tried: komponenty, direktívy a pipes.
- **exports** – deklarácie, ktoré by mali byt viditeľné a použiteľné v komponentoch iných modulov, ktoré tento modul importujú
- **imports** – ďalšie moduly, ktorých exportované triedy, bude môct module použiť
- **providers** – services modulu, ktoré budú sprístupnené globálnej kolekcii šlužieb, takže budú dostupne vo všetkých častiach aplikácie
- **bootstrap** – hlavný aplikačný view, volaný aj root component [24]

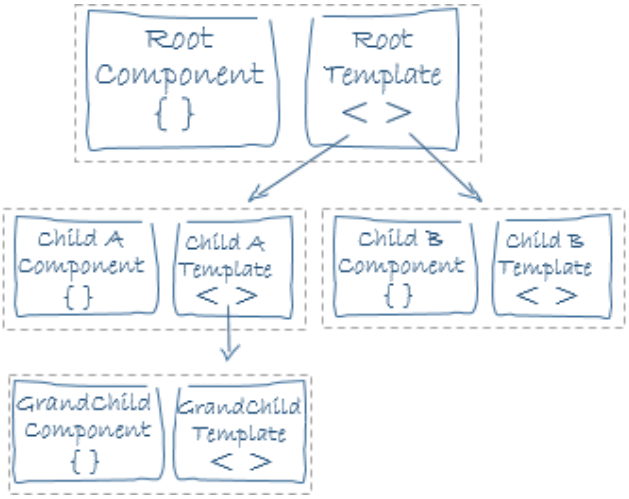
Ako bolo spomínané, existujú dva typy modulov, koreňové moduly a funkčné moduly. Rovnako ako v module je len jeden root komponent a ďalšie možné sekundárne komponenty, tak isto v aplikácii je len jeden root modul a žiaden alebo viacero funkčných modulov. Aby bolo možné spustiť aplikáciu, Angular potrebuje vedieť, ktorý z nich je koreňový modul. Jednoduchý spôsob, ako identifikovať koreňový modul, je pozrieť sa na vlastnosť `imports` jeho dekoračnej funkcie `@NgModule`. Ak modul importuje aplikáciu

`BrowserModule`, potom sa jedná o koreňový modul, ak sa namiesto toho importuje `CommonModule`, potom je reč o funkčnom module. [24]

2.2.2 Komponenty

Komponent je controller trieda so šablónou, ktorá rieši výstup aplikácie a logikou na stránke. Je to kus kódu, ktorý je možné použiť v celej aplikácii. Komponent vie, ako a čo vykresľovať a konfigurovať dependency injection. Komponent obsahuje dve dôležité veci. Jeden je výstup a druhý je nejaká logika. [26]

Ako vidieť na obrázku (*Obr. 5*), aplikácia Angular 2 nie je nič viac ako strom komponentov. V koreňovom uzle tohto stromu je komponent najvyššej úrovne teda samotná aplikácia.



Obr. 5 Strom komponentov [24]

Jednou z hlavných vlastností komponentov je to, že sú kompozitné. To znamená, že môžeme vytvoriť väčšie komponenty z menších. Aplikácia je teda komponent, ktorý vykresľuje iné komponenty. Vzhľadom k tomu, že komponenty sú štruktúrované v strome, komponent, ktorý sa vykresľuje, rekurzívne vykresľuje svoje detské komponenty. [27]

Vytvorenie komponentu

V komponente je potrebné definovať aplikačnú logiku v tele triedy. K tomuto sa pridáva TypeScript dekoratér `@Component`, ako je možno vidieť v ukážke zdrojového kódu (*Zdrojový kód 3*), ktorý dovoľuje modifikovať definíciu triedy alebo funkcie a pridáva metadáta do vlastností a argumentov funkcií.

```
1. @Component({
2.   selector: 'app-hello',
3.   template: '<p>Hello, {{name}}!</p>',
4. })
5. export class HelloComponent {
6.   name: string;
7.
8.   constructor() {
9.     this.name = 'World';
10.      }
11.    }
```

Zdrojový kód 3 Komponent HelloComponet [27]

Parameter `selector` určujete, ako bude komponent rozpoznaný pri vykresľovaní šablón HTML. Myšlienka je podobná výberu CSS alebo XPath. Selektor je spôsob, ako definovať, ktorý HTML element sa bude zhodovať s týmto komponentom. V HTML súbore to bude vyzerať takto:

```
1. <app-hello> </hello-app>
```

Zdrojový kód 4 Selektor komponentu HelloComponent [27]

Parameter `template` je forma HTML, ktorá hovorí Angularu, čo bude vykreslené v DOM. [27]

2.2.3 Šablóny

Výstup komponentu na obrazovku možno definovať pomocou šablóny (templates) v HTML formáte, ktorá hovorí Angular ako zobraziť komponent.

```
1. <h2>Hero List</h2>
2. <p><i>Pick a hero from the list</i></p>
3. <ul>
4. <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
5. {{hero.name}}
6. </li>
7. </ul>
8. <hero-detail *ngIf="selectedHero" [hero]="selectedHero"></hero-
   detail>
```

Zdrojový kód 5 Ukážka šablóny [24]

V poslednom riadku šablóny (*Zdrojový kód 5*) sa nachádza `<hero-detail>` tag. To znamená, že vnútri jednej šablóny môže byť použitý tag reprezentujúci iný komponent. [28]

2.2.4 Metadáta

Metadáta sú spôsob spracovania triedy. Ak máme nejakú triedu, napríklad HeroListComponent, bude ju Angular považovať za obyčajnú triedu, až kým sa nedá nejakým spôsobom vedieť, že sa jedná o komponent. V TypeScripte sa za týmto účelom používa dekoratér `@Component`, ktorý priraďuje metadáta.

```
1. @Component({
2.   selector:    'hero-list',
3.   templateUrl: './hero-list.component.html',
4.   providers:  [ HeroService ]
5. })
6. export class HeroListComponent implements OnInit {
7. /* . . . */
8. }
```

Zdrojový kód 6 Ukážka dekoratéru @Component [24]

V kóde vyššie (*Zdrojový kód 6*) je dekoratér `@Component,` ktorý definuje triedu, nachádzajúcu sa priamo pod ním, ako komponent. Obsahuje informácie, ktoré Angular potrebuje pre vytvorenie a prezentáciu komponentu a jeho výstupu .

V ukážkovom kóde sú tieto vlastnosti dekoratéru `@Component`:

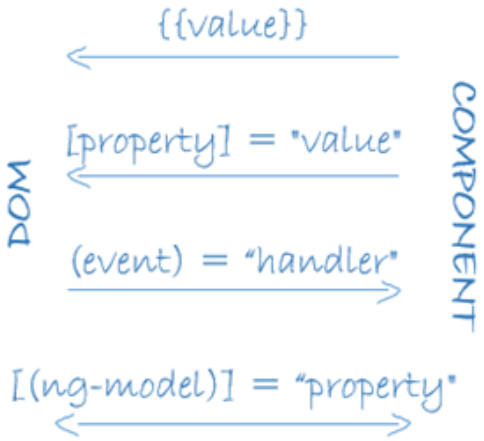
Vlastnosť `providers`: pole dependency injection poskytovateľov pre služby, ktoré komponent vyžaduje. Je to spôsob, ako povedať Angularu, že konštruktor komponentu vyžaduje službu HeroService, aby získal zoznam hrdinov na zobrazenie.

Vlastnosť `templateUrl`, je len vlastne cesta k html súboru, ktorý obsahuje šablónu komponentu.

Vlastnosť `selector`, bola popísaná vyššie. [24]

2.2.5 Data binding

Bez data bindgu, by vývojár bol zodpovedný za prepisovanie údajov do HTML a vytvárať funkcie pre sledovanie zmien, vykonaných užívateľom. Preto Angular framework ponúka data binding mechanizmus, ktorý slúži na synchronizáciu dát medzi modelom a výstupom na obrazovke. Pridaním binding značky do HTML šablóny sa dá Angularu vedieť ako je šablóna spojená s komponentom.



Obr. 6 Data Binding [24]

Ako ukazuje diagram (*Obr. 1*), existujú štyri formy syntaxi väzby údajov. Každá forma má smer buď do DOM, z DOM alebo v oboch smeroch.

Jednosmerný data binding

```
1. <li>{{name}}</li>
2. <img [src]="imageUrl"/>
3. <li (click)="selectPerson(name)"></li>
```

Zdrojový kód 7 Ukážka jednosmerného data bindingu

V HTML kóde šablóny vyššie (*Zdrojový kód 7*) je ukázane ako vyzerá bindovanie v jednom smere.

Interpolácia na prvom riadku kódu `{{name}}` zobrazuje hodnotu premennej hero.name, nachádzajúcej sa v komponente.

Property binding `[hero]` nastavuje atribút `src` elementu `img` hodnotou uloženou v premennej `imageUrl` v komponente.

Event binding (`click`) po kliknutí na element zavolá metódu `selectPerson` definovanú v komponente.

Obojsmerný data binding

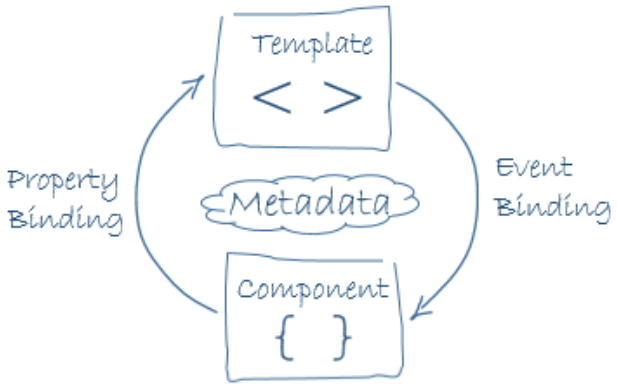
Obojsmerný data binding kombinuje vstupné a vstupné bindovanie do jednej notácií použitím `ngModel` direktívy. [31] Tu je príklad zo šablóny HeroDetailComponent:

```
1. <input [(ngModel)]="name">
```

Zdrojový kód 8 Ukážka dvojsmerného data bindingu

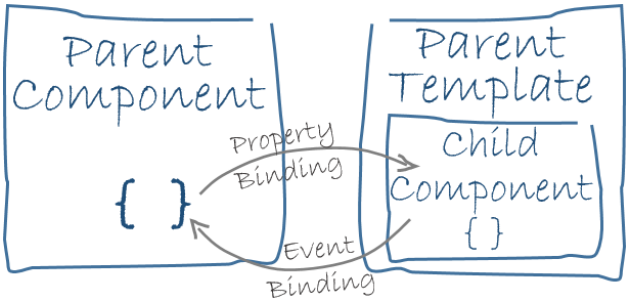
V obojsmernom data bindingu sa hodnota v input boxe nastavuje z komponentu rovnako ako property binding a zmeny používateľa v `input` boxe sa takisto automaticky prejavia zmenou hodnoty v premennej `name` v komponente.

Angular spracováva všetky data bindings raz za jeden cyklus udalostí v JavaScripte , od koreňa stromu komponentov aplikácie cez všetky podradené komponenty.



Obr. 7 Obojsmerný data binding [24]

Data binding hrá dôležitú úlohu pri komunikácii medzi šablónou a jej komponentom.



Obr. 8 Data binding v rámci komunikácie medzi rodičovským a potomkovým komponentom [24]

Data binding je tiež dôležitý pre komunikáciu medzi rodičovskými a potomkovými komponentami. [24]

2.2.6 Direktívy

Angular šablóny sú dynamické. Keď ich Angular vykresľuje, transformuje vzhľad, chovanie alebo vrstvy DOM elementov na základe inštrukcií v direktívach. [31] Direktívy sú jedným z hlavných stavebných prvkov Angularu používaných na vývoj aplikácii. Direktíva je trieda s `@Directive` dekoratérom. Komponent je direktíva so šablónou a `@Component` dekoratér je vlastne `@Directive` dekoratér rozšírený o šablónovo orientované vlastnosti. [24]

Existujú tri hlavné typy direktív v Angulare:

Komponent

Direktíva so šablónou, bola popísaná v časti o komponentoch.

