VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Inovace a optimalizace frontendu systému řízení vztahů se zákazníky
Inovation and Optimization of Customer Relationship Management System Frontend

Student: Bc. Ondřej Pešat
Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2020

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Ondřej Pešat**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T017 Informatika v ekonomice

Téma: Inovace a optimalizace frontendu systému řízení vztahů se zákazníky

Inovation and Optimization of Customer Relationship Management System Frontend

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoretická a metodická východiska v oblasti vývoje SPA webových aplikací
3. Analýza a popis výchozího stavu CRM systému ve firmě
4. Návrh a implementace vybraného řešení
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

BEAN, Martin. *Laravel 5 Essentials.* Birmingham: Pack Publishing, 2015. ISBN 978-1785283017.
FREEMAN, Adam. *Pro Angular 6 Third Edition.* New York: Apress, 2018. ISBN 978-1484236482.
LEHTINEN, Jarmo. *Aktivní CRM: řízení vztahu se zákazníky.* Přeložil Alena SVOZILOVÁ. Praha: Grada Publishing, 2007. ISBN 978-80-247-1814-9.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 22.11.2019
Datum odevzdání: 24.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Ekonomická fakulta 1 OSTRAVA

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

doc. Ing. Lenka Kauerová, CSc.
proděkanka pro studium
na základě pověření k jednání č.j.
VSB/19/050319/9900 ze dne 24. 9. 2019

Prohlašuji, že jsem celou diplomovou práci, včetně všech příloh, vypracoval/a samostatně.

V Ostravě dne 21.4.2020

.....

Bc. Ondřej Pešat

Rád bych poděkoval vedoucímu mé diplomové práce panu Ing. Vítězslavu Novákovi, Ph.D. z Katedry aplikované informatiky za odborné rady a konzultace, které mi poskytl během zpracování této diplomové práce.

Obsah

1 Úvod ... 6
2 Teoretická a metodická východiska v oblasti vývoje SPA webových aplikací ... 8
2.1 Pojem SPA – výhody a nevýhody ... 8
2.1.1 Výhody ... 9
2.1.2 Nevýhody ... 10
2.2 Přehled nástrojů pro vývoj SPA ... 11
2.2.1 JavaScript ... 11
2.2.2 TypeScript ... 12
2.2.3 Vue.js ... 13
2.2.4 React.js ... 15
2.2.5 Angular ... 16
2.3 Výběr frameworku pro vývoj SPA frontendu ... 18
2.4 Teoretická příprava pro vývoj v Angularu ... 19
2.4.1 Moduly ... 20
2.4.2 Komponenty ... 22
2.4.3 Služby ... 28
2.4.4 Observables a RxJS ... 31
2.5 Agilní metodiky vývoje SW ... 33
2.5.1 Stručná historie agilních metodologií ... 33
2.5.2 Hodnoty agilního softwaru ... 34
2.5.3 Principy agilního softwaru ... 35
2.6 Agilní metodiky vývoje SW – Kanban ... 36
2.6.1 Stručná historie Kanbanu ... 36
2.6.2 Podstata metody Kanban ... 37
2.6.3 Zásady a postupu metody Kanban ... 37
2.6.4 Koncept metody Kanban ... 37
2.6.5 Koncept Toku ... 38
2.6.6 Příklady využití ... 38
2.6.7 Kanban v IT ... 38
2.6.8 Kanban v produktovém vývoji a agilním softwaru ... 39
3 Analýza a popis výchozího stavu CRM systému ve firmě ... 40
3.1 Serverová architektura ... 40
3.2 Databáze ... 40
3.3 Backend ... 41
3.3.1 PHP a Laravel ... 41

3.4 Frontend ... 42
3.4.1 Architektura ... 42
3.4.2 Funkcionalita ... 43
3.5 Uživatelé systému ... 45
3.5.1 Externí uživatelé – klienti společnosti ... 45
3.5.2 Interní uživatelé – Obchodní zástupci ... 47
3.5.3 Interní uživatelé – Administrativní pracovníci ... 49
3.5.4 Interní uživatelé – Management ... 50
3.6 Požadavky na optimalizaci a nové funkce ... 52
3.6.1 Požadavky na nové funkce ... 52
3.6.2 Optimalizace stávajících funkcí ... 53
3.6.3 Požadavky na serverové zázemí ... 54
4 Návrh a implementace vybraného řešení ... 55
4.1 Průběh vývoje ... 55
4.2 Návrh a kódování designu ... 55
4.3 Návrh modulů a komponentů ... 57
4.3.1 AppModule – Kořenový modul ... 57
4.3.2 CoreModule – Jádro systému ... 57
4.3.3 SharedModule – sdílené zdroje pro celou aplikaci ... 58
4.3.4 AuthModule – modul pro autentizaci ... 59
4.3.5 ProfileModule – profil uživatele ... 59
4.3.6 InvestmentModule – modul pro práci s investicí ... 60
4.3.7 DocumentModule – modul s dokumenty ... 60
4.3.8 DashboardModule – modul s nástěnkami ... 61
4.3.9 StructureModule – modul pro hierarchickou strukturu ... 61
4.3.10 ReservationModule – modul pro rezervaci kanceláří ... 62
4.3.11 WallModule – zeď s příspěvky ... 62
4.3.12 AffiliateModule – modul pro affiliate provize ... 63
4.3.13 BeneficiaryModule – modul pro kontaktní osoby ... 63
4.3.14 NotificationModule – modul pro notifikace ... 63
4.3.15 FaqModule – modul pro troubleshooting ... 63
4.3.16 TrainingsModule – modul se školeními ... 64
4.3.17 Lazy-loading modulů ... 64
4.3.18 Měření rychlosti načítání modulů ... 65
4.4 Optimalizace struktury ... 65
4.4.1 Příprava databáze ... 66
4.4.2 Implementace na frontendu ... 68

4.4.3 Měření rychlosti načítání struktury ... 69
4.5 Vylepšení správy plateb ... 69
4.5.1 Optimalizace příchozích plateb ... 70
4.5.2 Implementace odesílání dávek platebních příkazů ... 70
4.6 Implementace asynchronní fronty pro procesy běžící na pozadí aplikace ... 72
4.7 Implementace elektronických podpisů ... 73
4.8 Implementace Progressive Web Apps ... 74
4.8.1 Přidání PWA funkcí ... 75
4.8.2 Implementace notifikací ... 76
5 Závěr ... 78
Seznam použité literatury ... 80
Seznam zkratek ... 82

1 Úvod

Soubor hmotných a nehmotných složek podnikání, čímž jakýkoli podnik bezesporu je, musí neustále zlepšovat své techniky a dovednosti, aby si na trhu udržel konkurenceschopnost. Dobře zvládnuté procesy a snížení administrativní zátěže přispívá ke snížení nákladů a plynulejšímu průběhu procesů v podniku. V české investiční společnosti, ve které byla tato diplomová práce realizována, se o automatizaci procesů a snížení administrativní zátěže stará komplexní webová aplikace, kterou bych si díky její složitosti, počtu uživatelů a řešeným procesům, nebál nazvat informačním systémem. Tento systém se dělí na dvě logicky související části. Tzv. „Office“ část slouží pro klienty a obchodní zástupce společnosti. Druhá část systému slouží pro práci administrativních pracovníků a pro řízení vztahů se zákazníky (Customer Relationship Management).

Zub času je ovšem neúprosný a zejména v odvětví informačních technologií způsobuje stárnutí nejenom hardwaru a softwaru jako takového, ale i způsobu jakým jsou tyto nedílné součásti každého chytrého zařízení a počítačů produkovány. Výjimkou nebyla ani tato relativně mladá a perspektivní investiční společnost, jejíž webová aplikace pro správu všech investic, klientů a jiných důležitých proměnných, začínala nevyhovovat zejména z hlediska rychlosti, návrhu a škálovatelnosti. Společnost se díky výhodnému investičnímu fondu rychle rozvíjela a rozrůstala. Počet klientů se neustále zvyšoval a s ním i množství dat, které musí společnost zpracovávat a uchovávat. Dalším problémem byla škálovatelnost. Architektura systému již nedovolovala jednoduché přidávání nových funkcí a programovému kódu chyběla dokumentace.

Vedení společnosti se rozhodlo, že aplikaci je nutno inovovat a optimalizovat. Inovací bylo myšleno využití moderních technologií a principů programování. Optimalizací bylo primárně myšleno zrychlení aplikace.

Cílem mé diplomové práce je inovace a optimalizace frontendové strany této webové aplikace pro správu investic. Jelikož jsem v této společnosti zaměstnán jako frontend vývojář, a byl jsem najat právě na tento projekt, účastnil jsem se celého procesu návrhu a vývoje. V této práci se částečně dotkneme i backendové strany aplikace a krátce se seznámíme i s architekturou aplikace jako celku. O vývoj a správu původní aplikace se staral pouze jeden externí programátor. S růstem společnosti postupně přibývali i zaměstnanci. Podstatně se rozrostlo i IT oddělení. Tyto lidské zdroje se primárně alokovaly na vývoj a optimalizaci aplikace, přičemž pro řízení procesů byla

implementována agilní metodika vývoje programových systémů, čímž se budeme rovněž v této práci zabývat.

Největší část práce bude samozřejmě obsahovat práci s frontendovými technologiemi a principy návrhového vzoru SPA (neboli Single Page Applications). Jsou to zejména MVC Framework od společnosti Google – Angular a multi-paradigmový programovací jazyk JavaScript, respektive jeho nástavbě společnosti Microsoft – TypeScript.

2 Teoretická a metodická východiska v oblasti vývoje SPA webových aplikací

Největší prostor v této práci bude věnován Single Page aplikacím a principům jejich vývoje. Tato technologie totiž sehrála klíčovou roli při vývoji a optimalizaci nového frontendu systému. Vedle modulárnosti systému nám tato technologie pomůže optimalizovat uživatelovu workflow a připravit mu tak o mnoho lepší uživatelský zážitek. To je koneckonců jedním z hlavních účelů Single Page aplikací jako takových.

Populárnost SPA dokazuje výskyt nových programovacích nástrojů a frameworků, které se v posledních letech objevily. Mezi tu nejznámější trojici patří bezpochyby Angular, React a Vue.js. Všechno jsou to frameworky postavené na programovacím jazyce JavaScript (resp. ECMAscript), nebo jeho aplikacích. To umožnilo snadnou migraci vývojářům na tyto frameworky. Neměli totiž problém se naučit jejich principy a syntaxi, protože JavaScript je jeden z hlavních programovacích jazyků k tvorbě webových aplikací. (Mikowski, 2014)

2.1 Pojem SPA – výhody a nevýhody

Dnes je většina návštěvníků webových stránek nebo webových aplikací rozmazlená a nejsou většinou ochotni čekat více než dvě sekundy na načtení stránky.

Single Page Applications jsou typem webových aplikací nebo webových stránek, které načítají své části bez obnovení stránky a načtení nového požadovaného obsahu ze serveru. SPA tedy interagují s uživatelem dynamickým přepsáním aktuální stránky. SPA funguje a běží uvnitř webového prohlížeče a jejich účelem je poskytovat vynikající UX (User Experience) tím, že se snaží imitovat přirozený pocit desktopových aplikací v prostředí webového prohlížeče. Jak již název napovídá, tak tyto aplikace se skládají pouze z jedné stránky, kterou navštívíme, a která později načítá všechen ostatní obsah pomocí programovacího jazyka JavaScript, bez jeho pomoci by nemohly fungovat. (Mikowski, 2014)

SPA tvoří HTML strukturu stránky a data nezávisle na sobě, a poté se vykreslují přímo ve webovém prohlížeči. Tohoto chování můžeme dosáhnout díky několika netriviálním JavaScriptových frameworkům, jako Angular, Vue.js nebo React. (Mikowski, 2014)

Drtivá většina z nás se s takovými aplikacemi již setkala a používá je ve svém každodenním životě, ať už se jedná o naši oblíbenou streamovací službu, emailový klient nebo sociální síť. Typickými příklady těchto aplikací jsou:

- Gmail,
- Google Maps,
- Facebook,
- Twitter,
- GitHub,
- Google Drive,
- Youtube.

2.1.1 Výhody

User Experience

Jedna z hlavních výhod SPA. Vysoká rychlost a celkový průběh aplikace velmi dobře napodobuje pocit, jaký dostáváme při pohybu v desktopové nebo mobilní aplikaci, neboť veškeré potřebné zdroje aplikace (HTML, CSS a skripty) jsou načteny pouze jednou za celý životní cyklus aplikace. (Mikowski, 2014)

SPA jsou celé spouštěny ve webových prohlížečích a pokud nevyžadují externí zdroje dat, tak mohou po načtení běžet i v režimu offline, neboť moderní prohlížeče mají velmi propracované systémy cachování. (Mikowski, 2014)

Rychlost

Jelikož SPA nepotřebují obnovovat celou stránku, ale jen požadovaný obsah, tak u nich pozorujeme výrazné zvýšení rychlosti oproti klasickým MPA (Multi Page Applications) aplikacím. A protože jsou všechny zdroje za celý životní cyklus aplikace načteny pouze jednou, prohlížeč si se serverem pouze vyměňuje data tam a zpátky. (Mikowski, 2014)

Jednodušší debugging

Je mnohem jednodušší debugovat SPA, neboť ze své podstaty má všechen zdrojový kód na jedné stránce. SPA frameworky obvykle obsahují své vlastní vývojářské nástroje, což ještě více usnadňuje debugovací proces. (Mikowski, 2014)

Offline funkčnost

Po načtení obsahu SPA je možno ji uložit do paměti cache v lokálním úložišti a tímto lze poskytnout funkčnost aplikaci uživateli i offline. (Mikowski, 2014)

Mobile-Friendly

V dnešní době velké procento uživatelů prohlíží web z mobilních zařízení. Frameworky používány k vývoji SPA většinou umožňují vytvoření mobilní aplikace, která bude využívat stejný zdrojový kód, ale půjde spustit na mobilním telefonu nebo jiném mobilní zařízení. (Mikowski, 2014)

2.1.2 Nevýhody

Optimalizace SEO

SEO neboli Search Engine Optimization je důležitou součástí většiny webů, protože pomáhá potencionálnímu uživateli náš web najít. SPA jsou celé operovány pomocí JavaScriptu a posílají požadavky o výměnu dat rovnou z klientovy strany, což je důvod proč SPA poskytují slabou podporu SEO. (Mikowski, 2014)

Memory management

Při vývoji SPA je použito velké množství JavaScriptového kódu a při běhu je většinou manipulováno se složitými datovými toky. Při práci s těmito datovými toky a obecně s SPA je nutno brát zřetel na potencionální memory leaky, neboli stav, kdy nyní již nepotřebná operační paměť využita aplikací není navrácena zpět operačnímu systému. (Mikowski, 2014)

Inicializační načtení aplikace

SPA vykazují delší inicializační načtení, neboť klient si musí do svého prohlížeče nechat stáhnout celý zdrojový kód aplikace.

Existují metody, jak tento problém zmírnit, jako například Lazy-loading, avšak i po této optimalizaci nebude načtení tak rychlé jako u tradičních webových aplikací nebo stránek. (Mikowski, 2014)

2.2 Přehled nástrojů pro vývoj SPA

2.2.1 JavaScript

Primárním nástrojem pro vývoj SPA aplikací je bezpochyby multiplatformní, objektově orientovaný, událostmi řízený skriptovací programovací jazyk JavaScript.

Bevacqua (2017) uvádí, že JavaScript je flexibilní a výkonný programovací jazyk, který je akceptován konzistentně všemi webovými prohlížeči. Spolu s HTML a CSS, je jádrem webových technologií. Zatímco HTML je zodpovědné za strukturu a CSS je zodpovědné za styl a design, JavaScript poskytuje webové stránce interaktivitu.