Atribútová direktíva

Atribútové direktívy sú spôsob, ako zmeniť vzhľad alebo správanie komponentu alebo natívneho prvku DOM. Tieto direktívy sa podobajú obyčajným HTML atribútom v šablónach. [32] Je možné vytvoriť si vlastné atribúty alebo používať aj build-in atribúty v Angulare. Vyššie bola opisovaná direktíva `ngModel`, ktorá implementuje obojsmerné data binding. Najpoužívanejšími sú `ngClass` direktíva, ktorá mení `class` atribút, ktorý je viazaný na komponent alebo element, a `ngStyle` direktíva, ktorá modifikuje `style` atribút elementu.[24]

Štruktúrna direktíva

Štruktúrne direktívy riešia akým spôsobom sa komponent alebo element vykresľuje. Upravuje vrstvy DOM pridávaním, nahradzovaním alebo mazaním jeho elementov. Angular má niekoľko vstavaných štruktúrnych direktív, ako napríklad `ngIf`, `ngFor` a `ngSwitch`. Direktíva `ngIf` podmienečne pridáva alebo maže obsah DOM na základe toho či je výraz `true` alebo `false`. Direktíva `ngFor` je spôsob ako opakovať element použitím poľa, cez ktoré sa `iteruje`. Pomocou `ngSwitch` direktívy je možné určiť na základe premennej, ktorý element sa bude vykresľovať. [24]

```
1.  <li *ngFor="let hero of heroes"></li>
2.  <hero-detail *ngIf="selectedHero"></hero-detail>
3.  <div [ngSwitch]="currentHero.emotion">
4.    <happy-hero *ngSwitchCase="'happy'"[hero]="currentHero"></happy-
    hero>
5.    <sad-hero *ngSwitchCase="'sad'"[hero]="currentHero"></sad-hero>
6.    <confused-hero
    *ngSwitchCase="'confused'"[hero]="currentHero"></confused-hero>
7.    <unknown-hero  *ngSwitchDefault [hero]="currentHero"></unknown-
    hero>
8.  </div>
```

Zdrojový kód 9 Ukážka použitia direktív

2.2.7 Služby

Služby sú JavaScript metódy, ktoré sú zodpovedné len za vykonanie konkrétnej úlohy. Angular služby sú injektované pomocou dependency injection. [28] Služba je široká kate-

gória, čokoľvek môže byť službou. Typicky služba je trieda s jasne definovanou špecifickou úlohou, akú ma vykonávať. Príkladom služby môže byť posielanie dotazov na server, prihlasovanie a iné.

Angular nemá žiadnu definíciu služby, napriek tomu sú základom každej Angular aplikácie. Služby sú najviac používane v komponentoch. Triedy komponentov by mali byť štíhle. Nepreberajú údaje zo servera, neoverujú vstupné údaje ani nepristupujú priamo do konzoly. Tieto úlohy sú vykonávane službami. [24]

2.2.8 Dependency Injection

Účelom dependency injection je zjednodušiť správu závislostí v komponentoch. Dependency injection vytvára novú inštanciu triedy spolu s požadovanými závislosťami. [30] Väčšina závislostí sú služby. Znížením množstva informácií, ktoré komponent potrebuje vedieť o svojich závislostiach, môžu byť dosiahnuté jednoduchšie jednotkové testy a kód bude flexibilnejší.

Angular dokáže zistiť, ktoré služby potrebuje komponent pri pohľade na typy parametrov konštruktora. Napríklad konštruktor komponentu v ukážke (*Zdrojový kód 10*) potrebuje HeroService:

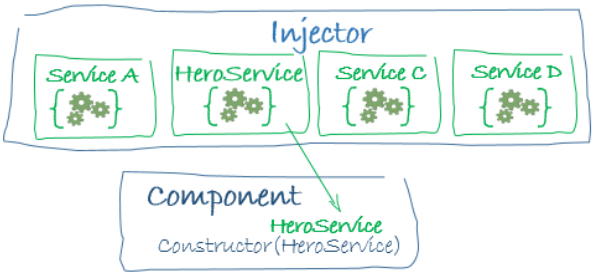
```
1. constructor(private service: HeroService) { }
```

Zdrojový kód 10 Konštruktor komponentu

Keď Angular vytvorí komponent, najprv sa spýta injektora na služby, ktoré komponent vyžaduje.

Injektor udržiava zásobník inštancií služieb, ktorý predtým vytvoril. Ak požadovaná inštancia služby nie je v zásobníku, injektor ju vyrobí a pridá ju do kontajnera predtým, ako vráti službu do Angularu. Keď sú všetky požadované služby obslúžene a vrátené, Angular môže zavolať konštruktor komponentu s týmito službami ako argument. Tomuto sa hovorí dependency injection.

Proces injektovania HeroService vyzerá nasledovne:



Obr. 9 Injektovanie služby HeroService [24]

Ak injektor nemá HeroService, musí byť poskytovateľ (provider) služby HeroService registrovaný injektorom. Provider je spôsob ako vytvárať služby. [29] Provider môže byť registrovaný v moduloch alebo komponentoch. Ak chceme aby služby boli k dispozícií v rámci celej aplikácie je potrebné pridať providerov do koreňového modulu. Ak je provider registrovaný v rámci komponentu, znamená to že nové inštancie služieb budú vytvorené pre každú inštanciu komponentu. [24]

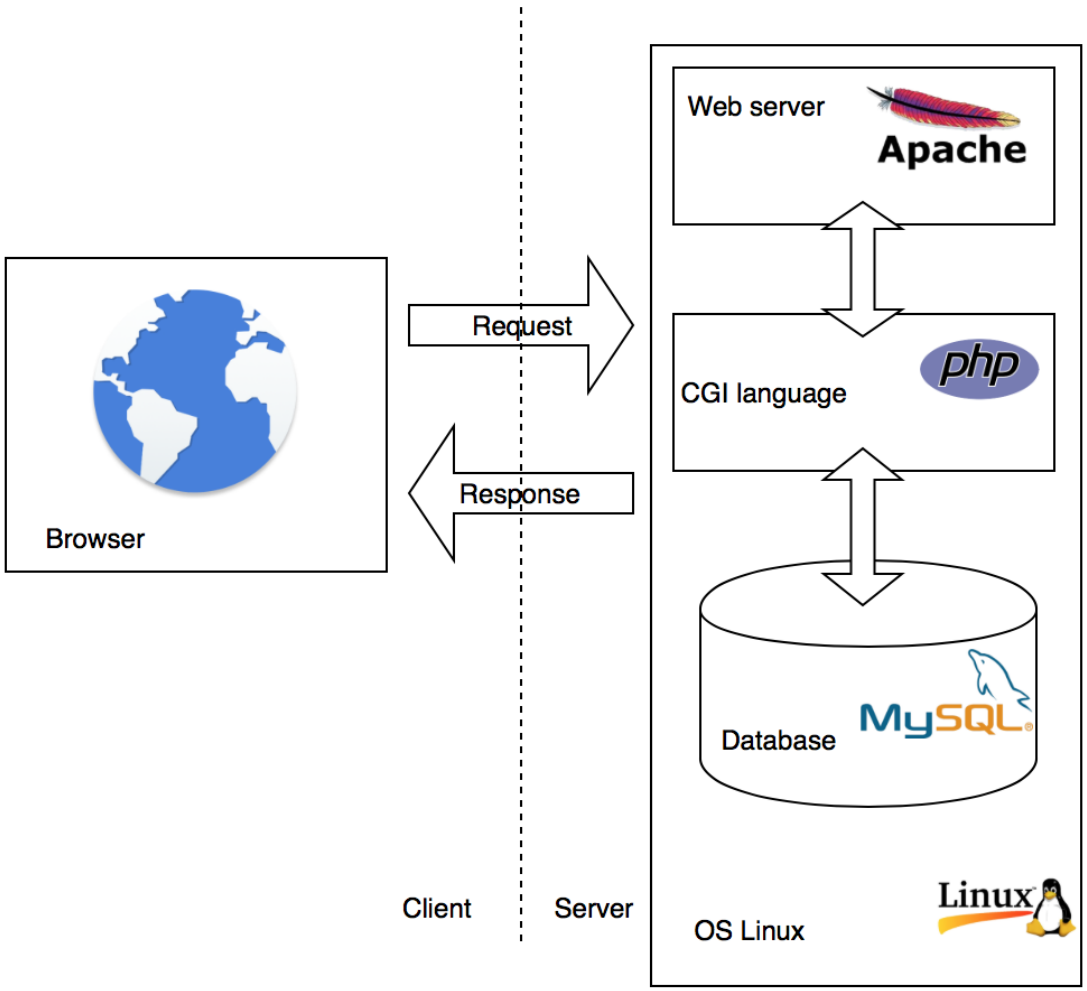
3 LAMP STACK A JEHO ZROVNANIE S MEAN STACK

V tejto kapitole sú najprv stručne popísané hlavné časti balíčka LAMP stack a tabuľke zhrnuté hlavné rozdiely medzi LAMP a MEAN prístupom.

3.1 LAMP stack

LAMP je open-source platforma na vývoj webových aplikácií, ktorá používa operačný systém Linux, Apache ako webový server, MySQL ako systém riadenia relačných databáz a PHP ako objektovo orientovaný skriptovací jazyk. (Niekedy sa miesto PHP používa Perl alebo Python.) [34]

3.1.1 Časti LAMP stacku



Obr. 10 LAMP stack architektúra

Na obrázku vyššie sú zobrané hlavné časti LAMP stacku a vzájomne vzťahy medzi nimi. Týmito hlavnými časťami sú:

Linux

Operačný systém, na ktorom bežia všetky súčasti LAMP stacku. Môže to byť serverovo orientovaný Linux alebo pokročilá verzia Linuxu s grafickým rozhraním a vývojovým prostredím.

Apache

Webový server Apache beží na linuxovom servery a prijíma požiadavky od klienta a odpovedá na tieto požiadavky. Ak používateľ požiadavku o napríklad o súbor index.html, Apache vyhľadá tento súbor na servery a pošle ho späť do klientovho prehliadača. Apache obsahuje mnoho funkcií, ktoré sú implementované ako kompilované moduly rozširujúce jadro. Medzi tieto funkcie patrí podpora programovacích jazykov na strane serveru, autentizačné moduly, moduly pre kompresiu dát webových stránok, module pre ochranu a prevenciu webových aplikácii pred napadnutím a iné. Apache podporuje aj virtuálny hosting, čo umožňuje na jednom fyzickom počítači obsluhovať viacero webových stránok. [40]

MySQL

MySQL je jedným z najpopulárnejších open-source databázových systémov. Databáza MySQL ukladá všetky údaje a konfiguračné informácie pre webovú stránku. Často krát sú dáta vyžadované pre generovanie webových stránok, v závislosti na tom kto je návštevník. Môžu to byť dáta ako meno, objednávky, stav účtu a iné. MySQL je relačný databázový systém používaný pre weby. Beží na severy a je ideálny pre malé aj veľké aplikácie. Používa štandardizované SQL dotazy. [33]

PHP

PHP je skriptovací jazyk spúšťaný na strane serveru určený primárne na vytváranie webových stránok ale aj akýkoľvek iných aplikácií. PHP kód je interpretovateľný web serverom cez PHP procesor, ktorý generuje výslednú webovú stránku, ktorú používateľ na strane klienta vyžaduje. PHP kód môže byť priamo vkladaný do HTML súborov. [33]

3.2 Porovnanie LAMP a MEAN

Medzi jedny z najobľúbenejších softwarových balíčkov alebo prístupov, pre vývoj webových aplikácií, patria momentálne LAMP a MEAN. Sú postavené na dvoch odlišných technológiách či už zo strany front-endu alebo back-endu.

V tabuľke (**Chyba! Nenalezen zdroj odkazů.**) nižšie sú zhrnuté hlavné rozdiely medzi oboma prístupmi z hľadiska použitých technológií.

Tab. 1 Porovnanie LAMP a MEAN

Technológia	**LAMP**	**MEAN**
Jazyk	Najčastejšie PHP, ale môže byť aj Perl alebo Python	JavaScript, v prípade novej verzie Angular 2 TypeScript
Databáza	MySQL – SQL relačná databáza	MongoDB – NoSQL, dokumentová (JSON dokumenty) databáza
Webový server	Apache	Node.js spolu s frameworkom Express.js slúžiacim ako middleware pre routovanie
Operačný systém	L ako Linux, no môže existovať aj vo verzií s Windows	Nie je závislý na operačnom systéme, akýkoľvek systém, na ktorom je možné spustiť Node aplikácie

Samozrejme je možné obe balíčky trochu premiešať. Mnoho vývojárov používa MongoDB s Apache a PHP a iní používajú MySQL s Node.js. Framework Angular môže fungovať s akýmkoľvek serverom, a to aj s PHP, ktorý poskytuje údaje z MySQL.

II. PRAKTICKÁ ČÁST

4 POŽIADAVKY ZADÁVATEĽA

V roku 2016 v Českej Republike nadobudol účinnosť zákon o vedení registra zmlúv. Všetky verejné inštitúcie, územné samosprávne celky, štátne podnik, právnické osoby v ktorých má väčšinovú majetkovú účasť štát alebo územný samosprávny celok a ďalšie inštitúcie definované týmto zákonom povinnosť zverejňovať novo uzatvorené zmluvy v tomto registre. Zadávateľ dal požiadavku na vytvorenie aplikácie, ktorá bude v týchto zmluvách vyhľadávať a zobrazovať ich.

4.1 Funkčné požiadavky

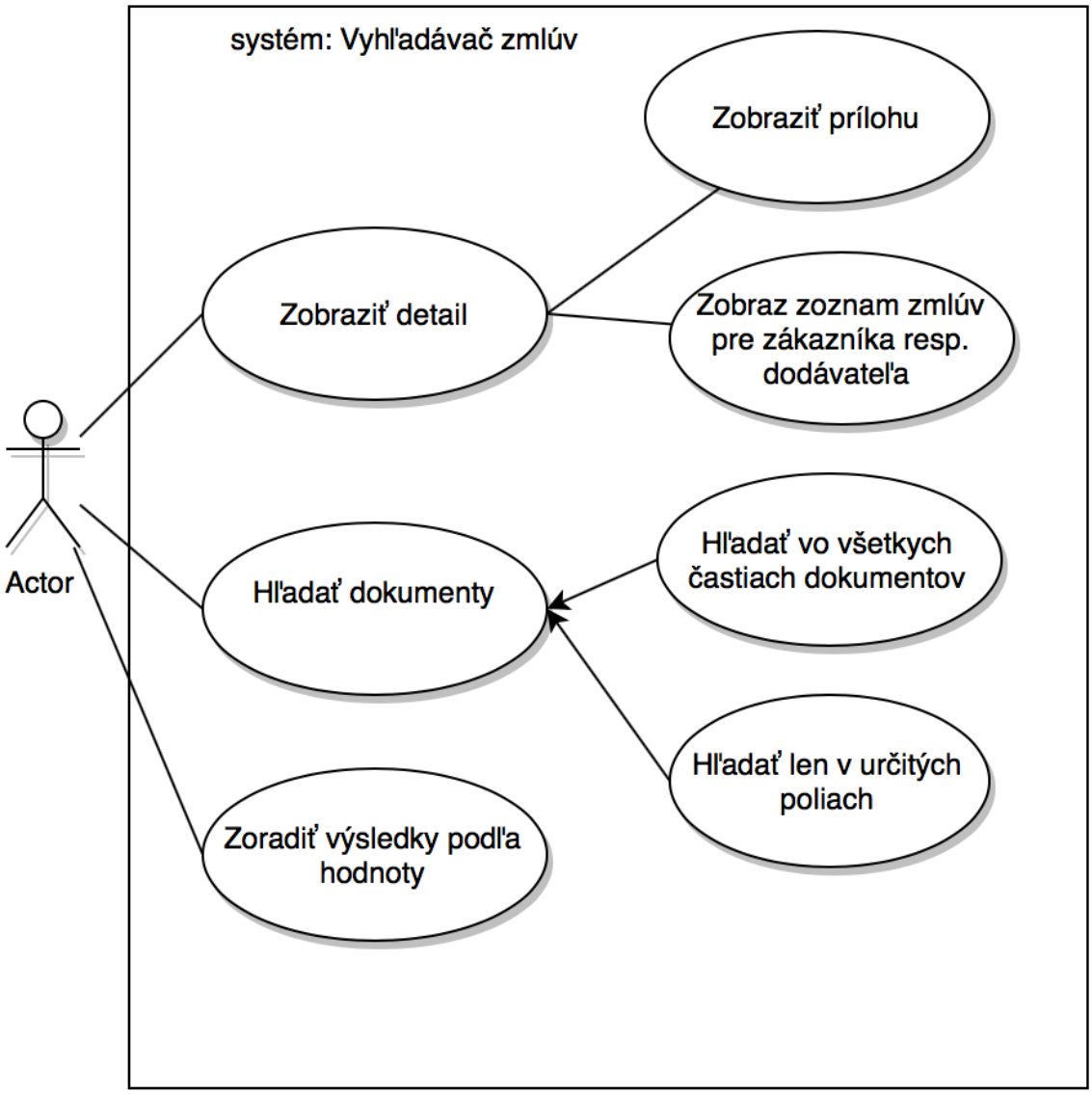
Je požadované vytvorenie single-page webovej aplikácie, ktorej cieľom bude možnosť zmluvy vyhľadávať na základe určitých vyhľadávacích kritérií, zobrazovať detail týchto zmlúv ako aj samotné zmluvy. Vyhľadávanie bude možné jednoduché na základe textu hľadaného vo všetkých častiach dokumentu, ako aj podrobné vyhľadávanie na základe týchto parametrov: Názov zmluvy, IČO zákazníka, meno zákazníka, IČO dodávateľa a meno dodávateľa. Výsledky vyhľadávania budú reprezentované zoznamom položiek so základnými údajmi o zmluve. Po kliknutí na meno zákazníka alebo dodávateľa bude možné zobraziť všetky zmluvy pre tohto zákazníka respektíve dodávateľa. Detail zmluvy bude zahrňovať všetky informácie o zmluvy ako aj odkaz na prílohy k zmluve. Aplikácia by mala byť schopná zobraziť tieto prílohy. Ďalej bude možné zoradiť zmluvy vo výsledkoch vyhľadávanie podľa výšky hodnoty zmluvy a to zostupne aj vzostupne. Aplikácia slúži čisto na vyhľadávanie, tým pádom správa dokumentov nebude možná cez užívateľské rozhranie.

4.2 Nefunkčné požiadavky

Zadávateľ určil aj niekoľko nefunkčných požiadaviek. Jednou z nich je, aby serverová aplikácia mohla byť spustená na akomkoľvek servery bez ohľadu na operačný systém. Ďalej je žiadané, aby v rámci celého riešenia bol používaný jeden programovací jazyk. Keďže zadávateľ poskytol ElasticSearch snapshot, v ktorom sú archivované zmluvy, dokumenty budú uložené v ElasticSearch databáze a bude využívaný ElasticSearch REST API pre vyhľadávanie v dokumentoch.

4.3 Diagram prípadov užitia

V diagrame prípadov užitia (*Obr. 11*), inak nazývaného aj use case diagram, je zobrazené chovanie aplikácie z podhľadu používateľa (actor), teda jednotlivé úkony ktoré môže používateľ v aplikácií vykonávať.



Obr. 11 Diagram prípadov užitia

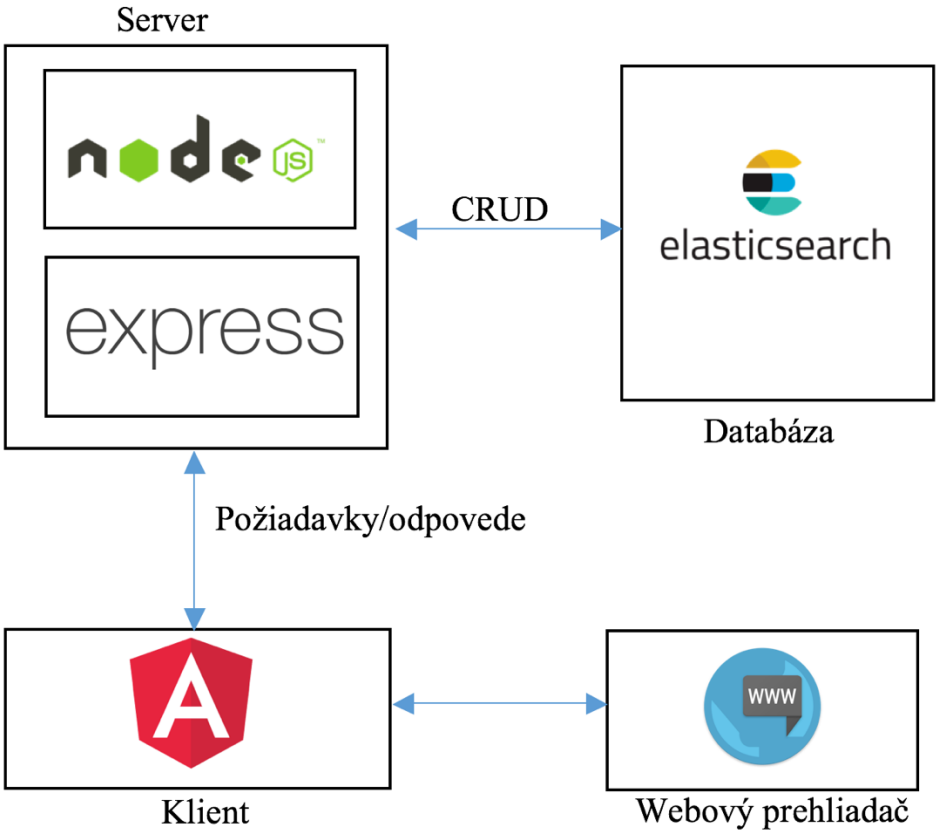
5 POPIS APLIKÁCIE

Jednou z nefunkčných požiadaviek bolo použitie jedného programovacieho jazyka v rámci celej aplikácie. Ideálnym riešením pre splnenie tejto požiadavky je použitie prístupu MEAN.js stack, ktorý používa jediný programovací jazyk JavaScript. Bude sa jednať o ľahko modifikovaný MEAN stack, keďže namiesto dokumentovej databázy MongoDB bude použitý ElasticSearch, z dôvodu poskytnutia zmlúv vo forme ElasticSearch snapshotu.

Väčšina technológií balíku MEAN stack boli popisované v teoretickej časti, okrem ElastasticSearch. Elasticsearch je vysoko škálovateľný open-source fulltextový vyhľadávací a analytický engine. Umožňuje rýchlo a v reálnom čase ukladať, vyhľadávať a analyzovať veľké objemy dát v takmer reálnom čase. Všeobecne sa používa ako engine alebo technológia, ktorá poháňa aplikácie, ktoré majú zložité vyhľadávacie funkcie a požiadavky. [41]

Použitie MEAN stacku tiež rieši aj nefunkčnú požiadavku na beh serverovej aplikácie na akomkoľvek operačnom systéme, keďže MEAN stack, používa Node.js platformu, ktorá je nezávislá na operačnom systéme.