Pravdou však zůstává, že JavaScript se nikdy neměl stát základním kamenem moderního webového vývoje. Ve skutečnosti byl tento jazyk vytvořen za pouhé dva týdny za úplně jiným účelem. (Bevacqua, 2017)

Podle autora Bevacqua (2017) popularita tohoto programovacího jazyka, který se ještě pár lety zpátky používal na jednoduché funkcionality a úpravy na webových stránkách, nyní roste exponenciálně. Důvod tohoto růstu byla úprava standardů ECMAScript a přidání schopností s tímto jazykem vyvíjet a modifikovat webové aplikace různými metodami a způsoby, a dokonce i do nich zakomponovat real-time funkcionality. JavaScript na sebe upoutává pozornost programátorů nejen díky své jednoduchosti ale i kvůli efektivnosti. JavaScript se v posledních letech stal doslova hitem mezi vývojáři, protože změnil webové prohlížeče na aplikační platformy:

- JavaScript může být použit jak na front-endu tak i na backendu,
- JavaScript je standardizován, takže je často aktualizován o nové funkcionality,
- JavaScript pracuje s DOM (Document Object Model), neboli strukturou webu, kterou zobrazuje prohlížeč,
- JavaScript umožňuje webovým stránkám mít interaktivní prvky, jako například scroll transitions,
- JavaScript nabízí širokou škálu frameworků a knihoven, které pomáhají vývojářům tvořit komplexní aplikace jednodušeji.

Na přelomu tisíciletí velké společnosti jako Google začali experimentovat s použitím JavaScriptu při psaní backendu. Tato skutečnost pomohla společnostem využít webové developery pro psaní programového kódu v serverovém kontextu. (Bevacqua, 2017)

JavaScript, psaný na straně serveru, získal popularitu, protože umožnil potřebnou škálovatelnost, která je potřebná pro backendové služby a cloud-computing. V serverovém prostředí může JavaScript bez problému komunikovat s ostatními programovacími jazyky, ba dokonce i s databázemi. Webové prohlížeče mají enginy, které zpracují JavaScript velmi rychle, to vedlo k využití JavaScriptu na serveru. (Bevacqua, 2017)

Node.JS, nebo Node, je jedna z nejpopulárnějších verzí serverového JavaScriptu. Node byl použit pro psaní obrovských platforem jako například eBay. Node umožňuje vývoj vysoce škálovatelných webových aplikací, platforem pro chatování a multiplayerových her. (Bevacqua, 2017)

2.2.2 TypeScript

Společnost Microsoft navrhla TypeScript s ohledem na specifické architektonické parametry, které umožňují TypeScript plně a snadno se integrovat s existujícím JavaScriptovým kódem a zároveň poskytnout robustní funkce mimo JavaScript. (Freeman, 2019)

Jak už jsme několikrát zmínili, TypeScript je nástavba nad JavaScriptem. Tento vztah umožňuje jazyku TypeScript porozumět a pracovat s jakýmkoli programovým kódem, který je validním JavaScriptovým kódem. Jakýkoli platný JavaScriptový kód je validním TypeScript kódem s pár výjimkami. (Freeman, 2019)

JavaScript je nesmírně shovívavý k hodnotě přiřazené do proměnné a nevytváří žádná pravidla pro danou proměnnou, ani pro konstruktory, kteří ji používají, neboť je to programovací jazyk s dynamickým typováním. V prostředí JavaScriptu předání číselného argumentu funkci, která očekává řetězec, negeneruje v průběhu vývoje žádné chyby, avšak při běhu programu může způsobit nepříjemné události, když funkce nebude schopna tento argument využít. (Freeman, 2019)

Aby se předešlo těmto runtime problémům, tak byl TypeScript navržen jako striktně typovaný jazyk, který provádí statickou kontrolu datových typů v průběhu kompilace do JavaScriptu. Kvůli flexibilitě je kontrolování datových typů nepovinné, nicméně většina klíčových výhod TypeScriptu s touto kontrolou úzce souvisí a je to také hlavní důvod proč TypeScript při vývoji použít. (Freeman, 2019)

Nová syntaxe a nová klíčová slova nám pomohou více a jednodušeji aplikovat principy a postupy objektově orientovaného programování. TypeScript nám totiž

poskytuje třídy, rozhraní a moduly k tomu, abychom mohli správně a přehledně strukturovat náš programový kód a zapouzdřit v něm znovupoužitelné struktury, které umožní jednoduchou udržitelnost a škálovatelnost. V rámci tříd jsme dokonce schopni definovat úroveň viditelnosti jejich datových položek a metod pomocí vestavěných modifikátorů – public, private a protected. (Freeman, 2019)

V rámci vývoje v programovacím jazyce TypeScript pracují vývojáři ve dvou rozdílných kontextech – návrh a spuštění. V kontextu návrhu používáme TypeScript přímo pro psaní naší aplikace. Protože TypeScript není podporován žádným z webových prohlížečů, je potřeba, aby se z něj stal validní JavaScriptový kód. Bez tohoto procesu bychom nemohli náš „návrhový kód" spustit. V kontextu spuštění musí být tedy všechen TypeScript kód překompilován do JavaScriptu a poté spuštěn na cílové platformě – například ve webovém prohlížeči. Webovému prohlížeči není poskytnuta informace, že kód je zkompilován – vypadá jako čistý JavaScript a není problém ho v prohlížeči spustit. (Freeman, 2019)

Aby webovému prohlížeči byl poskytnut validní spustitelný kód, TypeScript kompilátor zpracuje prvky jazyka TypeScript a implementuje je tak, aby mohly existovat v kontextu jazyka JavaScript. Jak již víme, v programovacím jazyce TypeScript existují prvky (kontrola datových typů a rozhraní), které nemohou být do JavaScriptu převedeny. Tyto nepodporované prvky jsou ze zkompilovaného kódu jednoduše odstraněny. Tento proces je známý jako typové smazání. Jejich odstranění nemá žádný dopad na funkcionalitu zkompilovaného kódu, protože tyto unikátní prvky jsou zde jen pro zjednodušení vývojového procesu a nepřepisují nic v jádře programovacího jazyka JavaScript. (Freeman, 2019)

2.2.3 Vue.js

Vue se stal nejpopulárnějším frontend GitHub projektem v roce 2018 se svými 117 tisíci hvězdami a více než 29 tisíci forky. Vytvořil ho Evan You, který pracoval na mnoha projektech ve společnosti Google, při kterých využíval Angular.js. Byl vytvořen jakožto odlehčený konkurent Angularu a ve své podstatě nabízí k němu příjemnou alternativu. (Copes, 2018)

Jeden z hlavních důvodů, proč vývojáři mají tento framework rádi je ten, že je velmi progresivní. To znamená, že se elegantně adaptuje na potřeby vývojáře. Dále může

být jednoduše integrován s Vaší aplikací pomocí HTML tagu `<script>`. (Copes, 2018)

Evan You se při vývoji Vue.js snažil vzít to nejlepší z hlavních frameworků, které byly na trhu dostupné (React.js, Knockout, Angular.js, …). Zkusme se nyní podívat na výhody a nevýhody při vývoji v tomto frameworku. (Copes, 2018)

Výhody frameworku Vue.js jsou:

- **velikost** – Tato vlastnost je jedno z největších lákadel. Produkční build Vue frameworku je překvapivě velmi malý – pouze 18KB po zazipování. To propůjčuje požadovanou flexibilitu a rychlost, kterou, když ji porovnáme s ostatními frameworky, tak staví Vue do pozice jasného vítěze. Zároveň je celý ekosystém malý a rychlý proto, že dovoluje uživateli separovat tzv. Template-to-Virtual-DOM kompilátor a také runtime,
- **integrační schopnost** – Vue také nabízí jednu z nejlepších integračních schopností, neboť může být použit na Single Page aplikace i na komplexní webové aplikace. Malé interaktivní části frameworku mohou být jednoduše integrovány do dalších frameworků/knihoven jako Django, Laravel, a WordPress,
- **škálovatelnost a všestrannost** – Můžete použít Vue buď jako knihovnu nebo jako celistvý framework. Vue může být také jednoduše použit na vývoj velkých znovupoužitelných šablon,
- **adaptabilita** – Většina vývojářů, kteří studují Vue, většinou nemá výraznější problémy s pochopením základních principů této technologie. Tato studijní perioda bývá většinou krátká díky podobnostem s ostatními frameworky,
- **čitelnost** – Vue je relativně jednoduchý na čtení a pochopení, protože funkce v něm jsou vysoce přístupné. Navíc, správa HTML bloků se dá optimalizovat s použitím ostatních komponent.

Nevýhody frameworku Vue.js jsou:

- **nedostatek zdrojů** – Kvůli v tuto chvíli malému tržnímu podílu má Vue stále dlouhou cestu. Relativně nízký věk tohoto frameworku přispěl k trápení ohledně běžných pluginů, které dělají prací s externími nástroji složitější,
- **rychlá evoluce** – Zatímco svižná vývojová fáze je dobrá pro framework z hlediska schopností, křivka učení se začíná měnit. Výsledkem je, že online výukové materiály

jsou rychle zastaralé a oficiální dokumentace je někdy pouze jedinou pomocí při problémech.

2.2.4 React.js

React je další populární JavaScriptová knihovna, která pohání Facebook. React je vyvíjen pod open source licencí samotnou firmou Facebook od roku 2013. Velice rychle vzrostla jeho prominentnost pro vývoj obrovských webových aplikací, které zahrnují dynamické zpracování dat.

V lednu 2019, Facebook rozšířil svou podporu pro tuto knihovnu tím, že přesunul CLI (Command Line Interface) nástroj Create-React-App z inkubace na oficiální Facebook repositář. (Copes, 2019)

Jedním z hlavních případů, kde React nabízí velkou využitelnost je, když vývojář potřebuje rozdělit komplexní zdrojový kód a znovupoužít jednotlivé komponenty v bezchybném prostředí, které má kapacity pro práci s real-time daty. Elementy spojené s tzv. lifecycle hooks, jako například anotace, pomáhají rozšířit uživatelský zážitek z aplikace. (Copes, 2019)

React používá virtuální DOM (Document Object Model), který pomáhá s integrací do jakékoli aplikace. Také adoptuje JSX, což je speciální syntaxe pro vývoj React aplikací. Výhodou této syntaxe je, že si rozumí s vyhledávači a celkově s SEO v porovnání s ostatními JavaScriptovými frameworky nebo knihovnami. V odstavcích níže se podíváme na výhody a nevýhody knihovny React.js. (Copes, 2019)

Výhody knihovny React.js:

- **relativně snadný na naučení** – React ukazuje, jak jednoduchá syntaxe může být přínosem. Velká část kódu je totiž obyčejné HTML. Jednoduchost naučení u dané technologie je sice velmi subjektivní pojem, avšak oproti Angularu, kde je potřeba se naučit programovací jazyk TypeScript, je React určitě přívětivější pro začátečníky. Velká závislost Reactu na běžném HTML dělá React jedním s nejpoužitelnějších nástrojů pro méně zkušené vývojáře,
- **knihovna, ne Framework** – React je JavaScriptová knihovna a ne framework. To znamená, že poskytuje deklarativní metodu při definování UI komponent. Také může být jednoduše integrován s ostatními knihovnami. Jako příklad uvádím knihovnu React-Redux, která poskytuje nutný spojovací článek mezi knihovnami React a Redux,

- **data a prezentace** – React poskytuje celkovou separaci datové a prezentační vrstvy,
- **vazba s DOM** – Vývojáři s Reactem nemusí provazovat DOM elementy s funkcemi a metodami. React toto řeší sám na základě pár řádků kódu, které se dají využít na více místech,
- **znovu-použitelnost komponent** – Facebook vyvinul React způsobem, který poskytuje programátorům schopnost opakovaně použít komponenty aplikace kdekoliv a kdykoliv. Tato schopnost ušetří hodně času při vývoji. Další výhodou tohoto konceptu je, že pokud chceme změnit nebo vylepšit jeden komponent, tak není třeba to dělat na více místech.

Nevýhody knihovny React.js:

- **vysoké vývojové tempo** – prostředí Reactu je velmi dynamické a často se mění. To pro vývojáře znamená nutnost učit se novým věcem. Pro ty, kteří se s daným nástrojem teprve učí, to může být velmi frustrující,
- **JSX jako bariéra** – React hodně používá JSX, které jsme už zmiňovali výše. Jedná se v podstatě o syntaxi, ve které se kombinuje HTML s JavaScriptem. Tato syntaxe, kromě svých výhod, má i nevýhody. Jednou z hlavních nevýhod je určitě vysoká úroveň komplexity a strmá „křivka učení“,
- **problémy se SEO** – Uživatelé oznamovali různorodé problémy Reactu a SEO (Search Engine Optimalizaton). Ačkoli se většinou jednalo o pouhé spekulace, doporučuje se brát SEO při vývoji v potaz.

2.2.5 Angular

Angular je považován za „nejsilnější“ open source framework. Prezentuje se jako řešení „vše v jednom“, které si klade za cíl poskytnout vývojářům všechny nástroje rovnou po nainstalování. Od routování až po práci s HTTP požadavky. Angular adoptuje TypeScript, jakožto jazyk pro vývoj aplikací.

Pravdou je že „křivka učení“ tohoto frameworku byla v prvních verzích velmi strmá, s příchodem verzí 5 a 6 je situace podstatně jednodušší. Těchto zjednodušení bylo dosaženo pomocí robustního CLI, který odstranil potřebu znát a pamatovat si nízko-úrovňové detaily pro psaní základních aplikací. (Freeman, 2018)

Vyvíjen a spravován firmou Google, Angular následoval harmonogram, který určoval že každých šest měsíců vyjde nová verze. Poslední verze se hlavně soustředily na

přidání nových funkcionalit a zlepšení výkonu. V následujících verzích frameworku, bychom se měli dočkat použití nástroje Ivy Renderer, jakožto výchozího vykreslovacího systému. Nyní se podíváme na výhody a nevýhody při vývoji s frameworkem Angular. (Freeman, 2018)

Výhody frameworku Angular:

- **dokumentace** – Angular přichází s velmi detailní dokumentací. Kvalitní dokumentace je velmi dobrá věc pro nové programátory při volbě frameworku a také jim pomůže pochopit zdrojový kód ostatních vývojářů,
- **jednodušší PWA** – PWA neboli Progressive Web Applications se staly velmi populární a Angular na tuto popularitu pružně zareagoval. Integroval nástroje pro vývoj SPA přímo do frameworku. S Angularem je taky možno využívat nativní funkcionality mobilních aplikací,
- **optimalizovaný Build systém** – Build systém Angularu odstraňuje všechen nepotřebný runtime kód. To aplikaci dělá lehčí a rychlejší díky tomu, že množství JavaScriptu v aplikaci se zmenší,
- **Universal State Transfer API a DOM** – Angular ve verzi 5 představil Universal State Transfer API a DOM. Tato technologie pomáhá sdílet kód mezi serverem a klientovou stranou aplikace,
- **Data Binding a MVVM** – Angular umožňuje tzv. Two-Way Data Binding. Tato funkcionalita minimalizuje rizika potencionálních chybových hlášek a zajišťuje jednotné „chování aplikace“. Na druhé straně MVVM neboli Model View View Model umožňuje vývojářům pracovat odděleně na stejné aplikaci,
- **CLI** – Pravděpodobně nejoblíbenější funkcionalita pro majoritu vývojářů pracujících s tímto frameworkem. Tento nástroj automatizuje většinu procesů, se kterými se při vývoji setkáme (inicializace aplikace, konfigurace, kompilace, build). Angular CLI nám umožňuje vytvořit nový projekt, konfigurovat jeho nastavení anebo například spouštět unit a end-to-end testy pomocí několika jednoduchých příkazů,
- **Ahead-Of-Time kompilace** – Angular disponuje speciálním AOT kompilátorem, který konvertuje veškerý TypeScript a HTML kód do čistého JavaScriptu. To znamená že kód je zkompilován ještě před tím, než ho webový prohlížeč načte. AOT kompilátor je také daleko bezpečnější, nežli JIT (Just-In-Time) kompilátor,
- **dependency Injection** – Dependency Injection je technika, kdy jeden objekt poskytuje určité závislosti jinému objektu. Dependency Injection tedy umožňuje

vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou.