5.1 Architektúra aplikácie



Obr. 12 Architektúra aplikácie

Na obrázku (*Obr. 12*) je zobrazená architektúra aplikácie.

Serverová platforma Node.js je určená ku konfigurácii webového servera. Nad Node.js bude bežať framework Express.js, ktorý sa bude starať spracovanie požiadaviek pomocou RESTful API a o routovanie a o komunikáciu s ElasticSearch.

Vyhľadávací engine ElasticSerach bude bežať na samostatnom servery. ElasticSearch ma uložené dokumenty a bude pomocou CRUD api komunikovať so serverom.

Angular ako framework pre webového klienta je použitý pre vytvorenie používateľského rozhrania a komunikáciu so serverom.

5.2 Priebeh vývoja

Ako prvé bolo potrebné nainštalovať potrebné technológie, ako sú Node.js, pomocou npm správcu balíčkov nainštalovať Express.js a Angular-CLI technológie, ktoré sú tiež npm modulmi.

Po nainštalovaní všetkých potrebných modulov a technológií, sa začal samotný vývoj. Ako prvé sa v Angular-CLI vytvoril nový projekt pomocou príkazu `ng new contracts`. To vytvorilo zložku s rovnakým názvom, do ktorej sa vygenerovali potrebné súbory pre základnú kostru aplikácie a taktiež sa nainštalovali potrebné npm moduly.

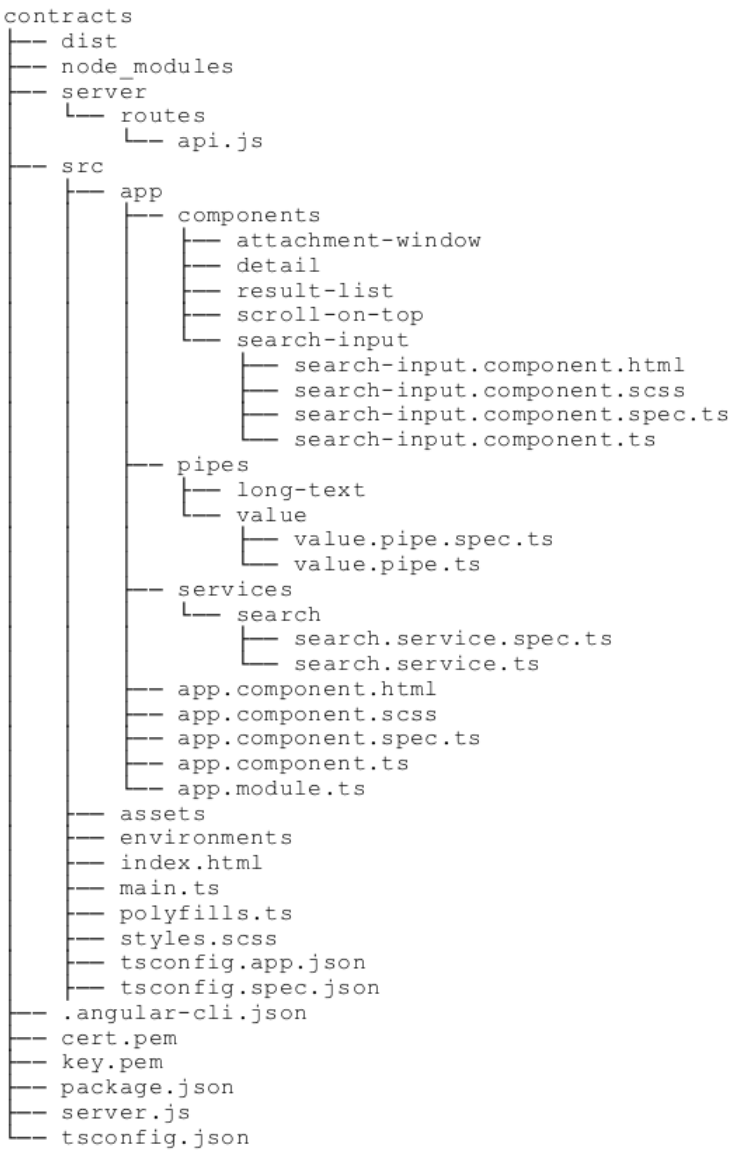
Počas takmer celého vývoja webovej aplikácie, nebol potrebný Node.js server s Express.js frameworkom, keďže Angular-CLI ponúka lite-server, ktorý sa spúšťa príkazom `ng serve`, a ten predstavuje jednoduchý node server reagujúci na zmeny kódu v súboroch tak, že po uložení okamžite načíta vyvíjanú aplikáciu v prehliadači. Komunikácia s ElasticSearchom prebiehala na priamo, a teda obchádzala serverovú aplikáciu.

Komponenty, služby ale aj pipes boli tak isto vygenerované pomocou konzolových príkazov ako `ng generate component/service/pipes`. To automaticky vygenerovalo súbory s potrebnými hlavičkami a v prípade komponentov a pipes boli automaticky pridané importy do koreňového modulu NgModule.

Po dokončení vývoja front-endu bola aplikácia zbuildovaná cez príkaz `ng build`. To vytvorilo distribučnú verziu Angular aplikácie s hlavným index.html súborom. Ako posledná bola vytvorená serverová Node.js využívajúca framework Express.js. V aplikácii bola nastavená cesta k tomuto index.html súboru, z toho dôvodu aby bola vrátená Angular aplikácia, keď sa prijme požiadavka o jej poslanie.

Ako posledné bolo potrebné nakonfigurovať routovanie Express.js aplikácie a presunúť kód pre komunikáciu s databázou z Angular aplikácie do Expressu.

5.3 Adresárová štruktúra aplikácie



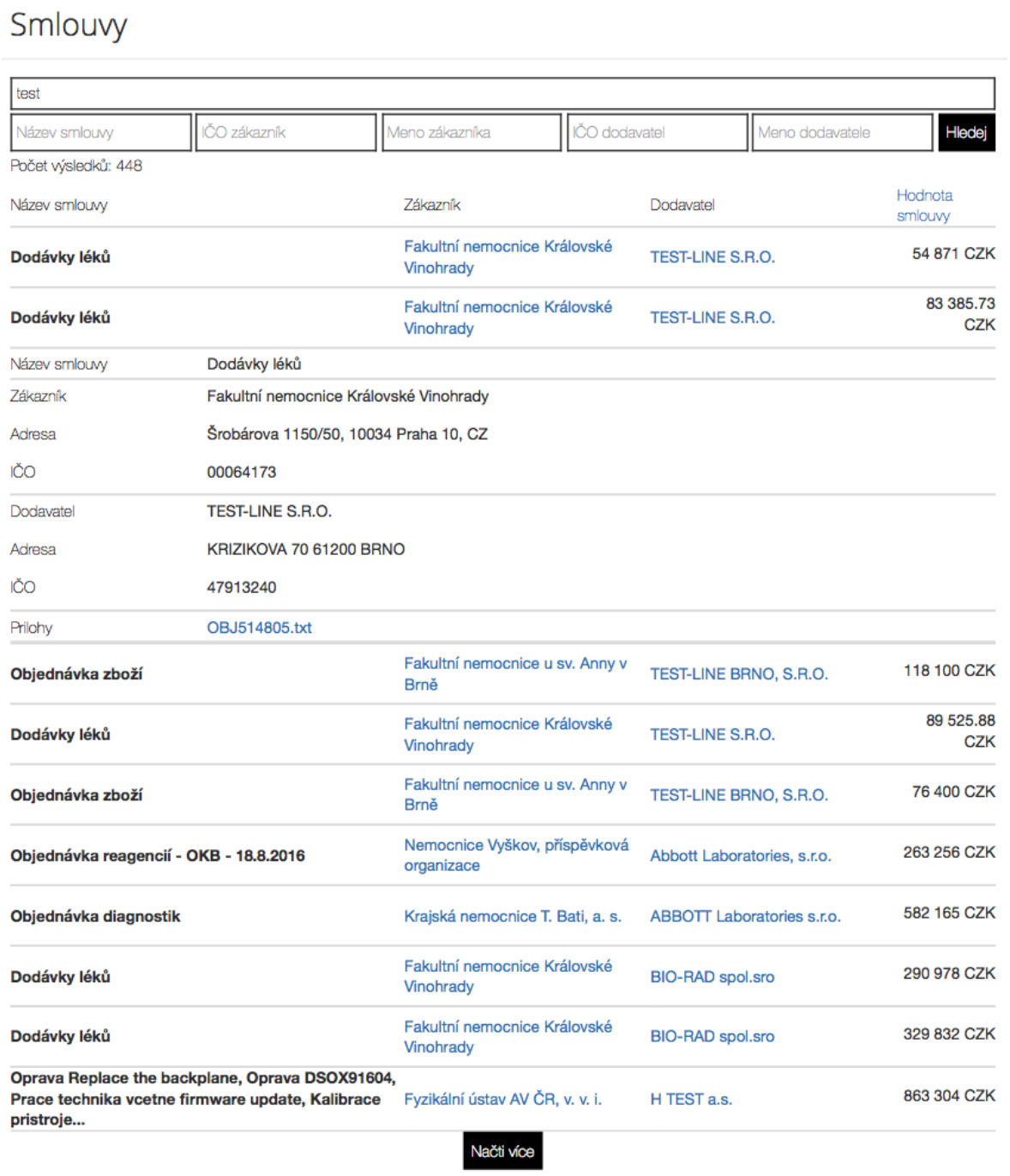
Obr. 13 Adresárová Štruktúra MEAN stack aplikácie

Na obrázku (*Obr. 13*) je adresárová štruktúra celého projektu, ktorá obsahuje:

- Priečinok dist: Angular-CLI po príkaze `ng build` vytvára tento priečinok, ktorý obsahuje kompilovaný kód, ktorý je možné hostiť na ľubovoľnom súkromnom serveri rovnako ako bežné webové stránky HTML
- Priečinok node_modules: Takisto generovaný pri vytváraní aplikácie, obsahuje moduly alebo knižnice, ktoré bude aplikácia používať.
- Priečinok server: Obsahuje súbor api.js, ktorý predstavuje Express aplikáciu pre routovanie.
- Priečinok src: Obsahuje všetky zdrojové súbory.

- Priečinok app: Väčšina kódu aplikácie je v tejto zložke. Pri vývoji bude vývojár vkladať súbory so zdrojovým kódom tohto priečinku.
- Objekty typu komponent, služba alebo pipe sú rozdelené do príslušných priečinkov.
- Po vygenerovaní komponentu použitím príkazu z Angular-CLI, sa vytvorí zložka z jeho názvom a štyrmi súbormi. Súbor *.hmtl obsahuje šablónu komponentu, *.scss obsahuje štýly pre túto šablónu. Súbor s príponou *.ts obsahuje definíciu triedy tohto komponentu.
- Súbory *.ts v prípade služieb a pipes majú rovnakú úlohu.
- Súbory *.specs.ts obsahujú jednotkové testy pre zdrojové súbory.
- Súbor app.module.ts: Definuje AppModule, ktorý hovorí Angularu, ako poskladať aplikáciu.
- Priečinok assets: Obsahuje obrázky, ikony a iné súbory, ktoré aplikácia vyžaduje ale nesúvisia s kódom.
- Priečinok enviroments: Obsahuje súbory s premennými prostredia pre vývojovú alebo produkčnú fázu.
- Súbor index.html: Hlavná HTML stránka, ktorá je vrátená keď, niekto navštívi stránku.
- Súbor main.ts: Hlavný vstupný bod pre aplikáciu.
- Súbor polyfills.ts: Rozličné prehliadače majú rôzne úrovne podpory pre webové štandardy. Pollyfills pomáhajú normalizovať tieto rozdiely.
- Súbor styles.scss: Globálny CSS súbor. Ak je vyžadované použiť nejaký štýl v rámci celej aplikácie je vhodné ich umiestňovať sem.
- Súbory tsconfig.json a tsconfig.spec.json: Obsahujú nastavenia pre TypeScript kompilátoru pre Angular aplikáciu.
- Súbor angular-cli.json: JSON súbor obsahujúci hlavnú konfiguráciu celej aplikácie vytvorenej pomocou Angular-CLIi. Obsahuje informácie o cestách k hlavným modulom a súborom.
- Súbor package.json: Obsahuje informácie o projektových závislostiach a to názov a verzia modulu. Na základe tohoto súboru príkaz npm install stiahne moduly a uloží ich do priečinku node_modules.
- Súbor server.js: Aplikácia Node.js serveru.
- Súbor tsconfig.json: Prítomnosť tohto súboru indikuje že sa jedná o TypeScript projekt. Špecifikuje root súbory a nastavenia kompilátora.