Nevýhody frameworku Angular:

- **komplexní syntaxe** – Stejně jako v prvních verzích Angularu, tak i v jeho posledních verzích, zahrnuje spoustu komplikovaných syntaktických principů,
- **migrace** – Migrování aplikace ze starší verze Angularu na novější může být velmi složitá, hlavně v rozsáhlých aplikacích.

2.3 Výběr frameworku pro vývoj SPA frontendu

„Angular není řešením každého problému a je důležité vědět, kdy byste měli použít Angular a kdy byste měli hledat alternativu.“ Freeman (2018, s.32) (vlastní překlad).

V předchozích kapitolách jsme si představili hlavní trojici frameworků, které se používají k vývoji SPA. V této kapitole si v krátkosti popíšeme, proč pro tento projekt byl zvolen právě Angular.

Angular jako svůj programovací jazyk využívá nástavbu programovacího jazyka JavaScript od společnosti Microsoft – TypeScript, neboť TypeScript poskytuje všechny potřebné nástroje a funkce pro vývoj rozsáhlých a komplexních Enterprise webových aplikací. TypeScript je rovněž vybaven našeptávači, kteří automaticky doplňují kód, a pokročilým refaktoringem. TypeScript v sobě obsahuje veškeré atributy, které známe z klasického objektově orientovaného programování (třídy, moduly, rozhraní, generiky…). Před spuštěním se kód napsán v TypeScriptu kompiluje do JavaScriptu. Protože je TypeScript nástavbou nad JavaScriptem, je každý JavaScriptový kód automaticky validním TypeScript kódem. Skutečnost, že při vývoji s frameworkem Angular je využíván programovací jazyk TypeScript, je velkou výhodou oproti ostatním frameworkům. Díky této skutečnosti se stane výsledný programový kód lépe čitelným a udržovatelným. (Freeman, 2018)

Angular využívá architekturu postavenou na komponentech. Každý komponent výsledné aplikace má striktní specifikaci a chování nezávislé na okolním prostředí. Jestliže nějaký komponent je třeba odstranit, upravit, nebo již neodpovídá potřebám, je velmi jednoduché jej vyměnit. (Freeman, 2018)

Angular si bere inspiraci z těch nejlepších aspektů vývoje aplikací na straně serveru, a používá je, aby obohatil HTML v prohlížeči. Tímto způsobem tvoří základ,

který umožňuje vývoj komplexních aplikací jednodušší. Jak tvrdí Freeman (2018, s. 3) Aplikace vytvořené ve frameworku Angular jsou postaveny na návrhovém vzoru Model-View-Controller (MVC), který si zakládá na tvoření aplikací, které jsou:

- **rozšiřitelné** – jakmile chápeme základy, je jednoduché pochopit, jak i komplexní aplikace napsané v Angularu fungují, a to znamená že je můžeme jednoduše obohacovat o nové funkce a tím vytvořit nové vlastnosti aplikace pro koncového uživatele,
- **udržitelné** – Aplikace napsané ve frameworku Angular jsou jednoduché na debugování a opravování případných chyb, což znamená znatelné zjednodušení dlouhodobé udržitelnosti aplikace,
- **testovatelné** – Angular má velmi dobrou podporu pro Unit a End-to-End testování,
- **standardizované** – Angular staví na vrozených možnostech webového prohlížeče, aniž by nám bránil v práci a umožňuje vytvářet aplikace vyhovující standardům, které využívají výhod nejnovějších funkcí jako například HTML5 API a dalších oblíbených nástrojů a frameworků.

Angular je open source Framework postaven na JavaScriptu a je sponzorován a spravován společnostní Google a byl použit při vývoji některých z největších a nejsložitějších webových aplikací. (Freeman, 2018)

Tento framework byl zvolen také díky tomu, že někteří zaměstnanci v nově vzniklém IT oddělení již měli profesní zkušenost s tímto nástrojem. Společnosti se také velmi zamlouvalo, že Angular je používán hlavně pro vývoj Enterprise webových aplikací a stojí za ním světoznámá společnost, která je v mnoha očích ikonou na poli počítačových technologií. Angular tedy ve všech ohledech splňoval technické požadavky projektu.

2.4 Teoretická příprava pro vývoj v Angularu

Angular je platformou a frameworkem pro vývoj Single Page aplikací za použití značkovacího jazyka HTML a programovacího jazyka TypeScript. SPA jsou aplikace, ke kterým se přistupuje z webového prohlížeče jako k ostatním webovým stránkám nebo aplikacím, avšak SPA poskytují více dynamické interakce připomínající spíše mobilní nebo desktopové aplikace. Největší rozdíl mezi klasickou webovou stránkou a SPA je redukovaný počet obnovení stránky. (Freeman, 2018)

Zpravidla devadesát-pět procent SPA kódu běží v prohlížeči a zbytek pracuje na serveru, když uživatel potřebuje nová data nebo potřebuje provést zabezpečené operace jako například autentizaci. Výsledkem je, že proces vykreslování se děje převážně na klientově straně. (Freeman, 2018)

Architektura aplikací napsaných v Angularu se opírá o určité základní pojmy a principy. Základními stavebními bloky jsou moduly (NgModules), které poskytují kompilační kontext pro komponenty. Moduly shromažďují související kód do funkčních celků a výsledná aplikace je tedy definována sadou těchto modulů. Každá aplikace má vždy alespoň jeden kořenový modul, který umožní zavedení aplikace (tzv. Bootstraping), a obvykle má mnohem více funkčních modulů. (Freeman, 2018)

Komponenty definují pohledy, což jsou sady grafických prvků na obrazovce, které si Angular může vybrat a upravit podle své programové logiky a dat. (Freeman, 2018)

Komponenty používají služby (Services), které poskytují specifickou funkčnost nepřímo nesouvisející s pohledy. Poskytovatelé služeb (Service Providers) mohou být provázány s komponenty skrze Dependency Injection, díky čemuž bude výsledný kód modulární, opakovaně použitelný a efektivní. (Freeman, 2018)

Komponenty i služby jsou jednoduše třídy s anotacemi, které označují jejich typ a poskytují metadata, jenž říkaji Angularu, jak je používat. Metadata komponentu přidruží jeho třídu k šabloně, která definuje pohled. Šablona kombinuje běžný HTML kód s Angular direktivami a značkami připojení, které umožňují Angularu modifikovat HTML před jeho vykreslením. Metadata pro třídu služby poskytují informace, které Angular potřebuje, aby ji poskytl komponentům prostřednictvím vkládání závislostí. (Freeman, 2018)

Komponenty aplikace obvykle definují mnoho pohledů, jenž jsou hierarchicky uspořádány. Angular poskytuje tzv. Router, který vývojářům pomůže definovat navigační cesty mezi pohledy. Router poskytuje sofistikované možnosti navigace ve webovém prohlížeči. (Freeman, 2018)

2.4.1 Moduly

NgModuly v Angularu se liší od JavaScriptových modulů (ES2015) tím, že doplňují jejich základní funkcionality. NgModul deklaruje kompilační kontext pro sadu komponent, která je věnována aplikační doméně, pracovnímu postupu nebo úzce

související sadě funkcí. NgModul může přidružit své komponenty k souvisejícímu kódu, jakým jsou například služby za účelem vytvoření funkčních jednotek. (Freeman, 2018)

Moduly nám pomáhají aplikaci organizovat do soudržných funkcionálních bloků tím, že zastřeší všechny komponenty, pipes, direktivy a služby. (Freeman, 2018)

Aplikace, napsané ve frameworku Angular, jsou modulární. Každá aplikace má alespoň jeden modul – tzv. kořenový modul, konvenčně pojmenován AppModule. Většina aplikací je složena z více modulů, pouze malé aplikace obsahují jenom kořenový modul. Je na vývojáři a jeho rozhodnutí, jak bude moduly používat. Typicky však mapujeme hlavní funkcionalitu nebo prvek aplikace na modul. Představme si nyní, že naše aplikace obsahuje čtyři hlavní části. Každá z nich bude mít svůj modul a k nim se přidá modul kořenový, tedy celkově pět modulů. (Freeman, 2018)

Organizování kódu do různých funkčních modulů pomáhá při řízení vývoje komplexních aplikací a při navrhování kódu pro opětovné použití. Tato technika navíc umožňuje využít Lazy-loadingu – tj. načítání modulů na vyžádání, abychom minimalizovali množství kódu, který je třeba načíst při spuštění. (Freeman, 2018)

Stejně jako moduly v JavaScriptu mohou NgModuly importovat funkce z jiných NgModulů a umožňují exportovat a používat jejich vlastní funkce jinými NgModuly. Chceme-li například v aplikaci použít službu pro routování, importujeme RouterModule.

Každý modul ve frameworku Angular je třída s anotací `@NgModule`. Anotace jsou funkce, které různými způsoby modifikují JavaScriptové třídy. Používají se, abychom na třídy aplikovaly metadata. Tato metadata specifikují konfiguraci těchto tříd a způsob jejich funkce. V dokumentaci Angularu (www.angular.io) jsou parametry anotace `@NgModule` uvedeny takto:

- **declarations (deklarace)** – komponenty, direktivy a pipes, které patří do tohoto modulu,
- **exports (exporty)** – podmnožina deklarací, která by měla být viditelná a použitelná v šablonách komponentů z jiných NgModulů,
- **imports (importy)** – další moduly, jejichž exportované třídy jsou vyžadovány šablonami komponent deklarovanými v tomto NgModulu,
- **providers (poskytovatelé)** – poskytovatelé služeb, které tento NgModul zviditelňuje v globální sbírce služeb. Tyto služby se stanou přístupnými ve všech částech aplikace. Lze také určit poskytovatele na úrovni komponentu, což je častější varianta,

- **bootstrap** – hlavní pohled aplikace, kterému se říká root a hostuje všechny ostatní pohledy aplikace. Parametrem bootstrap by měl disponovat pouze kořenový NgModul,
- **entryComponents (vstupní komponenty)** – zde jsou definovány komponenty, které jsou načteny svou třídou a ne selektorem.

NgModuly poskytují kompilační kontext pro své komponenty. Kořenový NgModul má vždy kořenovou komponentu, která je vytvořena během bootstrapu, ale jakýkoli NgModul může obsahovat libovolný počet dalších komponent, které lze načíst přes router nebo vytvořit pomocí šablony. Komponenty, které patří do NgModulu, sdílejí kontext aplikace. (www.angular.io)

Ukázka kódu 2.1: Základní NgModul (zdroj: www.angular.io)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';


@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

2.4.2 Komponenty

Komponenty jsou základní stavební komponenty uživatelského rozhraní každé aplikace napsané ve frameworku Angular. Komponent řídí jeden nebo více sekcí na obrazovce (těmto sekcím říkáme pohledy). (www.angular.io)

Komponent je soběstačný a reprezentuje znovupoužitelnou část uživatelského rozhraní. Jak je uvedeno v dokumentaci Angularu (www.angular.io) komponent je většinou složen ze tří důležitých částí:

- část HTML kódu, kterému se říká pohled,
- třídy, která zapouzdřuje všechna dostupná data a interakce s pohledem skrze API rozhraní datových položek a metod. V této třídě definujeme aplikační logiku,
- a výše zmíněného HTML elementu známého jako selektor.

Použitím anotace `@Component` poskytujeme třídě další metadata, která určují, jak má být komponent zpracován a používán za běhu aplikace. Metadata komponentu dále říkají Angularu, kde získat hlavní stavební bloky, které potřebuje k vytvoření prezentace komponentu a jeho pohledu. Metadata zejména spojují šablonu s komponentem, buď přímo s vloženým kódem nebo odkazem. Komponent s jeho šablonou společně definují pohled. (www.angular.io)

Kromě toho, že metadata komponentu obsahují nebo odkazují na šablonu, tak ještě například konfigurují, jak se lze na komponent odkazovat v rámci HTML a jaké služby vyžaduje. (www.angular.io)

Komponent poskytuje data pohledu pomocí procesu zvaného Data Binding. Znamená to, že elementy DOMu se sváží s datovými položkami komponentu. Data Binding se používá k zobrazení hodnot uživateli, změně stylizace elementů, k reakci na událost vyvolanou uživatelem atd. (www.angular.io)

Komponent musí patřit do některého z modulů. Abychom specifikovali, že komponent je členem modulu, musíme jej zmínit v datové položce `declarations` uvnitř modulu.

Ukázka kódu 2.2: Základní komponent (zdroj: www.angular.io)

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
/* . . . */
}
```

V ukázce kódu 2.2 výše vidíme některé z nejužitečnějších možností konfigurace anotace `@Component`:

- **Selector** – Selektor CSS, který říká Angularu, aby vytvořil a vložil instanci tohoto komponentu, kdekoli najde odpovídající značku v šabloně HTML. Pokud například HTML aplikace obsahuje `<app-hero-list></app-hero-list>`, vloží Angular mezi tyto značky instanci zobrazení `HeroListComponent`,

- **templateUrl** – Adresa HTML šablony tohoto komponentu, která je umístěná relativně k modulu. Alternativně můžete zadat HTML šablonu jako inline hodnotu parametru `template`. Tato šablona definuje pohled hostitelského komponentu,
- **providers** – Pole poskytovatelů služeb, které komponent vyžaduje. V tomto konkrétním příkladu to znamená, že Angular musí tomuto komponentu poskytnout instanci třídy `HeroService`, kterou konstruktor komponentu využívá k získání seznamu hrdinů k zobrazení.

Pohledy jsou obvykle uspořádány hierarchicky, což nám umožňuje upravit nebo zobrazit a skrýt celé sekce nebo stránky uživatelského rozhraní jako celek. Šablona bezprostředně spojená s komponentem definuje pohled hostitele tohoto komponentu. Komponent může také definovat hierarchii pohledů, která obsahuje vložená zobrazení, hostovaná jinými komponenty. Hierarchie pohledů může zahrnovat pohledy z komponentů ve stejném NgModulu, ale může také zahrnovat pohledy z komponentů, které jsou definovány v různých NgModulech. (Freeman, 2018)

Šablony vypadají jako běžné HTML, kromě toho, že obsahují také syntaxi specifickou pro Angular, která mění HTML na základě logiky aplikace, stavu aplikace a dat z DOMu. Šablona může použít vázání dat ke koordinaci aplikace a dat v DOMu, pipes se používají k transformaci dat před jejich zobrazením a direktivy pro aplikovaní logiky aplikace na to, co se má zobrazit. (Freeman, 2018)

Ukázka kódu 2.3: Příklad šablony komponentu (zdroj: www.angular.io)