5.4 Angular aplikácia

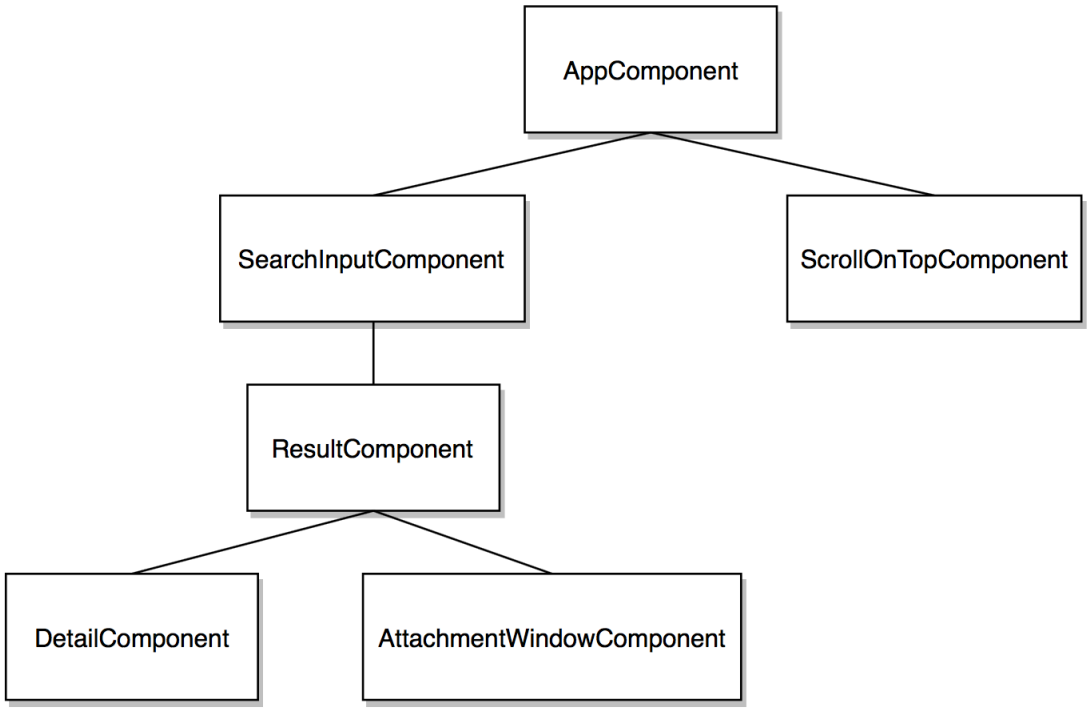


Obr. 14 Používateľské rozhranie

Na obrázku (*Obr. 14*) je screenshot používateľského rozhrania aplikácie. To je možné rozdeliť do niekoľkých komponentov. Ako bolo spomínané v kapitole o Angular 2, aplikáciu si možno predstaviť ako strom komponentov. Strom komponentov tejto aplikácie je možno vidieť na obrázku (*Obr. 15*). Koreňovým komponentov je AppComponent, v ktorom sú komponenty ScrollOnTopComponent, pre vrátenie sa na vrch stránky, a komponent SearchInputComponent ktorý zapuzdruje celé vyhľadávanie. Jediným uzlom

v SearchInputComponent je uzol ResultComponent, ktorý predstavuje zoznam výsledkov. Každá položka v zozname má v sebe komponenty DetailComponent pre zobrazenie detailu zmluvy a AttachmentWindowComponet pre zobrazenie prílohy k zmluve.

5.4.1 Komponenty



Obr. 15 Strom komponentov aplikácie

AppComponent

Je koreňovým komponentom aplikácie. Neobsahuje žiadnu aplikačnú logiku. V tele šablóny sa nachádzajú `<app-search-input>` a `<app-scroll-on-top>` elementy, ktoré predstavujú komponenty SearchInputComponent a ScrollOnTopComponent.

SearchInputComponent

Obr. 16 Komponet SearchInputComponent

Vykreslením šablóny tohto komponentu sa zobrazia polia pre vyhľadávanie v dokumentoch. Pri zadávaní textu do horného poľa sa automaticky začne volať funkcia, ktorá sa zavolá v prípade vzniku udalosti `keyup`, ktorá reaguje na zmeny vo vstupnom

poli. Volaná funkcia ďalej volá metódu triedy SearchService, ktorá sa stará o komunikáciu so serverom. Po skončení volania metóda vráti objekt tipu Observable. Observable otvára nepretržitý komunikačný kanál, v ktorom môžu byť v priebehu času emitované viaceré hodnoty údajov. Zavolaním metódy `subscribe` na tomto Observable objekte, je možné sledovať akékoľvek dáta, ktorá prúdia týmto kanálom, v tomto prípade JSON súbor obsahujúci výsledky vyhľadávania. [35]

Podobným princípom funguje vyhľadávanie podľa špecifických parametrov, po zadaní parametrov a stlačení tlačidla „Hledej“ sa zavolá funkcia, ktorá tiež volá metódu triedy SearchService, pre vrátenie výsledkov.

Komponent SearchInputComponet využíva dependency injection, keďže konštruktor komponentu vyžaduje službu SearchService.

Trieda komponentu tiež obsahuje členskú premennú `hitsCount` typu `number`, do ktorej sa ukladá počet nájdených dokumentov. V tele šablóny možno nájsť element < `<app-result-list>` so štruktúrnou direktívou `*ngIf="hitsCount"`. Takto je dosiahnuté, že celý obsah šablóny ResultListComponentu bude v rámci DOM renderovaný, len v prípade nenulového výsledku vyhľadávania.

ResultListComponent

Název smlouvy	Zákazník	Dodavatel	Hodnota smlouvy
684820 - měřící přístroj MD Test pro kontrolu elektrické bezpečnosti	Fakturace (Ministerstvo obrany)	MICRONIX,spol. s r.o.	121 800 CZK
Penetrační testy spisové služby - Neautentizovanýpenetrační test tlustého klienta a webové služby (pokus...	Zlínský kraj	AEC a.s.	50 000 CZK
Smlouva o spolupráci - test PANORAMA	Fakultní nemocnice v Motole	Bioptická laboratoř s.r.o.	0 CZK
Test v aerodynamickém tunelu	Výzkumný a zkušební letecký ústav, a.s.	Výzkumný a zkušební letecký ústav, a.s.	0 CZK

Obr. 17 Komponent ResultListComponent

Komponent reprezentujúci tabuľku výsledkov vyhľadávania (*Obr. 17*). Vstupným parametrom komponentu je foundContracts, ktorý nesie objekt v JSON formáte, ktorý je výsledkom vyhľadávania. Komunikácia rodičovského komponentu SearchInputComponent a jeho podradeného uzla prebieha pomocou Event Bindingu a Property Bindigu. Výstupom je objekt sort typu EventEmitter, ktorý emituje udalosti. Slúži na komunikáciu s rodičovským komponentom SearchInputComponent. Po kliknutí na popis Hodnota zmlu-

vy sa zavolá funkcia `sortByValueClick()`, ktorá vo svojom tele obsahuje riadok kódu `this.sort.emit('sort');` Tento kód emituje udalosť a do tela tejto udalosti vkladá text zo vstupného parametru metódy `emit`. Po kliknutí na meno dodávateľa alebo zákazníka, sa vyhľadajú všetky zmluvy pre tohto dodávateľa respektíve zákazníka a zobrazia sa vo výsledkoch vyhľadávania.

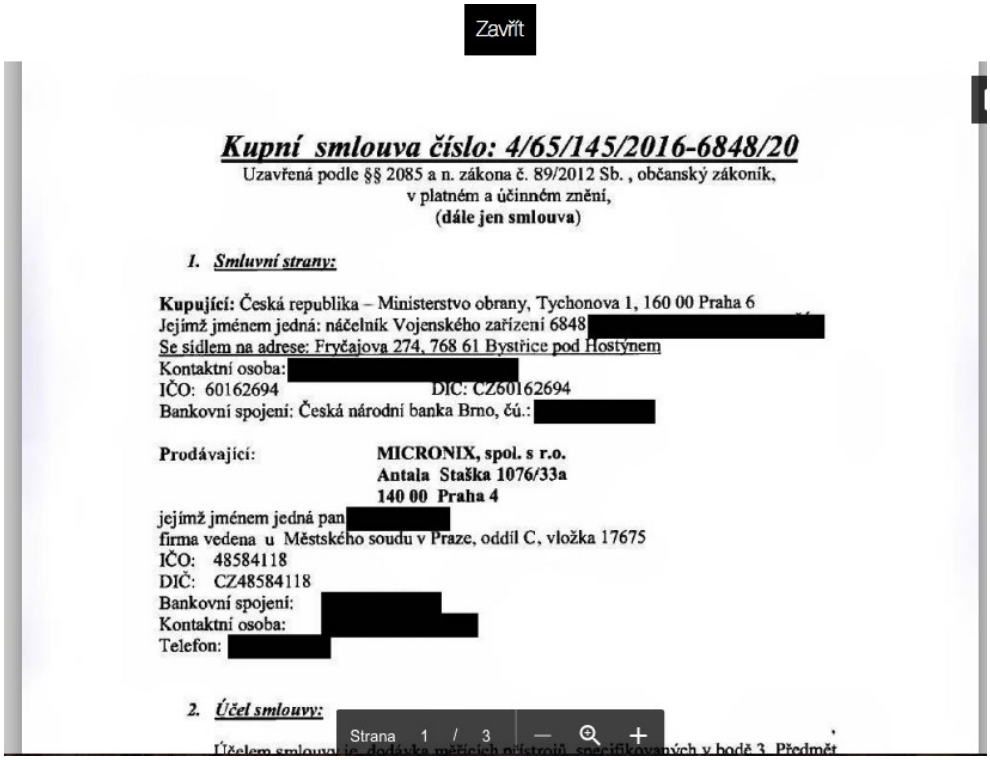
DetailComponent

Název smlouvy	Dodávky léků
Zákazník	Fakultní nemocnice Královské Vinohrady
Adresa	Šrobárova 1150/50, 10034 Praha 10, CZ
IČO	00064173
Dodavatel	TEST-LINE S.R.O.
Adresa	KRIZIKOVA 70 61200 BRNO
IČO	47913240
Prílohy	OBJ516300.txt

Obr. 18 Komponent DetailComponent

Po kliknutí na položku v zozname výsledkov vyhľadávanie sa rozbalí detail (*Obr. 18*). Tento obsahuje podrobnejšie informácie o zmluve, ako celý text názvu, adresa a IČO dodávateľa a zákazníka a odkazy na prílohy k zmluve. Vstupom do komponentu je `contract`, ktorá obsahuje prináležiaci dokument. Výstupom je opäť objekt typu EventEmitter, ktorý emituje udalosť pri kliknutí na prílohu a pošle názov prílohy vnútri tejto udalosti. Táto udalosť je obslúžená v rodičovskom komponente ResultListComponent, ktorý obsahuje metódu povoľujúcu zobrazenie šablóny.