```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>

<ul>
  <li *ngFor="let hero of heroes"
    (click)="selectHero(hero">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail
  *ngIf="selectedHero"
  [hero]="selectedHero">
</app-hero-detail>
```

Tato šablona používá typické prvky HTML, jako jsou `<h2>` a `<p>` a také obsahuje prvky Angular syntaxe pro šablony - `*ngFor`, `{{hero.name}}`,

`(click)`, `[hero]` a `<app-hero-detail>`. Prvky této syntaxe informují Angular o tom, jak vykreslit HTML na obrazovku pomocí logiky programu a dat:

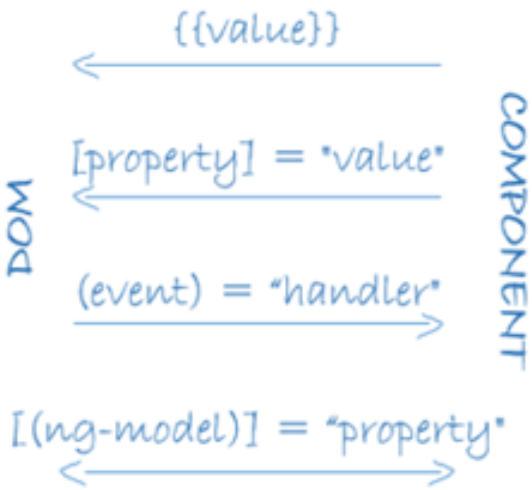
- direktiva `*ngFor` říká Angularu aby iteroval nad seznamem,
- `{{hero.name}}`, `(click)`, `[hero]` naváže data programu do a z DOMu v reakci na vstup uživatele,
- značka `<app-hero-detail>` v příkladu je prvek představující vnořený komponent `HeroDetailComponent`.

Vazba dat

Bez frameworku bychom byli odpovědní za vložení dat do HTML, ovládacích prvků, ale i za přeměnu odpovědí uživatelů na akce a aktualizace hodnot. Psaní takovéto „push and pull“ logiky ručně je nejenom únavné, ale i značně nečitelné a náchylné k chybám, což by jistě potvrdil každý zkušený frontend programátor. (Freeman, 2018)

Angular podporuje obousměrné vázání dat neboli mechanismus pro koordinaci částí šablony s částmi kódu komponentu. Přidáním specifických značek můžeme Angularu říct, jak obě strany připojit. (www.angular.io)

Obrázek 2.1 ukazuje čtyři typy syntaxe pro vazbu dat. Každý formulář má směr: do DOMu, z DOMu nebo do obou.



Obrázek 2.1: Formy Data Bindingu (zdroj: www.angular.io)

Na ukázce kódu šablony komponentu níže si můžeme povšimnout použití vazby dat.

Ukázka kódu 2.4: Příklad šablony komponentu (zdroj: interní zdroj)

```
<li>{{hero.name}}</li>
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<li (click)="selectHero(hero)"></li>
```

V ukázce kódu se vyskytují tyto syntaktické prvky:

- syntaxi `{{hero.name}}` se říká Interpolace a zobrazuje hodnotu datové položky komponentu hero.name v HTML elementu `<li>`,
- syntax `[hero]` předává hodnotu datové položky komponentu `selectedHero` do parametru `hero` podřízeného komponentu se selektorem `<app-hero-detail>`,
- vazba události `(click)` volá metodu komponentu `selectHero`, když uživatel klikne na jméno hrdiny.

Obousměrná vazba dat neboli Two-Way Binding (používá se hlavně ve formulářích řízených šablonou) kombinuje vazbu datových položek a událostí do jediného zápisu. Na ukázce kódu 2.5 níže vidíme část kódu šablony, která používá obousměrnou vazbu s direktivou `ngModel`.

Ukázka kódu 2.5: Příklad šablony komponentu (zdroj: www.angular.io)

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

U obousměrné vazby datová položka komponentu „teče“ do vstupního pole jako u vazby datové položky. Změny uživatele však také plynou zpět do komponentu a resetují datovou položku na nejnovější hodnotu, jako je tomu u vazby událostí. (www.angular.io)

Vazba dat hraje důležitou roli v komunikaci mezi šablonou a jejím komponentem a je také velmi důležitá pro komunikaci mezi nadřazenými a podřazenými komponenty.

Pipes

Angular pipes jsou speciální operátory, které umožňují deklarovat transformace hodnoty zobrazené v HTML šabloně. Třída s anotací `@Pipe` definuje funkci, která transformuje vstupní hodnoty na výstupní pro výsledné zobrazení v pohledu. (Freeman, 2018)

Angular má v sobě již těchto operátorů zabudovaných několik, jako například `date` nebo `currency`. Programátor si může samozřejmě definovat své pipes.

Chceme-li aplikovat tyto operátory k transformaci hodnoty v šabloně HTML, je nutno použít pipe symbol (|).

Ukázka kódu 2.6: Příklad použití pipe operátoru (zdroj: interní zdroj)

```
{{interpolated_value | pipe_name}}
```

Operátory lze řetězit a poslat výstup jedné funkce, aby byl transformován funkcí jinou. Pipes mohou přijímat i argumenty, které říkají, jak transformaci provést.

Ukázka kódu 2.7: Příklad použití pipe operátoru `date` (zdroj: interní zdroj)

```
<!-- Default format: output 'Jun 15, 2015'-->
<p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->
<p>The date is {{today | date:'fullDate'}}</p>

<!-- shortTime format: output '9:43 AM'-->
<p>The time is {{today | date:'shortTime'}}</p>
```

Direktivy

Šablony v Angularu jsou dynamické. Když je Angular vykreslí, transformuje DOM podle pokynů daných direktivami. Direktiva je třída s anotací `@Directive()`. (Freeman, 2018)

Komponent je technicky vzato direktivou. Komponenty jsou však natolik charakteristické a klíčové, že Angular definuje anotaci `@Component()`, která rozšiřuje anotaci `@Directive()` o funkce orientované na šablony. (Freeman, 2018)

Kromě komponentu existují dva další druhy direktiv: strukturální a atributové. Angular v základu obsahuje celou řadu direktiv obou druhů a vývojář si může definovat své vlastní direktivy pomocí anotace `@Directive()`. (Freeman, 2018)

Stejně jako u komponentů, metadata direktivy přidruží anotovanou třídu k prvku selektoru, který použijeme pro vložení do HTML. V šablonách se direktivy obvykle objevují v tagu elementu jako atributy, buď podle názvu, přiřazení nebo vazby. (Freeman, 2018)

Strukturální direktivy mění rozvržení šablony přidáním, odebráním a nahrazením prvků v DOM. Na ukázce kódu 2.8 níže vidíme použití dvou strukturálních direktiv:

Ukázka kódu 2.8: Příklad použití dvou strukturálních direktiv (zdroj: interní zdroj)

```
Příklad šablony komponentu

<li *ngFor="let hero of heroes"></li>
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

- `*ngFor` je iterativní direktiva, která na ukázce kódu výše vykresluje jeden element `<li>` pro každého hrdinu v seznamu,
- `*ngIf` je podmínka, která na ukázce kódu výše vykresluje komponent `HeroDetail`, pouze pokud vybraný hrdina existuje.

Atributové direktivy mění vzhled nebo chování existujícího prvku. V šablonách vypadají jako normální HTML atributy, odtud název atributové. (www.angular.io)

Direktiva `ngModel`, která implementuje obousměrnou vazbu dat, je příkladem atributové direktivy. `ngModel` upravuje chování existujícího prvku (zpravidla tagu `<input>`) nastavením jeho hodnoty a reagováním na změny. Příklad použití atributové direktivy `ngModel` vidíme na následující ukázce kódu 2.9. (www.angular.io)

Ukázka kódu 2.9: Příklad použití atributové direktivy (zdroj: interní zdroj)

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

Komponenty jsou velmi důležité stavební bloky z hlediska softwarové architektury. Jejich hlavní princip je osamostatnit jednotlivé části uživatelského rozhraní. Představme si, že máme dva UI bloky v jednom komponentu, a dokonce i ve stejném souboru. Na začátku vývojového cyklu aplikace sice tyto komponenty budou malé, ale s přibývajícími požadavky budou růst. Každá změna v komponentu ovlivní obě její části a také se zdvojnásobí nutné testy. Pokud bychom chtěli jeden z UI bloků znovupoužít někde jinde, nemůžeme tak učinit, aniž bychom přenesli i druhý blok. Tento scénář porušuje Single Responsibility princip. Angular tento princip používá tím, že nechává každou část stránky řídit jejím vlastním komponentem. Typická aplikace vyvinutá ve frameworku Angular vypadá jako strom komponentů. (Freeman, 2018)

2.4.3 Služby

Service neboli služba může být hodnota, funkce, datová položka nebo jiný zdroj, kterou naše aplikace potřebuje. Služba je obvykle třída, která slouží ke sdílení zdrojů mezi komponenty. (www.angular.io)

Komponenty by měli sloužit jen k přemostění komunikace mezi pohledem, uživatelem a aplikační logikou. Komponenty zpravidla nezískávají data ze serveru, nevalidují uživatelova vstupní data. Takové úlohy, a jiné netriviální procesy, delegují službám. (www.angular.io)

Angular rozlišuje komponenty od služeb, aby se zvýšila modularita a opětovné použití. V ideálním případě je úkolem komponentu poskytnout uživatelský komfort a nic víc. Komponent by měl prezentovat datové položky, a metody pro vazbu dat, aby zprostředkoval pohled (vykreslený šablonou). Komponent může delegovat určité úkoly na služby, jako je načítání dat ze serveru, validování vstupů uživatelů nebo logování na konzolu. Definováním takových úkolů ve speciální třídě, tyto úkoly zpřístupníme jakékoli součásti aplikace. Angular tyto zásady nevynucuje, ale pomůže nám dodržovat tyto zásady tím, že usnadňuje začlenění logiky naší aplikace do služeb a zpřístupňuje tyto služby komponentům, prostřednictvím vkládání závislostí. Těmto zásadám odstínění složitých funkcí a úloh od komponentů se říká Separation of Concerns. (Freeman, 2018)

Služby mohou záviset na jiných službách. Na ukázce kódu 2.10 níže vidíme službu `HeroService`, která závisí na službě `Logger`, a také používá `BackendService` k získání hrdinů. Tato služba může zase záviset na službě `HttpClient`, aby mohla získat hrdiny asynchronně ze serveru. (www.angular.io)

Ukázka kódu 2.10: Příklad služby (zdroj: www.angular.io)

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

Služby jsou fundamentálním základem pro každou Angular aplikaci, neboť veškeré jeho komponenty jsou konzumenty těchto služeb. (www.angular.io)

Vkládání závislostí

Vkládání závislostí, neboli Dependency Injection (zkráceně DI), je součástí frameworku Angular a používá se všude tam, kde framework poskytuje nové komponenty se službami. Komponenty jsou konzumenti služeb, to znamená, že můžeme do komponentu vložit službu, která tomuto komponentu poskytne přístup k jejím metodám a vlastnostem. (Freeman, 2018)

Chceme-li definovat třídu jako službu, musíme použít anotaci `@Injectable()`, která třídě poskytne metadata, jenž umožní třídě vstoupit do komponentu jako závislost. (Freeman, 2018)

Hlavním mechanismem je injector. Angular během bootstrapingu pro nás vytvoří injector pro celou aplikaci. Injector vytváří a udržuje kontejner instancí závislostí, který pokud možno používá opakovaně. Poskytovatel (Provider) je objekt, který říká injektoru, jak určitou závislost získat nebo vytvořit. Pro jakoukoli závislost, kterou potřebujeme v aplikaci, musíme zaregistrovat poskytovatele, aby injector mohl pomocí poskytovatele vytvářet nové instance. U služby je poskytovatel obvykle samotná třída služby. (Freeman, 2018)

Když Angular vytvoří novou instanci třídy komponentu, určí, které služby nebo jiné závislosti, jež komponent potřebuje tím, že se podívá na parametry konstruktoru. Na ukázce kódu 2.11 uvedené níže vidíme konstruktor komponentu `HeroListComponent`, který potřebuje `HeroService`. (www.angular.io)

Ukázka kódu 2.11: Konstruktor komponentu (zdroj: www.angular.io)

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

Když Angular zjistí, že komponent závisí na službě, nejprve zkontroluje, zda injector má nějaké existující instance této služby. Pokud požadovaná instance služby dosud neexistuje, injector jednu vytvoří pomocí registrovaného poskytovatele a předá jí komponentu. (www.angular.io)

Jakmile byly všechny požadované služby vyřešeny a předány, může Angular zavolat konstruktor komponentu s těmito službami jako argumenty. (www.angular.io)

2.4.4 Observables a RxJS

Observables jsou funkce nebo generické datové typy, které mohou vracet proud hodnot odběrateli v čase. Observables se používají pro usnadnění práce při vývoji aplikací, kde je použito asynchronní nebo reaktivní paradigma programování.

Observables poskytují podporu pro předávání zpráv mezi poskytovateli (publishers) a odběrateli (subscribers) v naší aplikaci. Observables nabízejí významné výhody oproti jiným technikám zpracování událostí, asynchronní programování a zpracování více hodnot. (www.angular.io)

Observables jsou deklarativní – to znamená, že definujeme funkci pro publikování hodnot, ale není vykonávána, dokud se konzument nepřihlásí k odběru. Přihlášený odběratel poté obdrží oznámení, dokud funkce neukončí svou činnost, nebo pokud odběratel svůj odběr neodhlásí. (www.angular.io)

Observable může v závislosti na kontextu doručit více hodnot jakéhokoli typu – literály, zprávy nebo události. API rozhraní pro příjem hodnot je stejné, ať už jsou hodnoty doručovány synchronně nebo asynchronně. Protože logiku nastavení řeší Observable, náš aplikační kód se musí starat pouze o odebírání hodnot a po odhlášení po dokončení. Ať už jde o stisky kláves, HTTP odpověď nebo intervalový časovač, rozhraní pro odběr hodnot a zastavení odběru je stejné. (www.angular.io)

Kvůli těmto výhodám jsou Observables hojně využívány v Angularu a jsou doporučovány i pro vývoj aplikací. (www.angular.io)

Základní pojmy a použití

Jako poskytovatel vytvoříme instanci třídy Observable, která definuje určitou funkci. Tato funkce se provádí, když odběratel volá metodu `subscribe()`. Funkce definuje, jak získat nebo vygenerovat hodnoty nebo zprávy, které mají být zveřejněny. (www.angular.io)

Chceme-li „spustit“ Observable, který jsme vytvořili a začít přijímat oznámení, zavoláme metodu `subscribe()` a předáme pozorovatele. Tím je JavaScriptový objekt, který definuje obslužné rutiny pro obdržená oznámení. Volání metody `subscribe()` vrací objekt `Subscription`, který má metodu `unsubscribe()`, kterou voláme, abychom zastavili přijímání oznámení. Na ukázce kódu 2.12 níže vidíme příklad, který

demonstruje základní model použití tím, že ukazuje, jak lze Observable použít k poskytovaní geolokačních aktualizací. (www.angular.io)

Ukázka kódu 2.12: Příklad použití Observable (zdroj: interní zdroj)

```
// Vytvoření Observable, která začne čekat na změny
// geolokace, jakmile začne uživatel odebírat.
const locations = new Observable((observer) => {
  let watchId: number;

// Jednoduché API získání polohy
  if ('geolocation' in navigator) {
    watchId =
navigator.geolocation.watchPosition((position: Position) =>
{
      observer.next(position);
    }, (error: PositionError) => {
      observer.error(error);
    });
  } else {
    observer.error('Geolocation not available');
  }

// Jakmile konzument zavolá unsubscribe(), vyčistíme data
// pro další odběr.
  return {
    unsubscribe() {
      navigator.geolocation.clearWatch(watchId);
    }
  };
});

// Zavoláním subscribe() započneme čekání na aktualizace.
const locationsSubscription = locations.subscribe({
  next(position) {
    console.log('Current Position: ', position);
  },
  error(msg) {
    console.log('Error Getting Location: ', msg);
  }
});

// Po deseti sekundách zrušíme čekání na aktualizace.
setTimeout(() => {
  locationsSubscription.unsubscribe();
}, 10000);
```

2.5 Agilní metodiky vývoje SW

„Být agilní, znamená mít schopnost vytvářet a reagovat na změny, abychom uspěli v nejistém a turbulentním prostředí.“ (www.agilealliance.com) (vlastní překlad)

Agilní přístup k vývoji softwaru obsahuje řadu metod a metodik, které našemu týmu pomohou myslet více efektivněji, pracovat efektivně a dělat lepší rozhodnutí. Tyto metody a metodiky se týkají všech oblastí tradičního softwarového inženýrství, včetně projekt managementu, návrhu architektury softwaru a zefektivnění procesů. Každá z těchto metod a metodik se skládá z praktik, které jsou optimalizovány tak, aby je bylo možno co nejsnadněji adaptovat. (Myslín, 2016)

Agilní přistup je také způsob myšlení, protože správné myšlení může mít velký vliv na to, jak tým tyto praktiky efektivně využívá. Toto myšlení pomáhá lidem v týmu mezi sebou sdílet informace, a na základě těchto informací provádět důležitá projektová rozhodnutí společně. Agilní myšlení je o plánování, návrhu a zlepšení procesů pro celý tým. Agilní tým používá tyto praktiky tak, že každý sdílí stejné informace a každá osoba v týmu má podíl na tom, jak se tyto praktiky používají. (Myslín, 2016)

Agilní softwarový vývoj je tedy zastřešující termín, který se používá k popisu souboru metod a přístupů, které pomohou dodat zákazníkovi častou přidanou hodnotu. V jádru agilního přístupu samo-organizující se týmy využívají postupy, které odpovídají jejich kontextu, a vytvářejí řešení prostřednictvím spolupráce. (Myslín, 2016)

2.5.1 Stručná historie agilních metodologií

Na konci devadesátých let začala stoupat popularita různých metodik vývoje softwaru, z nichž každá měla svůj vlastní soubor nápadů. Každý z přístupů však měl několik společných rysů zdůrazňujících spolupráci mezi vývojovým týmem a obchodními partnery, časté dodávání přidané hodnoty, malé organizující se týmy a inovativní způsob vytváření, testování a nasazování zdrojového kódu. (Donald, 2017)

Pojem „Agilní“ byl použit pro tuto sbírku metodologií před dvaceti-jedna lety v roce 2001, kdy 17 odborníků na vývoj softwaru se sešlo v Utahu a společně diskutovalo a sdílelo různé přístupy k vývoji softwaru za účelem vytvoření Agilního manifestu. (Donald, 2017)

Program Agilní manifest vyústil v dohodnutou sadu čtyř hodnot a dvanácti principů mezi skupinou, které tvoří základ agilního vývoje softwaru a jsou dodnes běžně používány. (Donald, 2017)

2.5.2 Hodnoty agilního softwaru

Jednotlivci a interakce před procesy a nástroji

Vážit si lidí více než procesů nebo nástrojů, tento výrok lze snadno pochopit, protože to jsou lidé, kteří reagují na obchodní potřeby a řídí vývojový proces. Pokud proces nebo nástroj řídí vývoj, tým hůře reaguje na změny a s menší pravděpodobností uspokojí potřeby zákazníků. Komunikace je příkladem rozdílu mezi oceněním jednotlivců versus mezi oceněním procesů. V případě jednotlivců je komunikace plynulá a komunikuje se, pokud vznikne potřeba komunikovat. V případě procesu je komunikace naplánovaná a vyžaduje konkrétní obsah. (Donald, 2017)

Funkční software před obsáhlou dokumentací

Historicky bylo obrovské množství času věnováno dokumentování produktu pro vývoj a konečné dodání. Technické specifikace, technické požadavky, technický prospekt, návrhové dokumenty, testovací scénáře, plány dokumentace a schválení požadovaná pro každou z nich. Seznam byl rozsáhlý a byl příčinou dlouhých zpoždění ve vývoji. Agilní přístup nevylučuje dokumentaci, nýbrž ji zefektivňuje do formy, která dává vývojáři to, co je potřeba k provedení práce. Agilní manifest si váží dokumentace, ale více si váží fungujícího produktu. (Donald, 2017)

Spolupráce se zákazníky před smluvním vyjednáváním

Vyjednávání je období, kdy zákazník a produktový manažer vypracují podrobnosti o dodávce. S vývojovými modely, jako je model vodopád, zákazníci sjednávají požadavky na produkt velmi podrobně před zahájením jakékoli práce. To znamená, že zákazník je zapojen do procesu vývoje před zahájením a po jeho dokončení, nikoli však během procesu. Agilní manifest popisuje zákazníka, který je zapojen a spolupracuje na celém procesu vývoje. Díky tomuto faktu je daleko jednodušší uspokojit potřeby zákazníka. Agilní metody mohou zapojit zákazníka pomocí pravidelných ukázek.

Reakce na změny před dodržováním plánu

Při tradičním vývoji softwaru se považuje změna za náklad, takže by se jí mělo zabránit. Záměrem bylo vyvinout podrobné, propracované plány, s definovanou sadou funkcí, které mají stejně vysokou prioritu jako všechno ostatní. (Donald, 2017)

Zkrácení iterací znamená, že priority mohou být přesunuty z jedná iterace do druhé a nové funkce mohou být přidány až v následující iteraci. Agilní pohled na vývoj je ten, že změny mohou vždy produkt zlepšit a dodávají přidanou hodnotu. (Donald, 2017)

2.5.3 Principy agilního softwaru

V agilním manifestu je zahrnuto dvanáct principů. Popisují kulturu, ve které je změna vítána, a práce je zaměřena na zákazníka. Také demonstrují záměr popsaný Alistairem Cockburnem, jedním z lidí, kteří podepsali agilní manifest. Cockburnovým cílem je přiblížit vývoj potřebám podniku.

1. **Spokojenost zákazníků prostřednictvím včasného a nepřetržitého dodávání softwaru** – Zákazníci jsou spokojenější, když dostávají fungující software v pravidelných intervalech, spíš než aby čekali delší dobu na kompletní vydání.
2. **Přizpůsobení měnícím se požadavkům během procesu vývoje** – Schopnost zabránit zpoždění při změně požadavku nebo funkce.
3. **Časté dodávky fungujícího softwaru** – SCRUM vyhovuje tomuto principu, protože tým pracuje v softwarových sprintech nebo iteracích, které zajišťují pravidelné dodání pracovního softwaru.
4. **Spolupráce mezi podnikatelskými subjekty a vývojáři v průběhu projektu** – Jestliže se spojí obchodní a technický tým, bude se dosahovat lepších rozhodnutí.
5. **Podpora, důvěra a motivace zúčastněných lidí** – Motivované týmy s větší pravděpodobností dodají svou nejlepší práci než týmy nemotivované.
6. **Povolení interakce tváří v tvář** – Komunikace je úspěšnější, když jsou vývojové týmy umístěny u sebe.
7. **Fungující software je hlavním měřítkem pokroku** – poskytování fungujícího softwaru zákazníkovy je klíčovým faktorem úspěchu a pokroku.
8. **Agilní procesy podporující konzistentní tempo vývoje** – týmy stanovují opakovatelnou a udržovatelnou rychlost, jakou mohou dodávat fungující software. Tuto rychlost lze opakovat s každou verzí.

9. **Pozornost k technickým detailům a návrhu zvyšuje obratnost** – Správné dovednosti a dobrý design zajišťují, že tým dokáže lépe udržet tempo a neustále zlepšovat produkt.
10. **Jednoduchost** – Vyvíjet jen tolik, aby byla práce v okamžiku splněna.
11. **Samo-organizující se týmy podporují kvalitní architektury, požadavky a návrhy** – Zkušení a motivovaní členové týmu, kteří se umí dobře rozhodovat, pravidelně komunikují s ostatními členy týmu a sdílejí nápady, které jsou základem pro kvalitní produkty.
12. **Pravidelně uvažovat o tom, jak činnosti zefektivnit** – sebehodnocení, zlepšování procesů a moderní techniky pomáhají členům týmu pracovat efektivněji.

Záměrem agilního přístupu je sladit vývoj s obchodními potřebami. Agilní projekty se zaměřují na zákazníka a podporují jeho bezprostřední účast. Výsledkem je, že agilní přístup se stal zastřešujícím pojmem pro vývoj softwaru v celém odvětví softwarového vývoje. (Donald, 2017)

2.6 Agilní metodiky vývoje SW – Kanban

Kanban je vizuální systém pro řízení práce napříč procesy. Kanban vizualizuje proces i skutečnou práci, která tímto procesem prochází. Cílem Kanbanu je identifikovat potencionální úzká místa v našich procesech a snažit se je odstranit, tak aby mohla práce skrz ně procházet efektivně a optimální rychlostí. (Šochová, 2014)

2.6.1 Stručná historie Kanbanu

Na počátku 40.let minulého století vyvinul Taiichi Ohno (průmyslový inženýr a podnikatel) první systém Kanban pro společnost Toyota automotive v Japonsku. Byl vytvořen jako jednoduchý plánovací systém, jehož cílem bylo optimálně řídit práci a zásoby v každé fázi výroby. (www.digite.com)

Klíčovým důvodem pro vývoj Kanbanu byla nedostatečná produktivita a efektivita v závodech Toyota ve srovnání s konkurenčními americkými automobilkami. Díky Kanbanu dosáhla Toyota flexibilního a efektivního systému řízení výroby v reálném čase, který zvýšil produktivitu a současně snížil nákladově náročnou zásobu surovin, polotovarů a hotových výrobků. (www.digite.com)

2.6.2 Podstata metody Kanban

Zatímco Kanban byl představen Taiichim Ohnem ve výrobním průmyslu, byl to David J. Anderson, kdo jako první aplikoval tento koncept na IT, vývoj softwaru a znalostní práci obecně. Tato aplikace proběhla v roce 2004. Anderson stavěl na dílech Ohna, Goldratta, Dremmingse a dalších, aby definoval, co je to metoda Kanban. Jeho první kniha o Kanbanu – „Kanban: Successfully Evolutionary Change for your Technology Business“, vydána v roce 2010, je nejobsáhlejší definicí metody Kanban. (Šochová, 2014)

Metoda Kanban je proces pro postupné zlepšování všeho, co děláte – ať se jedná o vývoj softwaru, IT, personální obsazení, nábor lidských zdrojů, marketing a prodej, nákup atd. Ve skutečnosti může mít téměř každá obchodní funkce prospěch z uplatňování principů metodologie Kanban. (Šochová, 2014)

2.6.3 Zásady a postupu metody Kanban

Metoda Kanban dodržuje řadu zásad a postupů pro řízení a zlepšování pracovního postupu. Je to evoluční metoda, která podporuje postupné zlepšování procesů organizace. Pokud se dodržují tyto zásady a postupy, bude se moci úspěšně používat Kanban pro maximalizaci výhod pro náš obchodní proces – zlepšení toku, zkrácení doby cyklu, zvýšení hodnoty pro zákazníka s větší předvídatelností – to vše je dnes rozhodující pro jakékoli podnikání. (Šochová, 2014)

2.6.4 Koncept metody Kanban

Kanban je nepřerušující evoluční systém řízení změn. To znamená, že stávající proces je vylepšen v malých krocích. Implementací mnoha menších změn se snižuje riziko pro celý systém. Evoluční přístup Kanbanu vede k nízkému nebo žádnému odporu v týmu a zúčastněných zainteresovaných stranách. (Anderson, 2010)

Prvním krokem při zavedení Kanbanu je vizualizace pracovního postupu. To se děje ve formě jednoduché tabule a karet. Každá karta na tabuli představuje dílčí úkol. (Anderson, 2010)

V klasickém Kanban modelu jsou v tabulce tři sloupce:

- **ToDo** – Tento sloupec obsahuje seznam úkolů, které ještě nebyly zahájeny (někdy též zvaný Backlog),
- **In Progress** – Skládá se z probíhajících úkolů,

- **Done** – Skládá se z dokončených úkolů.

Tato jednoduchá vizualizace sama o sobě vede k velké transparentnosti v rozložení práce a existujícím problémovým místům. Kanban tabule mohou samozřejmě zobrazovat i komplikované pracovní postupy v závislosti na složitosti pracovního postupu a potřebě vizualizovat a prozkoumat konkrétní části pracovního postupu, aby se identifikovala úzká místa. (Anderson, 2010)

2.6.5 Koncept Toku

Jádrem Kanbanu je koncept tzv. Toku (Flow). To znamená, že karty by měly „protékat“ systémem rovnoměrně, bez dlouhých čekacích dob nebo blokování. Vše, co brání toku by mělo být kriticky prozkoumáno. Kanban má různé techniky, metriky a modely, a pokud jsou důsledně aplikovány, může to vést ke kultuře neustálého zlepšování. (Anderson, 2010)

Koncept Toku je velmi důležitý. Snahou o jeho zlepšení můžeme dramaticky zvýšit rychlost našich dodacích procesů a zároveň zkrátit dobu cyklu. Zlepšit kvalitu svých produktů nebo služeb získáním rychlejší zpětné vazby od svých zákazníků, ať už interních nebo externích. (Anderson, 2010)

2.6.6 Příklady využití

Velkou výhodou Kanbanu je jeho jednoduchost. Kanban však není jen o vizualizaci procesu na fyzické nebo elektronické tabuli a posouvání nalepovacích nebo digitálních lístků. Jak je mož vidět, jde o mnohem více. Z implementace Kanbanu se dá nejvíce těžit, pokud budeme metodicky uplatňovat všechny zásady a postupy. (Anderson, 2010)

Současné trendy z celého světa ukazují, že Kanban získává na popularitě a používá se v mnoha různých oblastech, od malých agentur a začínajících podniků až po tradiční organizace všech velikostí. (Anderson, 2010)

2.6.7 Kanban v IT

Kanban není metodologie vývoje softwaru ani projektového řízení. Kanban neříká nic o tom, jak by měl být software vyvíjen. Nehovoří ani o tom, jak by měly být softwarové projekty plánovány a implementovány. Kanban proto není frameworkem pro management, jako například Scrum. Účelem Kanbanu je místo toho neustále zlepšovat pracovní proces. (Šochová, 2014)

Kanban byl použit v operacích vývoje softwaru společnosti Microsoft v roce 2004. Od té doby byl Kanban nadšeně přijat v IT, aplikačních ale i DevOps softwarových týmech. (www.digite.com)

Přednost Kanbanu je v tom, že se dá použít na jakýkoli proces nebo metodiku. Ať už používáme agilní metody, jakou jsou Scrum, XP a další, nebo tradiční metody – vodopád, iterativní atd. – můžeme použít Kanban a postupně začít zlepšovat své procesy, zkrátit dobu cyklu a zlepšit svůj tok práce. (www.digite.com)

2.6.8 Kanban v produktovém vývoji a agilním softwaru

Týmy pro vývoj aplikačního softwaru a technologických produktů přijaly Kanban jako způsob implementace principů agility. Metoda Kanban poskytuje technologickým týmům velký soubor zásad pro vizualizaci jejich práce, poskytování produktů a služeb nepřetržitě a častějšímu získávání zpětné vazby od zákazníků. V důsledku toho pomáhá týmům rychleji vstoupit na trh s větší věrností tomu, co zákazníci od těchto produktů a služeb očekávají. (www.digite.com)

Definice Kanbanu v IT sektoru prošla v posledních třech až pěti letech vlastním vývojem. Dnes je Kanban považován za metodu, která přináší obratnost v řízení a zlepšování poskytování služeb postupným, evolučním způsobem. (www.digite.com)

Metoda Kanban navíc poskytuje důležité zásady a techniky pro lepší řízení závazků Service Level Agreement (SLA), dodávání produktů na trh Just-In-Time a minimalizaci rizika a nákladů z důvodu zpoždění. Kanban pomocí konceptů, jako je Class of Services, Deferred Commitment a Two-Phase commit, pomáhá zákazníkům a doručovacím týmům účinně spolupracovat a pomáhá zajistit, aby se na správných věcech pracovalo ve správný čas. (www.digite.com)

3 Analýza a popis výchozího stavu CRM systému ve firmě

„Klíčový aspekt CRM je vytvoření takového vztahu se zákazníkem, který bude silný a přinese podniku co nejvyšší hodnotu.“ Lehtinen (2007, s.36)

Nemůžeme zlepšit něco co nelze změřit. Stejně tak nemůžeme optimalizovat systém, který nemáme zmapován, a proto si v této kapitole představíme aktuální stav celého CRM systému. Podíváme se, jak je celá aplikace postavená a technologicky řešená z pohledu technologického stacku, datového modelu a databáze, funckionality a serverové architektury. Důležitým milníkem pro tuto práci je frontendová část aplikace, protože cílem této práce je optimalizace právě frontendu, nicméně pro úplnost provedeme analýzu celé aplikace.