AttachmentWindowComponent



Obr. 19 Komponent AttachmentWindowComponent

Tento komponent obsahuje `<iframe>`, ktorého atribút `src` je nastavený na adresu docs.google.com, do ktorej sa vkladá aj URL dokumentu, ktorý chceme zobraziť. Komponent je nastavený aby prekryl celú aplikáciu. Po kliknutí na tlačidlo „Zavřít“ sa okno zavrie.

ScrollOnTopComponent

Obsahuje HostListener, ktorý sleduje udalosti na host elementoch, konkrétne udalosť scrollovania. Ak zachytí udalosť scrollovania dole zobrazí sa šípka na pravom boku, ak hore šípka zmizne. Kliknutím na šípku sa zavolá metóda `scrollOnTop()` , ktorá scrolluje cez dokument až k hlavičke.

5.4.2 Služby

Aplikácia obsahuje jedinú service triedu a to SearchService.

SearchService

Táto trieda služby obsahuje dve členské metódy a to `advancedSearch` pre pokročilé vyhľadávanie a `simpleSearch` pre jednoduché. Metódy sú volané v triede komponentu SearchInputService a ako vstupné hodnoty preberajú textové reťazce v poli vyhľadávania. V tele funkcie je objekt v JSON formáte, ktorý je zložený z vstupných parametrov vyhľadávanie, ale nepredstavuje ešte samotné dotazy na databázu. Tento JSON je potom prekonvertovaný na string a poslaný v tele požiadavky pomocou metódy POST. Server odpovie objektom typu Observable.

5.4.3 Pipes

V aplikácie boli využité aj pipes. Pipe objekty sú triedy, ktoré preberajú dáta ako výstup a transformujú ich na požadovaný výstup. V ukážkovej aplikácii sa nachádzajú dve triedy s Pipe dekoratérom.

LongTextPipe

Táto pipe sa stará o to aby sa vkladaný text nevypisoval celý ale len jeho prvých 100 znakov. Keďže zmluvy majú často krát dlhé názvy a nezmestili by sa do položiek zoznamu zmlúv, je použitá táto pipe.

ValuePipe

Sumy zmlúv sú v dokumentoch uložené v číselnom formáte, bez oddelených tisícov. Táto pipe preberá tieto hodnoty a vkladá medzi po každých troch čísliciach, pre lepšiu čitateľnosť sumy.

5.5 Node.js aplikácia

Serverová aplikácia je rozdelená do dvoch častí, do súborov server.js a api.js.

V súbore server.js sa najprv pomocou direktívy `require()` načíta HTTPS modul a uloží sa inštancia HTTPS do premennej. Nad touto inštanciou je volaná metóda `http.createServer()` pre vytvorenie inštancie serveru, ktorá bude sledovať určitý port.

5.6 Express.js aplikácia

Keďže Express je Node.js framework, ktorý využíva serverová aplikácia je tiež rozdelený do oboch súborov. Aplikačný framework, vytvára REST API pre komunikáciu s klientskou Angular aplikáciou.

Vytvorená inštancia Express má nastavený port, na ktorom beží server. Pomocou metód get a post, možno zachytávať požiadavky. Na základe cesty sa rozhoduje, aká akcia bude vykonávaná, a ako server odpovie na požiadavky. Ak bude cesta začínať `/search`, bude sa smerovať na post metódy podľa toho či ďalej bude nasledovať `/adv` pre pokročilé vyhľadávanie, alebo `/sim` pre jednoduché vyhľadávanie. V oboch prípadoch sa budú volať callback metódy, tie vo svojom tele obsahujú kód pre poskladanie dotaz v JSON formáte, ktorý bude posielané na ElasticSearch klienta. Keď tento klient odpovie vráti nájdene dokumenty a tie budú poslané v tele odpovedi.

5.7 ElasticSearch

Dáta sú uložené v clusteroch pozostávajúcich z jedného alebo viacerých uzlov, inak povedané serverov. Clustery takto držia všetky dáta a poskytujú indexovanie a vyhľadávanie naprieč všetkými uzlami. Uzol predstavuje jeden server, ktorý je súčasťou clusteru. Index je kolekcia dokumentov, ktoré majú spoločnú charakteristiku. Indexy sú definované menom, a toto označenie sa používa pri vykonávaní indexovania, vyhľadávania, aktualizácie alebo odstraňovaní dokumentov v tomto indexe. Cluster môže obsahovať neobmedzené množstvo indexov. Reprezentácia samotných dokumentov je podobná ako u MongoDB teda formou dokumentov.

Ukážková aplikácia obsahuje len jeden typ a to zmluvy. Tie sú zoskupené pod jediným indexom „contracts“. Taktiež je využívaný jediný cluster s len jedným uzlom. Dokument zmluvy obsahuje informácie, o názve zmluvy, zmluvných stranách, URL adresy príloh a samotný obsah zmluvy.

6 ZHODNOTENIE POUŽITÝCH TECHNOLÓGIÍ

Za cieľom vytvoriť aplikáciu, ktorá bude jednoduchá na vývoj, bol použitý balíček MEAN stack, ktorý zjednodušuje vývoj z viacerých hľadísk.

Táto kapitola obsahuje zhodnotenie technológii použitých v aplikácií z viacerých hľadísk, ktorými sú:

- Z hľadiska požiadaviek na vývojový tým
- Z hľadiska rýchlosti implementácie
- Z hľadiska udržateľnosti a rozšíriteľnosti

6.1 Zhodnotenie z hľadiska požiadaviek na vývojový tým

Z hľadiska požiadaviek na vývojový tým je použitie MEAN výhodou, keďže všetky časti tohto balíčku používajú jediný programovací jazyk JavaScript. Tým pádom v prípade jedného vývojára stačí aby dostatočne ovládal tento jazyk, alebo v prípade tímovej práce na zložitejších projektoch, netreba vyhľadávať expertov pre PHP a JavaScript alebo špecialistu na front-end a odborníka na back-end ako je tomu u LAMP prístupu. Aj keď Angular 2 aplikácie sa píšu v jazyku TypeScript, táto skutočnosť nezvyšuje výrazne nároky na vývojový tým, keďže možnosť písania v objektovo orientovanom jazyku je skôr výhodou a všetci programátori by mali poznať objektové programovanie.

Pri vývoji Angular Aplikácie bolo použité prostredie príkazového riadku Angular CLI. Toto prostredie uľahčuje vývojárom prácu, tým že pomocou jednoduchých príkazov je možné vytvoriť projekt, pridávať komponenty, pipes, služby a iné. Ďalším využívaným nástrojom bol balíkový systém npm, ktorého balíky boli použité aplikáciou. To uľahčilo vývoj, keďže nebolo nutné vytvárať vlastné knižnice, ako napríklad modul pre komunikáciu s ElasticSearch klientom, jQuery modul a iné.

Prechod na MEAN stack prináša vývojovému tímu množstvo výhod, z ktorých tri najvýznamnejšie je jeden jazyk, flexibilita v platforme nasadenia a zvýšená rýchlosť pri vyhľadávaní údajov. Avšak, tento prechod nie je bez kompromisov. Akýkoľvek existujúci kód bude musieť byť prepísaný do jazyka JavaScript.

6.2 Zhodnotenie z hľadiska rýchlosti implementácie

Z časového hľadiska je vývoj pomocou týchto technológii časovo nenáročný, keďže technológie MEAN balíčku sú jednoduché na naučenie. Miernou nevýhodou vývoju Angular 2

je to, že sa jedná o relatívne nový framework, tým pádom nemá ešte vytvorenú veľkú komunitu vývojárov. Taktiež v prípade Angular 2 dokumentácie, by bolo vhodné túto dokumentáciu rozšíriť, keďže niektoré časti nie sú dostatočné popísané, a pri vývoji bolo nevyhnutné hľadať odpovede v odborných článkoch alebo fórach.

Webová aplikácia vytvorená pomocou MEAN stacku je možné implementovať v rámci akéhokoľvek operačného systému. Zatiaľ čo LAMP stack je viazaný na nejaký derivát operačného systému Linux, keďže všetky jeho hlavné zložky Apache, MySQL a PHP sú primárne vyvíjane pre Linux, MEAN nemá takéto obmedzenia. Tým pádom Linux môže byť vhodný aj pre aplikáciu postavenú na MEAN, ale nie je jedinou možnosťou. Akýkoľvek operačný systém, na ktorom je možné spustiť Node.js je možnou alternatívou. [38]

LAMP stack používa Apache server, pretože je to jedna z najstabilnejších dostupných možností. Služba MEAN používa server Node.js. MEAN stack môže znamenať lepší výkon aplikácie, pretože Node.js je úplne neblokujúci a založený na udalostiach, čo umožňuje skutočnú súbežnosť medzi požiadavkami. Toto je dôležité najmä v dnešných dňoch, keď je kladený dôraz na výkon aplikácie bežiacej na mobilnom zariadení. Ďalej, ako bolo spomínané, Node.js podporuje jediný back-end jazyk a to JavaScript. Apache má väčšiu podporu jazykov a teda existuje aj viacero bezplatných rozšírení.

Ako bolo spomínané MEAN stack primárne používa MongoDB, no pre ukážkovú aplikáciu bol použitý ElasticSearch. Ten je rovnako ako MongoDB nerelačnou databázou. Frontend aplikácie obsluhuje všetko v JSON ((JavaScript Objet Notation) formáte, preto je vhodné použiť dokumentovo orientovanú databázu. JavaScript dokáže v podstate akýkoľvek JavaScript objekt konvertovať do JSON formátu. Ak je kód na strane servera aj klienta napísaný v jazyku JavaScript, potom je rozumné používať databázu využívajúcu JSON formát. Týmto je možné výrazne redukovať čas písaním kódu, ktorý je potrebný na serializáciu a deserializáciu dát.

Relačné databázy s podporou veľmi komplexných štruktúrovaných dotazov na databázu umožňujú vykonávať komplexné operácie s údajmi. Chýbajúca schéma v NoSQL dokumentových databázach umožňuje definovanie rozdielnych objektov, ktoré nevyžadujú rozsiahle zmeny kódu a odstraňuje potreby rozsiahlych a komplikovaných dotazov, tým pádom systém často môže pracovať efektívnejšie ako podobná architektúra vytvorená na relačnej databáze.

6.3 Zhodnotenie z hľadiska udržateľnosti a rozšíriteľnosti

Aj keď v oblasti JavaScriptu vznikajú neustále nové technológie a vylepšenia, ako je aj využívaný framework Angular2 a jazyk TypeScript, tak stále sa jedná o JavaScript, ktorý nestráca podporu v prehliadačoch a v najbližších rokoch ani nestratí. Ako bolo spomínane Angular2 nemá spätnú kompatibilitu s predošlou verziou, no nová verzia Angularu vo verzii 4, je spätne kompatibilná. Angular 2 má podporu vo všetkých moderných prehliadačoch, no aj tak niektoré funkcie by nemuseli byť možné v starších ale v niektorých prípadoch aj najnovších prehliadačoch. Tieto problémy, by čiastočne mohli riešiť polyfills, čo sú vlastne knižnice, ktoré implementujú funkciu na webových prehliadačoch, ktoré túto funkciu nepodporujú. [37] Pri prechode na najnovšiu verziu by mohli nastať problémy, keďže aj pri vývoji boli pokusy aj o využívanie technológií z tejto najnovšej verzií, ako napríklad Angular's animation system, no jeho podpora nie je rozšírená na prehliadače Safari a Edge, čo je však tiež možno vyriešiť s pomocou polyfills.

Keďže Angular aplikácie sú založené na komponentoch, je možné ich jednoducho rozširovať vytváraním nových komponentov. Je vhodné riadiť sa filozofiou: všetko je komponent. Konkrétne v rámci ukážkovej aplikácie by mohol byť vytvorený komponent pre vyššie spomínane prihlasovanie, komponent so vstupnými poľami pre vytváranie nových dokumentov, komponent pre mazanie dokumentov a ich úpravu. Každý z komponentov by využíval prislúchajúce služby, ktoré by posielali požiadavky na server, pre vykonanie týchto činností.

Aplikácia obsahuje už viacero bezpečnostných prvkov, ktoré sú podrobnejšie popísané v ďalšej kapitole. To umožňuje aby aplikácia do budúcnosti mohla byť rozšírená o obrazovku na prihlásenie. Aplikácia už používa HTTPS server, čím je zabezpečené, že nie je možné sledovať komunikáciu medzi serverom a klientom, čiže by potencionálne bolo sťažené zistenie prihlasovacích údajov. Prihlásený používateľ by mohol pridávať nové dokumenty do databázy, upravovať ich a mazať.

Nerelačné databázy vynikajú pri spravovaní operačných údajov (ODS – operational data store), teda údajov z rozličných zdrojov. To v prípade našej aplikácie znamená, že ak by bolo nutné rozšíriť databázu o veľké množstvo dát, je možné tieto dáta rozdeliť na viaceré servery pomocou shardov, v prípade že by sa všetky dokumenty nezmestili na jeden server.

Node.js aplikácia beží v jednom vlákne, aby bolo možné použiť viacero jadier procesoru je potrebné vytvoriť viacero aplikačných serverov a distribuovať záťaž. O vyvažovanie záťa-

že sa bude starať Load Balancer, ktorý bude priraďovať požiadavky jednotlivým serverom. Toto by bolo možné využiť aj v ukážkovej aplikácií. Pri rastúcom počte požiadaviek na server, by bolo možné takto rozšíriť serverovú infraštruktúru, aby fungovala spoľahlivo a rýchlo.

7 BEZPEČENOSTNÉ ASPEKTY APLIKÁCIE

Táto kapitola zhodnotí aplikáciu po stránke bezpečnosti. Aplikácia je zabezpečená po viacerými spôsobmi. Medzi bezpečnostné opatrenia patria použitie HTTPS komunikačného protokolu, využitie zabudovanej sanitácia v Angulare, injektovanie DomSanitzer, použitie proxy serveru a iné.

7.1 HTTPS protokol

Aplikácia používa zabezpečený hypertextový prenosový protokol (HTTPS), ktorý je zabezpečenou verziou HTTP, teda spája HTTP protokol spolu s protokolom SSL. HTTPS protokol umožňuje bezpečnú komunikáciu v počítačovej sieti. HTTPS vytvára bezpečnostný kanál pre nezabezpečenú sieť využívajúci asymetrickú kryptografiu. To zaisťuje ochranu pred sledovaním a útokmi typu man-in-the-middle a tým pádom nie je možné zachytávať obsah požiadaviek a odpovedí klienta.

7.2 Prevencia proti cross-site scriptingom (XSS)

Angular obsahuje vlastný bezpečnostný model zabraňujúci cross-site-scripting útokom (XSS). Cross-site scripting je spôsob akým vkladať škodlivý kód do webových stránok. Je jedným z najčastejších typov útokov. Predstavuje spôsob, akým útočník môže vkladať škodlivý kód do webových stránok. Tento kód môže ukradnúť, najčastejšie prihlasovacie údaje.

Na zablokovanie XSS útokov, je potrebné zabrániť vloženiu škodlivého kódu do DOM. Bezpečnosť je ohrozená v prípade, že útočník môže vkladať HTML tag <script>, v rámci ktorého môže pustiť ľubovoľný kód, ďalej v prípade href atribútu v `<a href="javascript:">` elemente a iné.

Bezpečnostný model Angularu blokuje XSS chyby tak, že predvolene nastaví všetky hodnoty ako nedôveryhodné. Keď je táto hodnota vložená do DOM zo šablóny prostredníctvom property, atribútu, štýlu, class binding alebo interpoláciou, Angular ošetruje a deaktivuje hodnoty. Sanitácia je proces, pri ktorom sa nedôveryhodná hodnota transformuje, na hodnotu, ktorej vloženie do DOM je bezpečné a často sa zachová pôvodná hodnota.

Nasledujúca šablóna (*Zdrojový kód 11*) binduje hodnotu `htmlSnippet`, raz pomocou interpolácie do vnútra elementu a po druhý krát nastavením vlastnosti [innerHTML].

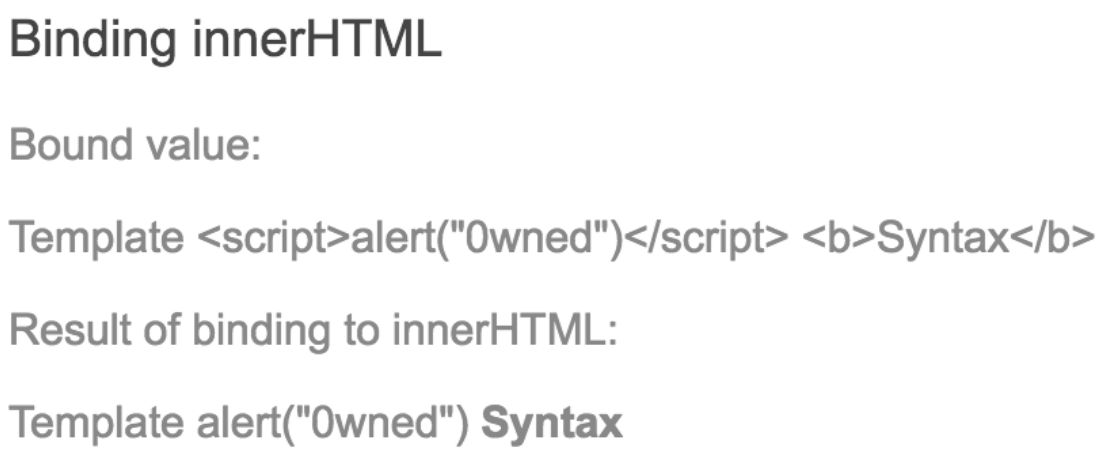
```
1. <h3>Binding innerHTML</h3>
2. <p>Bound value:</p>
3. <p class="e2e-inner-html-interpolated">{{htmlSnippet}}</p>
4. <p>Result of binding to innerHTML:</p>
5. <p class="e2e-inner-html-bound" [innerHTML]="htmlSnippet"></p>
```

Zdrojový kód 11 Šablóna bindujúca htmlSnippet [36]

```
1. export class InnerHtmlBindingComponent {
2. // For example, a user/attacker-controlled value from a URL.
3. htmlSnippet = 'Template <script>alert("0wned")</script>
   <b>Syntax</b>';
4. }
```

Zdrojový kód 12 Komponent s premennou htmlSnippet [36]

Vnútri `htmlSnippet` (*Zdrojový kód 12*) sa nachádza kód, ktorý by sa potencionálne mohol vykonať. Avšak Angular rozpozná hodnotu ako nebezpečnú a automaticky ju ošetrí tak, že odstráni `<script>` a `<b>` tagy ale zachová text vnútri elementov. Podobným spôsobom je ošetrené bindovanie do tela `<p>` elementu, kedy vypíše obsah celej premennej len ako text. Na obrázku (*Obr. 20*) vidíme ako boli ošetrené hodnoty. [36]



Obr. 20 Ukážka zabránenia XSS útoku [36]

7.3 Zabezpečenie pomocou DomSanitizer

Aplikácia tiež implementuje aj Angular API DomSanitizer, ktorý pomáha predchádzať XSS útokom, tým že ošetruje hodnoty, aby boli bezpečné pre použitie v iných DOM kontextoch. Aby bola premenná označená za dôveryhodnú, injektuje sa DomSanitizer a zavolá

sa jedna z metód `bypassSecurityTrust...`, do parametra ktorej sa vloží dôveryhodná hodnota.

V projekte bola použitá metóda `bypassSecurityTrustResourceUrl`, ktorá označí ako bezpečnú zdrojovú URL adresu, ktorá môže byť použitá na načítanie spustiteľného kódu, v `<script src>` alebo `<iframe src>`. To zabezpečí, že akákoľvek iná URL bude ošetrená a nebezpečný kód bude znefunkčnený.

V kóde nižšie (*Zdrojový kód 13*) je vidieť nedôveryhodnú hodnotu ukladanú do `dangerousUrl`, ktorá nebude bindovana do atribútu `src` a hodnotu ktorá je v parametre funkcie `bypassSecurityTrustUrl`, ktorá bude bindovná do `src`, lebo je označená ako bezpečná. [36]

```
1. constructor(private sanitizer: DomSanitizer) {
2. this.dangerousUrl = 'javascript:alert("Hi there")';
3. this.trustedUrl = sanitiz-
   er.bypassSecurityTrustUrl(this.dangerousUrl);
```

Zdrojový kód 13 Ukážka použitia DomSanitizer API [36]

7.4 Ďalšie bezpečnostné aspekty

Pre účely zachovania bezpečnosti aplikácie boli dodržiavané osvedčené postupy:

- Použitie posledných uvoľnených Angular knižníc. Tým pracujúci na Angulare pravidelne aktualizuje knižnice a opravuje chyby ktoré môžu predstavovať bezpečnostné riziko.
- Nepoužívanie modifikovaných kópii Angularu. Existujú upravené verzie frameworku Angular, ktoré môžu byť vhodnejšie pre použitie v určitých projektoch, no tieto verzie nemusia obsahovať dôležité opravy a vylepšenia zabezpečenia.
- Vyhýbanie sa Angular API ktoré bolo v dokumentácii označené ako „Bezpečnostné riziko“ (Security Risk).

K REST API ElastiSearch má prístup len Express aplikácia. Express si možno predstaviť aj ako proxy server, ktorý je prostredníkom medzi klientom a ElasticSearch (ES) serverom. Voči ES serveru vystupuje Express aplikácia ako klient, ktorý posiela dotazy na tento ES server. Tým pádom nie je toto REST API viditeľné zo strany klienta. Týmto je zabezpečené to, že je znemožnené vkladať, meniť a mazať dáta v ElastiSearch databáze. Ďalším stupňom ochrany proti nechceným CRUD operáciám je, že ElasticSearch API má vlastné

metódy pre každý typ dotazu na databázu, to znamená, že ak by sa telo NoSQL dotazu vložilo do metódy na mazanie, ES túto požiadavku nevykoná, lebo nepozná parametre v tomto dotaze.

Aby sa zabránilo posielaniu škodlivých dotazov na databázu, nie sú telá príkazov zostávané na strane klienta ale až na strane serveru. V tele požiadavky na server sa posielajú len parametre vyhľadávania. Keď sa táto požiadavka doručí na server, zostaví sa z nich dotaz, ktorý sa pošle na ElasticSearch klienta.

ZÁVER

Cieľom tejto diplomovej práce bolo oboznámenie sa s vývojovým frameworkom Angular2 a jeho použitie v rámci prístupu MEAN stack, ktorý okrem frameworku Angular obsahuje aj technológie Node.js, Express.js a databázu MongoDB. V teoretickej časti práce boli popísané všetky časti balíčka MEAN stack. Podrobnejšie bol rozoberaný spomínaný framework Angular2, kde sa práca zameriavala na jeho architektúru, a taktiež na jazyk TypeScript, v ktorom sú Angular2 aplikácie vyvíjané. V závere teoretickej časti bol spomenutý už starší prístup LAMP stack a jeho stručné porovnanie s prístupom MEAN.