3.1 Serverová architektura

Na počátku životního cyklu CRM systému se nedalo přímo hovořit o serveru ani o serverové architektuře, neboť aplikace běžela v prostředí webového hostingu, který zprostředkovávala známá česká firma Wedos. Parametry tohoto webhostingu byly dostačující, jako například neomezená velikost prostoru pro webovou aplikaci, dostatečně velký prostor pro databázi a e-maily, základní antispamová kontrola či podpora používání automatizovaných procesů CRON.

Ačkoli tyto parametry nebránily provozu aplikace ani růstu celé firmy, bylo postupem času rozhodnuto, že se místo webového hostingu použije cloudové řešení. Bylo zvoleno řešení Cloud VPS od firmy OVH zejména díky úrovni služeb, velikosti a reputaci firmy a příznivé cenové nabídce. Cloudový outsourcing serverových technologií byl zvolen z toho důvodu, neboť v čase rozhodování ještě nebyla potřeba firmy mít vlastní serverovou základnu a lidské zdroje, které by zabezpečovaly její správu a chod.

3.2 Databáze

Datový model obstarává klasická relační databáze implementována v prostředí systému řízení báze dat MySQL. Databáze je relativně rozsáhlá a obsahuje přes padesát tabulek a stovky tisíc záznamů. Databáze je zálohována v pravidelných intervalech a její správa je prováděna pomocí softwaru phpMyAdmin, který umožňuje veškeré funkce správy databáze přímo ve webovém prohlížeči. Mezi problémy databáze původního systému bych zařadil zejména nedostačující logování, neexistující monitoring plánovaných úloh a velikost místa na pevném disku.

3.3 Backend

3.3.1 PHP a Laravel

Dominantním programovacím jazykem původního CRM systému se stal skriptovací programovací jazyk PHP. Stalo se tak z několika důvodů. Prvním z nich byla zkušenost vývojáře s tímto programovacím jazykem. Druhým důvodem byla schopnost spustit kód na webhostingu, kde aplikace původně běžela. Třetím důvodem byla skutečnost, že pro vývoj CRM systému bylo použito jádro již z existujícího CMS systému vyvinutém rovněž v programovacím jazyce PHP.

Je jasné, že díky složitosti aplikace nemohlo být použito PHP ve své čisté (vanilla) podobě a bylo potřeba sáhnout po některém z frameworků, které poskytují sadu nástrojů a výrazně zjednodušují vývoj rozsáhlých webových aplikací. Tímto frameworkem se stal framework Laravel. Laravel je open source PHP framework pro vývoj webových aplikací vyvinutý programátorem Taylorem Otwellem a jeho týmem. První verze byla vydána v únoru 2012. Jedná se o framework poskytovaný zdarma ke stažení pod open source MIT licencí. Framework se opírá o velmi dobře známý návrhový vzor MVC (Model View Controller). Model zde obsahuje aplikační data a funkce, View slouží jakožto prezentační vrstva a zobrazuje data pomocí HTML, Controller se naopak stará a spravuje interakce mezi uživatelem, modelem a pohledem. Obrovskou výhodou frameworku, která do jisté míry podporuje jeho popularitu je, že většina věcí potřebná k vývoji webových aplikací je dostupná tzv. Out of the Box a jsou dostupné hned po nainstalování. Mezi tyto věci bychom mohli zařadit kupříkladu autentizaci, routování, nástroje pro tvoření a komunikaci s databází, nástroje pro odesílání mailů, sessions, caching a mnoho dalších.

Laravel se stal základním kamenem původního CRM systému a kvůli některým specifickým funkcím a požadavkům ze strany firmy byl rozšířen o několik externích knihoven. Mezi tyto rozšíření patří:

sboo/multiauth

Knihovna umožňující autentifikovat uživatele v rámci více modelů a tabulek. Tento plugin je v původním CRM systému proto, aby běžní uživatelé byli odstínění od manažerů a administrátorů a tyto skupiny měli odlišné modely. Knihovna je dostupná pod MIT licencí, která umožňuje použití zdrojových kódů i v komerčních projektech a vývojář je může modifikovat dle potřeby.

barryvdh/laravel-debugbar

Knihovna, umožňující pokročilý debugging v rámci Laravelu. Knihovna zobrazí v aplikaci lištu, ve které vývojář vidí spoustu užitečných informací o aplikaci a jejím stavu. Je možné zde pozorovat a analyzovat veškeré požadavky, dotazy na databázi, vyhozené výjimky, dobu trvání dotazů apod. Díky tomuto lze odhalit slabá nebo pomalá místa v aplikaci. Tato knihovna je dostupná rovněž pod MIT licencí.

simplesoftwareio/simple-qrcode

Pomocí této knihovny umožňujeme v aplikaci práci s tzv. QR kódy, což jsou prostředky pro automatizovaný sběr dat. Informace, které jsou do nich zakódovány, stačí promítnout do zařízení pomocí fotoaparátu či jiných čteček. V aplikaci jsou využity pro uložení informací o platbách. Pomocí mobilní aplikace internetového bankovnictví se veškeré údaje automaticky předvyplní a uživatelé mohou pohodlně a rychle odesílat platby. Informace o platbách jsou do QR kódu zakódovány v podobě textového řetězce, kterému se říká SPAYD (Short Payment Descriptor). Tento řetězec obsahuje základní informace o platbě, jako například částku, splatnost a příjemce.

3.4 Frontend

3.4.1 Architektura

Frontend původního CRM systému není separován od backendu a je součástí aplikace. Původní CRM systém je Multi page aplikace a veškeré HTML, šablony jsou vykreslovány na serveru a přes HTTP protokol odesílány do klientova prohlížeče. Pro serverové vykreslování je využívána technologie Laravel Blade.

Blade je jednoduchý, avšak výkonný template engine dodávaný uvnitř frameworku Laravel. Na rozdíl od jiných template enginů Blade nezakazuje používat čistý PHP kód v HTML šablonách, naopak všechny šablony jsou zkompilovány do čistého PHP kódu a uloženy do cache paměti, dokud nejsou modifikovány.

V důsledku toho, že frontend není separován, je nedostatečná modularita systému, složitější údržba a aktualizace. Vývoj by se náročněji rozděloval mezi členy týmu a serverové vykreslování je znatelně pomalejší, protože při většině uživatelových akcí musí prohlížeč znovu stáhnout veškerá data, i když tyto data v sobě již zahrnují části aplikace, které jsou na všech stránkách. To, že frontend je vykreslován na serveru a nekomunikuje

s backendem skrze API rozhraní znamená, že v tomto stavu nelze vyvinout mobilní aplikaci ani připojit jiné aplikace, které by mohly být v budoucnu v softwarovém portfoliu firmy.

Design aplikace je nakódován pomocí CSS preprocessoru SASS a je postaven na šabloně firmy Coder Themes. Design je laděn do tmavých barev a moderního stylu, avšak s přibývajícími funkcemi se začala ztrácet jednoduchost a přehlednost jednotlivých pohledů.

3.4.2 Funkcionalita

Funkcionalita prezentační vrstvy aplikace je naprogramována pomocí programovacího jazyka JavaScript, společně s populární knihovnou jQuery, a to kvůli její jednoduchosti a množství pluginů. Vedle jQuery byla použita i spousta pluginů, které pomohly vyřešit specifické potřeby aplikace. Jedná se o pluginy:

autoNumeric.js

V aplikaci se nachází spoustu polí pro zadávání čísel a zejména částek v různých měnách. Klasické HTML number input jsou pro tento use case nedostačující, a proto je v původním CRM využíván plugin autoNumeric pro jQuery, který poskytuje formátování pro čísla a většinu světových měn.

DataTables

Jeden z nejdůležitějších pluginů dosavadní aplikace. Aplikace obsahuje velké množství různých tabulek, které obsahují komplexní datové soubory. Plugin DataTables se stará o vylepšení funkcionalit klasických tabulek, jako například řazení, rozdělení na stránky, vyhledávání, vnořené tabulky apod. Jedná se o velmi dynamický plugin, ze kterého se dají data exportovat do PDF, CSV či odeslat přímo na tisk.

Flatpickr

Flatpickr se používá k jednoduššímu zadávání datumů. Aby uživatel nemusel zadávat datum ručně, po kliknutí na textový input se zobrazí kalendář, kde jednoduše datum zvolí. Plugin má spoustu funkcí a také zajišťuje jednotný formát pro celou aplikaci. Mimo rozsáhlého nastavení nabízí i podporu celé řady jazykových mutací.

Fullcalendar

Pro potřeby zobrazení interaktivního kalendáře je použit plugin Fullcalendar. Kalendář je interaktivní a slouží pro rezervaci a přihlašování k událostem. Kalendář

nabízí více pohledů – měsíční, týdenní, denní a tzv. agendu. Do kalendáře lze vkládat a mazat události. Plugin podporuje i relativně složitější principy načítání dat jako Lazy-loading. Plugin je distribuován pod MIT licencí.

Chart.js

Pro grafickou reprezentaci a vizualizaci se používají v aplikaci většinou liniové a sloupcové grafy. Tyto grafy jsou vykreslovány pomocí pluginu Chart.js, dostupného pod MIT licencí. Je postaven na HTML5 a pro vykreslování grafů využívá canvas. Podporuje i responzivní vykreslování a disponuje kvalitní a obsáhlou dokumentací.

OrgChart

Tento plugin je v aplikaci použit pro vykreslování hierarchické struktury obchodních zástupců společnosti. Rovněž je dostupný pod MIT licencí.

StyleSelect

Aby výchozí HTML select elementy ladily s designem webové aplikace a tvořili tak funkční celek, je nutné na ně aplikovat kaskádové styly. Tento plugin umožňuje stylovat výchozí select elementy tak, že vytvoří vlastní dropdown menu a překryje jím daný select. Plugin umožňuje i vyhledávání v nabízených hodnotách. Tento plugin byl vytvořen na míru pro potřeby aplikace.

Swiper.js

Pro zobrazení příspěvků na nástěnce se v CRM systému používá rotující grafický element – tzv. carousel. O implementaci tohoto elementu se stará plugin Swiper, kterému je poskytnut dataset a plugin z něj vykreslí carousel.

jQuery Validation

Jedno z nejpopulárnějších jQuery rozšíření. Rozšíření řeší problematiku validování HTML formulářů v reálném čase. Pro každý prvek ve formuláři se jednoduše nadefinuje sada pravidel – zda je nutné jej vyplnit, zda se jedná o email, minimální délka řetězce, zda se má rovnat s jiným prvkem (například heslo a ověření hesla) atd. Výhodou je i možnost doprogramování vlastních pravidel.

3.5 Uživatelé systému

Uživatelé systému se dělí na externí a interní. Mezi interní uživatele patří klienti firmy, kteří jakožto investoři svěřili svůj finanční kapitál do správy firmě. Firma tento kapitál spravuje a svěřené prostředky zhodnocuje zejména investováním do:

- měnových párů – EUR/USD, GBP/USD apod.,
- indexů – DJIA, S&P, DAX a FTSE apod.,
- komodit – a to například do cenných kovů pšenice ropy atd.,
- obligací.

Jak jsme zmínili již v úvodu této práce, systém je rozdělen do dvou hlavních částí – tzv. Office část a CRM část. Office část slouží pro externí uživatele firmy a CRM pro interní. Účelem Office částí je poskytnout klientům společnosti přístup na svůj klientský účet, kde vidí informace o svých investovaných prostředcích. CRM část slouží pro interní uživatele firmy, jako jsou obchodní zástupci, management a administrativní pracovníci. Obě části jsou navzájem propojené, ale liší se funkcemi.

3.5.1 Externí uživatelé – klienti společnosti

Klienti jsou fyzické i právnické osoby. Tyto klienty do firmy přivádí obchodní zástupci, kteří jim nabízí firemní investiční produkty. Klienti disponují klientským účtem, kterým se mohou přihlásit do systému. Po přihlášení jsou přesměrováni do Office části systému, která je rozdělena na několik sekcí dle funkcionality. V odstavcích níže si sekce Office části popíšeme z hlediska jejich funkcí a účelu.

Domovská stránka

Po přihlášení je klientům zobrazena domovská stránka, které je graficky zpracována formou nástěnky. Na nástěnce jsou zobrazeny informace pomocí karet. Každá karta zobrazuje jiná data nebo zprávy. Nachází se zde taktéž posuvník s aktuálními kurzy akcií významných společností a hodnotami nejpopulárnějších kryptoměn. Je zde také liniový graf vývoje investičního fondu společnosti a sloupcový graf průměrného měsíčního procentuálního zhodnocení kapitálu. Spodní části nástěnky dominují novinky, které zde píše management firmy. Management firmy těmito novinkami sděluje zajímavé informace klientům. Jsou zde také nahrané dokumenty ke stažení.

Můj profil

Klasická sekce většiny webových aplikací, kterou známe zejména ze sociálních sítí. Po zobrazení této sekce zde klient vidí nejdůležitější informace týkající se jeho účtu. Ať už se jedná o aktuální stav všech jeho investic, sumu vkladů nebo sumu výběrů, profilový obrázek, jméno, příjmení, bydliště a ostatní informace spojené s klientským účtem. Jsou zde také dokumenty, které buďto nahrál do systému sám klient, nebo mu je systém vygeneroval. Mezi těmito dokumenty se objevují zpravidla dodatky ke smlouvám, žádosti o výběr aj. Pod nadpisem „Mé investice“ jsou vypsány všechny investice, které má klient u firmy založené.

Nastavení

Sekce, ve které si klient může upravovat různá nastavení týkající se jeho profilu. V systému je vykresleno několik tabulek, které obsahují velké množství informací, proto si v nastavení může klient zvolit počet řádků na stránku. Dále je zde možnost změnit jazyk smluv, nahrát profilový obrázek, zapnout si dvou-faktorové ověření a popřípadě změnit své heslo.

Detail investice

Po kliknutí na danou investici se otevře její detail, kde klient vidí její aktuální peněžní stav, zisk, vklady a výběry. Vidí zde také základní informace o investici, jako například kdo je její kontaktní osobou, kdy byla investice vytvořena, k jakému je přiřazena produktu a kdy byla aktivována. Níže lze vidět kartu s dokumenty spjatými s danou investicí, grafy zobrazující vývoj investice a komplexní tabulky s investičními pohyby. Tabulky lze exportovat do formátů CSV a PDF. Je zde umístěn i QR kód pro vložení peněžních prostředků do investice.

FAQ

Sekce, kde jsou vypsány často kladené otázky a jejich odpovědi. Obsahuje také kontaktní formulář pro odeslání dotazu na podporu.

Notifikační centrum

Pokud uživatel provede nějakou změnu nebo systém provede akci, která nějakým způsobem ovlivňuje daného uživatele nebo jeho profil, je o tom informován v sekci Notifikační centrum. Tato sekce je zpracována formou ikony zvonečku, kterou také známe z mnoha populárních sociálních sítí.

3.5.2 Interní uživatelé – Obchodní zástupci

Obchodní zástupci nabízejí klientům investiční produkty firmy a uzavírají s nimi smlouvy. Obchodní zástupci pro svou práci používají CRM část systému, která jim propůjčuje více funkcí a obsahuje více sekcí. Tyto funkce jim pomáhají lépe řídit administrativu spojenou se správou svých klientů a investic, a udržovat pevnost vztahů se zákazníky, neboť jak tvrdí Lehtinen (2007, s. 30) „dobrý vztah se zákazníkem nemůže být kopírován.“. CRM část obsahuje i všechny sekce a funkce, které se vyskytují v Office části pro klienty, neboť i obchodní zástupci mohou mít rovněž investované své finance.

Nyní si představíme sekce a funkce v CRM části pro obchodní zástupce.

Nástěnka

Nástěnka je jakási domovská stránka určená pro obchodní zástupce. Obsahuje informace spojené s jejich výkonností a s výkonností jejich struktury. Na první kartě lze vidět celkovou sumu vlastních investic, počet vlastních investic a sumu vkladů. Následující karta obsahuje stejnou formu informaci, avšak v kontextu struktury daného obchodního zástupce. Jsou to tedy celkové sumy investic ve struktuře, počet investic ve struktuře, suma vkladů ve struktuře. Tyto informace jsou vyjádřeny číselně i graficky formou liniových a sloupcových grafů. Pojem struktura obchodního zástupce si vysvětlíme v odstavci zabývajícím se touto problematikou. Dále je na nástěnce vidět výběr prvních třech příspěvků na zdi, dokumenty spojené se školením obchodních zástupců a novinky pro obchodní zástupce.

Zeď

Zeď je sekce, na kterou management firmy vkládá příspěvky s informacemi, které jsou určeny obchodním zástupcům. Jednotlivé příspěvky lze zobrazit detailně a lze na ně reagovat formou komentářů, které mohou mít libovolné zanoření a hloubku. Obchodní zástupci mohou i tvořit vlastní příspěvky.

Struktura

Obchodní zástupci mohou do firmy přivádět nejenom nové klienty ale i další obchodní zástupce. Z tohoto důvodu jsou obchodní zástupci uspořádání do hierarchické stromové struktury. V této sekci tedy může obchodní zástupce vidět, koho má tzv. „pod sebou“. Klienti a další obchodní zástupci, kteří jsou ve stromové struktuře pod právě přihlášeným obchodním zástupcem, jsou zobrazeni v tabulce spolu se základními údaji, jako například datum posledního přihlášení, datum vytvoření, aktuální stav investic a suma vkladů. Z tabulky se lze také přesměrovat na profily daných uživatelů. Struktura obchodních zástupců je z hlediska ukládání do databáze řešena velmi triviálně, a proto načítání této tabulky je zejména pro obchodní zástupce umístěné vysoko ve stromové struktuře značně zdlouhavé. V tabulce, ve které jsou uživatelé uloženi, je sloupec `owner_ID`. Tento sloupec odkazuje skrze primární klíč na jiného uživatele, který je jeho přímým vlastníkem. V rámci optimalizace je nutno tento způsob ukládání struktury změnit.

Rezervace

Firma disponuje několika kancelářemi, které poskytuje svým obchodním zástupcům jako pohodlné místo pro schůzky s klienty. Aby mohl obchodní zástupce kancelář využít, musí si nejprve rezervovat schůzku na daný termín. Tento problém řeší sekce Rezervace, která obsahuje rezervační kalendář. Kalendář je vykreslen pomocí pluginu Fullcalendar.

Statistiky

Aby měl obchodní zástupce přehled o výkonnosti uživatelů ve struktuře pod ním, využívá sekci Statistiky. Sekce obsahuje obdobnou tabulku, jakou je tabulka v sekci Struktura, avšak jsou v ní zobrazeni jen obchodní zástupci a detailnější informace o nich. Jsou to zejména objemy spravovaných investic a počet nových investic za kvartál. Tabulku lze filtrovat dle počátečního a koncového data.

Affiliate

Obchodní zástupci získávají za svou práci měsíční provize. Výpočet probíhá na základě objemu portfolia, jež spravuje. Dalším aspektem při rozdělování provize je také pozice obchodního zástupce v rámci struktury. V této sekci tedy obchodní zástupce vidí výši své provize za daný měsíc a může si zobrazit dokumenty spjaté s touto provizí.

Investiční kalkulačka

Aby mohl obchodní zástupce prezentovat klientovi výhody firemního investičního fondu, může využít sekci Investiční kalkulačka. Ta funguje takovým způsobem, že ze zadaných hodnot vypočítává celkový výnos na základě výše počátečního vkladu, délku investičního horizontu, pravidelného vkladu a aktuálního průměrného měsíčního výnosu.

Školení

Pro obchodní zástupce jsou konány různá školení a přednášky, na které se mohou v sekci Školení přihlásit. Mohou hlásit jak sebe, tak ostatní obchodní zástupce ze své struktury.

Vytvoření profilu

Pokud klient uzavřel smlouvu o svěření investičních prostředků, je potřeba mu vytvořit klientský účet k přihlášení do systému. Tuto akci dělá obchodní zástupce ručně v sekci Vytvoření profilu. Stačí vyplnit potřebné údaje a systém uživatele uloží a provede inicializační nastavení profilu.

3.5.3 Interní uživatelé – Administrativní pracovníci

Administrativní pracovníci vedou administrativní agendu, starají se o provozní, organizační a jiné administrativní záležitosti. Organizují a připravují firemní jednání a zasedání, komunikují s klienty a vedou příslušnou dokumentaci. Tito uživatelé používají ty části CRM systému, které jim umožňují kontrolu různých dokumentů, přípravu podkladů a péči o zákazníky. Vykonávání zmíněných činností jim usnadní sekce CRM systému, které jsou popsány níže.

Domovská stránka s vyhledávačem

Administrativní pracovníci často potřebují něco v systému vyhledat. Ať už se jedná o klienta, který volá na podporu, investici anebo dokument.

Archív a kontakty

Pokud klient rozváže smluvní vztahy s firmou, je potřeba přenést informace o něm do archívu kvůli nařízení spojených s GDPR. Do kontaktů si mohou zaměstnanci ukládat potřebné fyzické nebo právnické osoby.

Výpis všech uživatelů

Sekce obsahující tabulku se všemi uživateli firmy a informacemi o nich. Nabízí spoustu možností filtrace. Filtrovat se dá například podle pozice, typu uživatele, nastavení atd. Z tabulky se rovněž dá přesměrovat na profil daného klienta, nebo zobrazit vyskakovací okno s výpisem jeho investic.

Výpis všech investic

Obdobná tabulka jako s klienty, avšak tato obsahuje investice.

Výpis všech dokumentů

Obdobná tabulka jako s klienty, avšak tato obsahuje dukumenty. Dokumenty lze stáhnout, zobrazit, přegenerovat a označit příznakem „v kanceláři“.

Výpis všech affiliate reportů

Affiliate reporty jsou dokumenty, které systém každý měsíc generuje každému obchodnímu zástupci. V tomto reportu obchodní zástupce vidí výši své měsíční provize a výpis investic, ze kterých mu byla provize udělena. Administrativní pracovníci mají k reportům přístup právě v této sekci a lze jim měnit stav na „Vyplaceno“, popřípadě si je zobrazit.

Žádosti o výběry

Klienti mají samozřejmě možnost vybírat ze svých peněz finanční prostředky tím, že požádají o výběr prostřednictvím tlačítka na detailu investice. Administrativní pracovníci žádosti vidí souhrnně v této tabulce a jsou zodpovědní za jejich schvalování.

3.5.4 Interní uživatelé – Management

Specifická skupina uživatelů, která po přihlášení do systému disponuje nejvyššími pravomocemi. Její zodpovědností je řídit firmu. V CRM systému používají zejména tyto níže uvedené sekce.

Platby

Aby nebylo komplikované hlídat vklady do investic, jež klienti zasílají na bankovní účet, plní tato sekce roli jakési automatické kontroly vkladů. Firma disponuje bankovním účtem u ČSOB, která v době implementace původního CRM systému nenabízela API rozhraní, skrze které by šlo tento problém řešit. Proto bylo nutné pracovat

s emaily, které banka zasílá na firemní emailovou schránku. Emaily obsahují informace o příchozích platbách. Na serveru běží CRON, který za daný časový interval spustí PHP skript, který přečte nové emaily a na jejich základě přiřazuje platby k daným investicím. Výpis těchto plateb je pak zobrazen v přehledné tabulce, kde je management poté ručně schvaluje.

Informační panel

Tato sekce slouží jako log událostí v systému. Každá akce, která v systému proběhne, je zde zaznamenána i s informací o uživateli, jenž ji provedl a objekt, nad kterým jí provedl. Příklady logovaných událostí jsou například přihlášení, žádost o výběr, aktivaci investice atd.

Plnění cílů

Management je i o motivaci a odměňování za dobře odvedenou práci. K těmto účelům slouží tato sekce, kde management vidí, jak si vedou jednotliví obchodní zástupci firmy. Je zde vidět jejich hodnota uzavřených investic, hodnota nových investic včetně zhodnocení, počet investic a počet nových investic. Pakliže hodnota investic včetně zhodnocení daného obchodního zástupce překročí předem stanovenou mez, tak se manažerovi u jména obchodního zástupce objeví možnost udělit mu odměnu.

Školení

Management v pravidelných intervalech organizuje školení a testy pro obchodní zástupce. V této sekci se dají školení tvořit, upravovat a udělovat známky, pokud se jedná o již proběhlé školení.

Novinky

Sekce, ve které může management vkládat novinky na nástěnku pro obchodní zástupce a na domovskou stránku pro klienty. Lze zde nahrávat i soubory do systému.

Pohyby u produktů

Obchodníci firmy poskytují manažerům report o svých uskutečněných obchodech na burzách. Výsledky z těchto obchodů manažeři agregují a vkládají je do systému jakožto denní pohyby. Tyto pohyby pak násobí investice klientů a tím vzniká zisk.

FAQ

Manažeři shromažďují nejčastěji kladené otázky od klientů a jiných obchodních partnerů, kteří systém využívají. Poté je v této sekci mohou ukládat a přidávat k nim odpovědi.

Emaily

Téměř 100 % emailů, které systém odesílá jsou automatizované. Někdy je však potřeba odeslat email ručně se specifickým textem a jen určitým uživatelům. V této sekci lze tedy vytvořit a odeslat emailovou zprávu, ať už konkrétním uživatelům nebo skupinám uživatelů.

Správa Affiliate

Pohled určený k vygenerování měsíčních provizí pro obchodní zástupce. Je zde tlačítko pro spuštění výpočtu a vygenerování provizní PDF sestavy. Proces trvá několik minut a s přibývajícími uživateli a investicemi se stále zpomaluje. Výstupem procesu je jeden obsáhlý PDF report pro management a jeden report pro každého obchodního zástupce.

3.6 Požadavky na optimalizaci a nové funkce

Při plánování optimalizace systému si vedení společnosti a obchodní zástupci přáli, aby se implementovali i nové funkce. Tyto nové funkce měli interním zákazníkům usnadnit další administrativní procesy probíhající v prostředí podniku a klientům zpříjemnit fungování v aplikaci.

3.6.1 Požadavky na nové funkce

Implementace elektronických podpisů

Jestliže se klient rozhodne investovat své finanční prostředky do fondu společnosti, musí s ním obchodní zástupce sepsat smlouvu o svěření finančního kapitálu. Tuto smlouvu je nutno následně osobně doručit na pobočku, kde jí musí administrativní pracovník zkontrolovat a uschovat.

Celý tento proces by výrazně zjednodušila implementace elektronických podpisů. Po vytvoření uživatelského účtu by systém vygeneroval smlouvu a klient by ji mohl s obchodním zástupcem podepsal v pohodlí domova, smlouva by se poté uložila v systému. Administrativní pracovník by byl upozorněn notifikací a smlouvu zrevidoval v rámci aplikace.

Implementace Progressive Web Apps

Dle průzkumu společnosti se zhruba 30 % klientů přihlašuje do aplikace z mobilního zařízení. Aplikace samozřejmě disponuje responzivním designem pro mobilní zařízení, avšak díky možnosti využití pokročilejších funkcí mobilních zařízení začalo vedení společnosti uvažovat o mobilní aplikaci. Vývoj nativní mobilní aplikace se neplánuje, a to zejména kvůli nedostatku lidských zdrojů a nízkého poměru přidané hodnoty pro zákazníka k nákladům na vývoj. Díky těmto faktům společnost požaduje, ať nový frontend disponuje technologií Progressive Web Apps (PWA). Tato technologie umožňuje obohatit webové aplikace o funkce, které dávají uživatelům pocit, že pracují s mobilní aplikací. PWA kombinují to nejlepší z responzivních webů a mobilních aplikací. I přes fakt, že mobilní aplikace jsou velmi populární, management společnosti zjistil, že ne všichni zákazníci jsou ochotni si mobilní aplikace stahovat a instalovat. Funkce PWA, která umožňuje přidání webové aplikace na plochu mobilního telefonu, je rychlejší než vyhledávání a instalování aplikace z obchodů aplikací (Google Play, AppStore).

Klíčovou funkcí mobilních aplikací jsou upozornění. PWA technologie také umožňuje tzv. push notifikace, které zvyšují uživatelský zájem.

3.6.2 Optimalizace stávajících funkcí

Optimalizace struktury

Jak jsme již zmínili v kapitole 3.5.2 hierarchická struktura obchodních zástupců je v databázi udržována velice triviálním způsobem. Programově se s ní pracuje dobře ale rychlost načítání se stává problémem. Vysoce postaveným obchodním zástupcům se struktura načítá až několik sekund. Pro optimalizaci bude potřeba zvolit jinou metodu ukládání hierarchických dat. Cílem je vyhnout se psaní rekurzivních a obrovských vložených dotazů.

Vylepšení automatické správy plateb

Párování plateb s investicemi podle textů emailů z banky, které jsme si popsali v kapitole 3.5.4 je nespolehlivé a čas od času se v něm objevují chyby. Za dobu funkce původního CRM systému stihla banka ČSOB vydat vlastní rozhraní pro rychlou a bezpečnou výměnu dat, kterému říká Business Connector. Cílem této optimalizace bude tedy napojit Business Connector na nový CRM systém. V rámci optimalizace je

požadováno i naprogramovat novou funkcionalitu, která bude platby naopak automaticky odesílat.

Implementace asynchronní fronty pro procesy běžící na pozadí aplikace

Všechny procesy v backendové části aplikace se nyní dějí synchronně. Díky tomuto faktu se sice program chová deterministicky a je jednoduché jej debugovat. Nicméně velkou nevýhodou tohoto přístupu je skutečnost, že zpracování déle trvajících operací, jako například odesílání emailů, se nedějí na pozadí a uživatel musí čekat na jejich vykonání. Odpovědi ze serveru tedy trvají delší dobu a při vyhození programové výjimky se musí celý proces zvrátit, aby nedošlo k nekonzistenci v datech. Tyto asynchronní fronty budou implementovány hlavně na odesílání emailů a provádění výpočtů na pozadí. V budoucnosti se však budou starat i o další procesy.

3.6.3 Požadavky na serverové zázemí

Jeden z neméně důležitých požadavků byl, aby aplikace běžela na interním serveru. V kancelářích, kde bylo umístěno i nové IT oddělení se vystavěla speciální klimatizovaná serverovna. Vedení společnosti si přálo spravovat vlastní servery z několika důvodů. Jedním z těchto důvodů byla nezávislost na externích firmách a ukládání firemních dat na vlastní úložiště.

Firma plánuje soustředit více finančních prostředků do IT oddělení a podporovat různé IT projekty. Tyto servery budou tedy v budoucnosti sloužit pro běh většího množství aplikací.

4 Návrh a implementace vybraného řešení

V této kapitole si popíšeme implementaci frontendového řešení pro nový CRM systém. Na začátku kapitoly se krátce podíváme na agilní workflow, který byl použit při vývoji. Další podkapitoly již mapují implementaci a optimalizaci funkcionalit. V této kapitole jsou zahrnuty jen ty funkce a optimalizace, které jsem sám implementoval nebo pomáhal navrhovat.

4.1 Průběh vývoje

Na vývoj nového CRM systému se alokovaly dva programátoři na backendovou stranu aplikace, jeden frontend vývojář, tester a jeden správce serverů. Celý tým vedl zkušený team leader, který pomáhal s implementací složitějších funkcionalit. Vývoj se opíral o know-how agilního vývoje softwaru a Kanban proces.

Celý projekt se implementoval v týdenních sprintech na jejichž konci se kompiloval testovací build pro zákazníka. Pro správu sprintů a administrativu spojenou s rozdělováním úkolů zaměstnancům se používal známý software Jira od společnosti Atlassian, Inc. Tým komunikoval prostřednictvím platformy Microsoft Teams a zdrojový kód byl verzován v GitLabu.