Výstupom práce bola ukážková aplikácia, ktorá využíva prístup MEAN stack, kde namiesto databáze MongoDB bol použití engine ElasticSearch. Vývoj bol hlavne zameraný na front-end aplikáciu vo frameworku Angular2.

Pri vývoji aplikácie bola využitý npm module Angular-CLI. Táto aplikácia spustiteľná v príkazovom riadku, uľahčuje vývojárom prácu, pretože obsahuje viacero užitočných príkazov využívaných pri vývoji frontend aplikácie. Obsahuje príkazy pre vytvorenie nového projektu, pridávanie rôznych častí, testovanie, použitie lite serveru, buildovanie a iné. Keďže framework je relatívne nový, vývojár sa môže stretnúť s problémami, ako sú nedostatočne podrobná dokumentácia, slabšia komunita a chýbajúce moduly. Ďalšou novinkou, ktorá prichádza s Angularom vo verzii 2, je písanie v jazyku TypeScript. Ten je objektovo orientovaný, čo by pre vývojárov nemal byť problém, keďže tento prístup je veľmi rozšírený. Celkovo vývoj v tomto frameworku možno považovať za časovo nenáročný, dokonca aj v prípade vývojárov, ktorý začínajú s Angularom.

Takmer všetky funkcionality frameworku Angular 2 majú podporu v posledných verziách najpoužívanejších webových prehliadačov, no existujú aj také ktoré túto podporu nemajú. V tom prípade je možné použiť pollyfils, ktoré riešia túto nekompatibilitu.

Za výhodu možno považovať aj to, že Angula2 aplikácia je skladaná z modulov, ktoré možno spájať do seba ako puzzle a rovnako jednoduché je pridávanie nových modulov. Z hľadiska zabezpečenia aplikácie proti XSS útokom je tiež výhodou, to že Angular2 framework disponuje funkcionalitou, ktorá hrozbu týchto útokov potláča.

Aplikáciu by bolo možné ďalej rozšíriť o nové komponenty pre prihlasovanie sa a správu dokumentov, a obslužné metódy na strane front-endu, ako aj metódy spracúvajúce požiadavky o tieto činnosti. Aplikácia je už vopred zabezpečená použitím HTTPS serveru, čo by sťažovalo zistenie autentifikačných údajov v prípade man-in-the-middle útokov. Keďže sa

pracovalo s nerelačnou dokumentovou databázou, ktoré sú vhodné na spracovanie dát z rozličných zdrojov, bolo by možné v prípade nedostatočného miesta na jednom servery, databázu jednoducho rozdeliť na viacero serverov. Možnosťou rozšíriteľnosti na strane servera je aj použitie load balanceru, ktorý by prerozdeľoval požiadavky na jednotlivé Node.js servery. Toto riešenie by bolo vhodné v prípade veľkého množstva používateľov aplikácie.

ZOZNAM POUŽITEJ LITERATÚRY

[1] *MEAN Web Development*. Birmingham: Packt Publishing, 2014. ISBN 978-1-78398-328-5.

[2] *Building an E-Commerce Application with MEAN*. Birmingham: Packt Publishing, 2015. ISBN 978-1-78528-655-1.

[3] CHODOROW, Kristina. *MongoDB: the definitive guide*. Second edition. Sebastopol: O'Reilly, 2013. ISBN 978-1-4493-4468-9.

[4] Sharding. In: *MongoDB* [online]. c2008-2017 [cit. 2017-04-12]. Dostupné z: https://docs.mongodb.com/manual/sharding/

[5] MONGODB MANUAL. In: *MongoDB* [online]. New York: MongoDB, 2008 [cit. 2017-04-10]. Dostupné z: https://docs.mongodb.com/manual/replication/

[6] ELROM, Elad. *Pro MEAN Stack Development*. Apress, 2016. ISBN 978-148-4220-436.

[7] *Express.js Guide: The Comprehensive Book on Express.js*. CreateSpace Independent Publishing Platform, 2013. ISBN 978-1494269272.

[8] *MEAN Web Development*. Birmingham: Packt Publishing, 2014. ISBN 978-1-78398-328-5.

[9] ExpressJS - Middleware. In: *Tutorials Point* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-27]. Dostupné z: https://www.tutorialspoint.com/expressjs/expressjs_middleware.htm

[10] AngularJS - Overview. In: *Tutorials Point* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-04-26]. Dostupné z: https://www.tutorialspoint.com/angularjs/angularjs_overview.htm

[11] DICKEY, Jeff. *Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node.js*. San Francisco: Peachit Press, 2015. Develop and design. ISBN 01-339-3015-7.

[12] HEVERY, Miško. Sweet Spot for <angular/>. In: *Miško Hevery The Testability Explorer Blog* [online]. 2009 [cit. 2017-05-06]. Dostupné z: http://misko.hevery.com/2009/10/04/sweet-spot-for-angular/

[13] HAMILTON, Coman. Angular 2.0 announcement backfires. In: *Jaxenter* [online]. 2014 [cit. 2017-05-02]. Dostupné z: https://jaxenter.com/angular-2-0-announcement-backfires-112127.html

[14] EXBRAYA, Cédric. What's new in Angular 4? In: *Ninja-squad* [online]. St Just St Rambert, 2017 [cit. 2017-05-06]. Dostupné z: http://blog.ninja-squad.com/2017/03/24/what-is-new-angular-4/

[15] GECHEV, Minko. AngularJS in Patterns. In: *MINKO GECHEV'S BLOG* [online]. 2014 [cit. 2017-05-06]. Dostupné z: http://blog.mgechev.com/2014/05/08/angularjs-in-patterns-part-1-overview-of-angularjs/

[16] MULLE, Eelco. Angular 2 vs. Angular 1: Key Differences. In: *DZone* [online]. DZoone, 2015 [cit. 2017-05-06]. Dostupné z: https://dzone.com/articles/typed-front-end-with-angular-2

[17] Node.js - Introduction. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-08]. Dostupné z: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

[18] KAR, Shubhra. Node.js Performance Tip of the Week: Event Loop Monitoring. In: *StrongLoop* [online]. Foster City: StrongLoop, 2014 [cit. 2017-05-08]. Dostupné z: https://strongloop.com/strongblog/node-js-performance-event-loop-monitoring/

[19] KIRAN, Ravi. Node.js Tutorial Series - Getting Started. In: *DotNetCurry* [online]. Knowledge Visuals Pvt., 2015 [cit. 2017-05-19]. Dostupné z: http://www.dotnetcurry.com/nodejs/1143/nodejs-tutorial-series-beginner-experienced-developer

[20] What is npm? In: *Npm Documentation* [online]. 2017 [cit. 2017-05-09]. Dostupné z: https://docs.npmjs.com/all

[21] ROZENTALS, Nathan. *Mastering TypeScript*. Packt Publishing, 2015. ISBN 978-1784399665.

[22] TEMPLIER, Thierry. Angular 2, decorators and class inheritance. In: *Medium* [online]. A Medium Corporation, 2016 [cit. 2017-05-10]. Dostupné z: https://medium.com/@ttemplier/angular2-decorators-and-class-inheritance-905921dbd1b7

[23] NGMODULES. In: *Angular* [online]. Google, c2017 [cit. 2017-05-13]. Dostupné z: https://angular.io/docs/ts/latest/guide/ngmodule.html

[24] Architecture Overview. In: *Angular* [online]. Google, c2017 [cit. 2017-05-10]. Dostupné z: https://angular.io/docs/ts/latest/guide/architecture.html

[25] Angular 2 - Modules. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-13]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_modules.htm

[26] Angular 2 - Components. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-13]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_components.htm

[27] MURRAY, Nate, Ari LERNER, Felipe COURY a Carlos TABORDA. *Ng-Book 2: The Complete Book on Angular 2*. Fullstack.io, 2016. ISBN 978-0991344611.

[28] FAIN, Yakov a Anton MOISEEV. *Angular 2 Development with TypeScript*. Manning Publications, 2016. ISBN 9781617293122.

[29] Angular 2 - Services. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-12]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_services.htm

[30] Angular 2 - Dependency Injection. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-13]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_dependency_injection.htm

[31] Using Two-Way Data Binding. In: *Rangle.IO* [online]. Toronto: Rangle, c2017 [cit. 2017-05-14]. Dostupné z: https://angular-2-training-book.rangle.io/handout/components/app_structure/two_way_data_binding.html

[32] Angular 2 - Directives. In: *TutorialsPoint* [online]. Hyderabad: Tutorials Point, c2017 [cit. 2017-05-11]. Dostupné z: https://www.tutorialspoint.com/angular2/angular2_directives.htm

[33] MCLAUGHLIN, Brett. *PHP & MySQL the missing manual*. 2nd ed. Sebastopol,CA: O'Reilly Media, 2012. ISBN 978-144-9355-548.

[34] ROUSE, Margaret. LAMP (Linux, Apache, MySQL, PHP). In: *SearchEnterprise Linux* [online]. Newton: SearchEnterpriseLinux, c2017 [cit. 2017-05-12]. Dostupné z: http://searchenterpriselinux.techtarget.com/definition/LAMP

[35] Observables. In: *Rangle.IO* [online]. Toronto: Rangle, c2017 [cit. 2017-05-14]. Dostupné z: https://angular-2-training-book.rangle.io/handout/observables/

[36] Security. In: *Angular* [online]. Google, c2017 [cit. 2017-05-13]. Dostupné z: https://angular.io/docs/ts/latest/guide/security.html

[37] Browser Support. In: *Angular* [online]. Google, c2017 [cit. 2017-05-13]. Dostupné z: https://angular.io/docs/ts/latest/guide/browser-support.html

[38] HERSKOVITS, Itay. MEAN vs LAMP – How Do They Stack Up? In: *BackAnd* [online]. Palo Alto: BackEnd, 2014 [cit. 2017-05-15]. Dostupné z: http://blog.backand.com/mean-vs-lamp/

[39] The History. In: *AngularJS Ant the Search for the Best Front-End Framework* [online]. [cit. 2017-05-06]. Dostupné z: https://sites.google.com/a/maricopa.edu/angularjs_and_the_search_for_the_best_front_end_framework/the-history

[40] EFTAIHA, Diana. An Introduction to Apache. In: *Envato Tuts+* [online]. 2012 [cit. 2017-05-13]. Dostupné z: https://code.tutsplus.com/tutorials/an-introduction-to-apache--net-25786

[41] Getting Started | ElasticSearch. *Elastic* [online]. Mountain View: Elastic, c2017 [cit. 2017-05-14]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/5.1/getting-started.html

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

MEAN	MongoDB, Express.js, Angular, Node.js
LAMP	Linux, Apache, MySQL, PHP
SPA	Single page application
SQL	Structured Query Language
PHP	Personal Home Page alebo Hypertext Preprocessor
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
URL	Uniform Resource Locator
CGI	Common Gateway Interface
XSS	Cross- Site Scripting
SSL	Secure Sockets Layer
ODS	Operational data store
DOM	Document Object Model
API	Application Programming Interface
npm	Node Package Manager
CLI	Command Line Interface
IČO	Identifikačné číslo organizácie
CSS	Cascading Style Sheet
TypeScript	Typewitten JavaScript
Perl	Practical Extraction and Report Language
DB	DataBase
ES	ElasticSearch

CRUD	Create Read Update Delete
REST	REpresentational State Transfer
jQuery	Cross-platform JavaScript knižnica
I/O	Input/ Output
TCP	Transmission Control Protocol
IP	Internet Protocol
MVC	Model View Controller

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM ZDROJOVÝCH KÓDOV