Každý zaměstnanec po přihlášení do softwaru Jira vidí u každého projektu Kanban tabuli. Ve sloupci backlog si našel úkol, který mu byl přiřazen a pustil se do vypracování analýzy daného problému. Analýza je dokument, který popisuje návrh řešení daného problému a odhadovanou časovou náročnost. Analýzu zrevidoval team leader a pokud ji schválil, tak zaměstnanec může na úkolu pracovat.

Aby změny ve zdrojovém kódu daného zaměstnance neovlivňovaly funkcionality ostatních zaměstnanců, využíval se v nástroji git feature branch přístup. V tomto přístupu se pro každou novou funkcionalitu tvoří zvlášť větev a s ní požadavek na merge do hlavní větve zdrojového kódu. Na schvalování merge requestů má práva pouze team leader. Jakmile zaměstnanec dokončí činnost v dané větvi, team leader provede nad touto větví code review a pokud se změnami souhlasí, tak request schválí a git automaticky zapojí zaměstnancovu větev do hlavní větve.

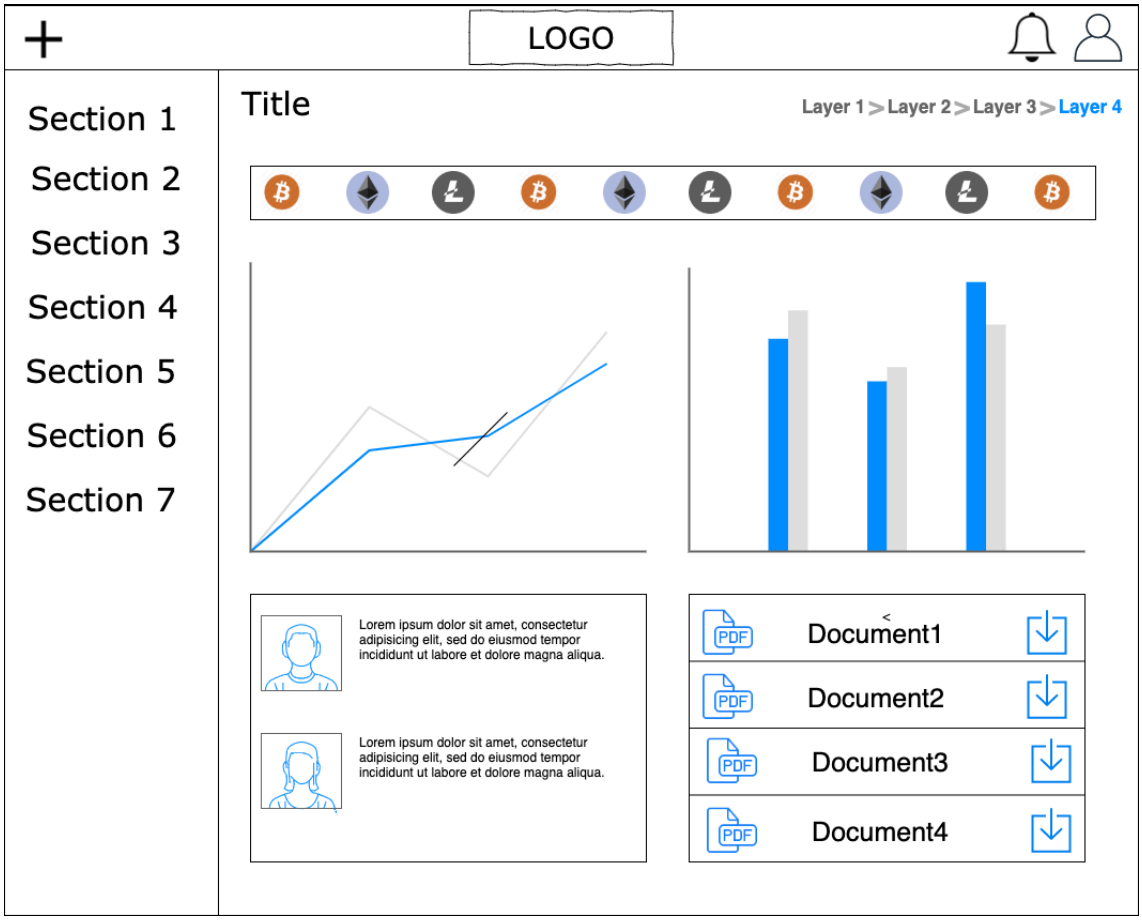
4.2 Návrh a kódování designu

Ještě před samotným programováním nového frontendu aplikace se navrhoval modernější a jednoduší design. Při úzké spolupráci s externím grafikem byl navržen zcela

nový design, jehož dominantní barvy reprezentovali brand firmy. Na základě požadavků společnosti byl design systému navrhnut ve dvou provedeních – světlý a tmavý, mezi kterými bude moci uživatel ve svém nastavení pohodlně přepínat. Design byl nakreslen pomocí platformy Sketch a prezentován klíčovým lidem ze společnosti v nástroji inVision, kteří design odsouhlasili.

Odsouhlasením započala fáze kódování designu do formy HTML šablon, do kterých se v pozdějších kapitolách implementovaly programové funkcionality.

Design kopíruje tzv. admin dashboard princip, ve kterém je obrazovka rozdělena na tři logické části. Horní panel, levé menu s navigací a obsahová část. Tento princip je zobrazen na návrhu wireframe.



Obrázek 4.1: Wireframe návrh domovské stránky (zdroj: interní zdroj)

Na obrázku vidíme základní wireframe návrh grafické struktury aplikace. Design aplikace byl stylován a pozicován pomocí CSS preprocesoru SASS, který výrazně obohacuje funkčnost klasického CSS. Po dokončení kódování byl použit nástroj Gulp.js, který SASS zdrojový kód zkompiloval do jednoho CSS souboru. Tento soubor byl poté vystaven minifikaci. Minifikace je proces, kdy jsou ze zdrojového kódu interpretovaných

programovacích jazyků odstraněny veškeré nepotřebné znaky, aniž by došlo ke změně jeho funkcionality. Tento proces výrazně zredukuje výslednou velikost souboru.

4.3 Návrh modulů a komponentů

V této kapitole se již dostáváme ke zmiňovanému frameworku Angular a jeho modulárnímu přístupu k tvorbě webových aplikací. Popíšeme si, jak byl systém rozdělen z hlediska jednotlivých modulů. Moduly jsou zde chápány jako části aplikace, související s určitou funkcionalitou. Moduly jsou navrhovány tak, aby byly na sobě nezávislé a daly se libovolně přidávat, mazat či upravovat.

4.3.1 AppModule – Kořenový modul

Každá aplikace vytvořená ve frameworku Angular musí mít alespoň jeden kořenový modul, dle konvencí pojmenován jako AppModule. Aplikaci spouštíme tak, že aplikujeme proces bootstraping na kořenový modul.

4.3.2 CoreModule – Jádro systému

Modul obsahující základní prvky, bez kterých by aplikace nemohla fungovat. Obsahuje komponenty spojené s rozložením aplikace na náš dashboard layout. Tento modul importuje RouterModule, který nám umožní v aplikaci využívat funkcionalit tzv. Routování neboli navigaci z jednoho pohledu na jiný, když uživatel provede určité akce.

Komponenty deklarované v modulu:

MainComponent

Hlavní komponent aplikace, který je využíván jako jakýsi „hub“. Komponent je načten po úspěšném přihlášení uživatele a je rodičem všech ostatních komponentů dostupných pro autentizované uživatele. HTML šablona tohoto komponentu obsahuje „kostru“ aplikace, která je určitým vzhledovým rámcem pro ostatní vizuální prvky.

NavbarComponent

Komponent určen pro implementaci horního panelu. Krom loga společnosti se v něm nachází dvě kontextová menu. Jedno slouží pro rychlou navigaci na profil uživatele, nastavení uživatele a odhlášení z aplikace. Druhé kontextové menu slouží pro zobrazení notifikací a odkaz do notifikačního centra. Notifikace jsou součástí notifikačního modulu, který si později představíme.

Mezi zdroji, které tento modul importuje je i tzv. CommonModule. Neboť CommonModule poskytuje všechny základní direktivy a pipes běžně potřebné ve všech ostatních komponentech aplikace.

LeftMenuComponent

Součástí každé webové aplikace je nějaký druh navigačního menu, ve kterém si uživatel vybírá, do které části aplikace se chce přesunout. V našem případě se jedná o boční panel nacházející se v levé části obrazovky. O vykreslení tohoto panelu se stará LeftMenuComponent. Položky v menu jsou dynamické, a ne každý uživatel má stejný počet položek. Aby komponent věděl, které položky v menu zobrazit, musí vyslat požadavek metodou GET na server. Server v URL adrese požadavku očekává ID uživatele, pomocí kterého uživatele identifikuje a zašle mu obsah menu, na který má oprávnění.

FooterComponent

Komponent sloužící pro vykreslení patičky webové aplikace. Patička obsahuje informace o copyright a aktuální verzi systému.

4.3.3 SharedModule – sdílené zdroje pro celou aplikaci

Různorodé zdroje, jakými jsou například třídy a knihovny, které je nutné sdílet napříč celou aplikací obsahuje právě SharedModule neboli sdílený modul. Tento modul importují všechny ostatní moduly aplikace.

Komponenty deklarovány v modulu:

FilesComponent

Grafický prvek zobrazující nahrané soubory.

DocumentsComponent

Grafický prvek zobrazující vygenerované dokumenty.

LineChartComponent

Komponent určen k vykreslení liniového grafu. Pro vykreslení je nutno komponentu předat dva vstupní parametry: časové řady a popisky časových řad.

BarChartComponent

Velmi podobný komponent jako LineChartComponent. Disponuje i stejnými vstupními parametry.

ModalFileUploadComponent

Komponent, který zobrazí dialogové okno pro nahrání souboru.

CounterupComponent

Komponent, který je využíván pro zobrazení číselných údajů.

4.3.4 AuthModule – modul pro autentizaci

Modul určený pro autorizaci a autentizaci uživatele. Modul je načten zpravidla po prvním načtení aplikace, kde je uživatel vyzván, aby zadal své přihlašovací údaje.

Komponenty deklarované v modulu:

LoginFormComponent

Vykresluje přihlašovací obrazovku, obsahuje formulář pro vyplnění emailové adresy a hesla.

LogoutComponent

Komponent, na který je uživatel přesměrován po úspěšném odhlášení.

4.3.5 ProfileModule – profil uživatele

Komponenty deklarované v modulu:

ProfileComponent

Komponent, který zobrazuje profil daného uživatele. Pokud je přihlášený uživatel klientem, tak si může zobrazit pouze svůj profil. Obchodní zástupci jsou oprávněni zobrazovat si profily klientů a ostatních obchodních zástupců v rámci své struktury.

CreateEditCustomerComponent

Komponent vykreslující formulář pro vytvoření nového uživatelského účtu.

SettingsComponent

Obrazovka pro ukládání různorodých nastavení uživatelského profilu.

BeneficiaryComponent

Grafický prvek zobrazující kontaktní osoby daného klienta.

InvestmentComponent

Grafický prvek zobrazující souhrnné informace o dané investici. Prvek je interaktivní a po kliknutí je uživatel přesměrován na danou investici.

InvestmentCreateModalComponent

Dialogové okno, sloužící obchodním zástupcům pro tvoření nových investic.

4.3.6 InvestmentModule – modul pro práci s investicí

Načtení tohoto modulu do paměti prohlížeče nastává tehdy, jakmile si uživatel zobrazí detail investice.

Komponenty deklarované v modulu:

InvestmentDetailComponent

Detailní informace o investičním účtu. Jeden ze stěžejních komponentů aplikace.

InvestmentCalculatorComponent

Komponent zajišťující funkci investiční kalkulačky, jakožto prezentačního nástroje výhodnosti firemního investičního fondu pro klienty.

InvestmentWithdrawRequestsComponent

Grafický prvek zobrazující uživatelovy žádosti o výběry z investičního účtu.

4.3.7 DocumentModule – modul s dokumenty

V aplikaci je možno archivovat, zobrazovat a stahovat PDF dokumenty. Tento modul je využíván pro funkce spojené s dokumenty.

Komponenty deklarované v modulu:

DocumentDetailComponent

Pohled, který slouží k zobrazení PDF dokumentu uvnitř aplikace. Obsahuje export na fyzické zařízení, stránkování, zoom a ostatní užitečné funkce pro čtení PDF souborů.

OfficeDocumentListComponent

Tabulka obsahující veškeré nahrané dokumenty v systému, které má oprávnění vidět právě přihlášený uživatel.

4.3.8 DashboardModule – modul s nástěnkami

Modul určený pro zobrazování nástěnek v aplikaci. Nástěnka je typ grafického uživatelského rozhraní, které často poskytuje přehledné pohledy na klíčové ukazatele výkonu (KPI), relevantní pro konkrétní uživatele nebo typy uživatelů.

Komponenty deklarovány v modulu:

HomeComponent

Domovská stránka s nástěnkou určenou pro klienty společnosti.

DashboardComponent

Domovská stránka s nástěnkou určenou pro obchodní zástupce firmy.

NewsComponent

Grafický prvek zobrazující novinky na dané nástěnce.

WallPartialComponent

Grafický prvek obsahující první tři nejnovější příspěvky ze zdi.

4.3.9 StructureModule – modul pro hierarchickou strukturu

Modul pro implementaci hierarchické struktury obchodních zástupců.

Komponenty deklarované v modulu:

StructureComponent

Komponent vykreslující tabulku, ve které obchodní zástupci tráví většinu času v aplikaci. Zobrazuje jim totiž strukturu uživatelů v rámci jejich hierarchické struktury. Tato tabulka dále slouží jako jakýsi rozcestník mezi profily podřízených uživatelů. Tabulka zobrazuje pouze úroveň struktury uživatelů bezprostředně pod přihlášeným obchodním zástupcem.

TreeComponent

Grafický prvek zobrazující celý stromový graf hierarchické struktury počínaje přihlášeným obchodním zástupcem.

StatisticsComponent

Obdobná tabulka jako ve Structure komponentu. Tato tabulka však obsahuje více informací týkajících se výkonnosti obchodních zástupců, umístěných pod přihlášeným obchodním zástupcem. Tato tabulka implicitně nezobrazuje klienty.

4.3.10 ReservationModule – modul pro rezervaci kanceláří

Modul importující jednu z mála externích knihoven – FullCalendar. Modul obsahuje pouze jeden komponent zobrazující kalendář pro zarezervování kanceláře v daném městě.

Komponenty deklarované v modulu:

ReservationComponent

Jak jsme již zmínili výše, tento komponent zobrazuje kalendář s rezervacemi. Nad kalendářem jsou tlačítka pro zvolení lokality a typu pohledu. K dispozici jsou pohledy měsíční, týdenní a denní.

4.3.11 WallModule – zeď s příspěvky

Modul pro implementaci zdi s příspěvky.

Komponenty deklarované v modulu:

WallComponent

Výpis všech příspěvků na zdi.

WallPostComponent

Grafický prvek zobrazující jeden příspěvek na zdi. Prvek je interaktivní, po kliknutí je uživatel přesměrován na detail příspěvku.

WallPostDetailComponent

Pohled zobrazující detail příspěvku a výpis komentářů asociovaných s tímto příspěvkem. V tomto pohledu lze i reagovat na daný příspěvek.

CreateEditWallPostComponent

Formulář pro tvoření nebo úpravu příspěvku na zdi.

4.3.12 AffiliateModule – modul pro affiliate provize

Modul sloužící obchodním zástupcům pro přehled o jejich měsíčních provizí. Tento modul obsahuje i funkcionality pro generování faktur.

Komponenty deklarované v modulu:

AffiliateListComponent

Komponent obsahující tabulku s měsíčními provizemi.

4.3.13 BeneficiaryModule – modul pro kontaktní osoby

Modul, jehož účelem je zapouzdřovat funkcionality spojené s administrací kontaktních osob.

Komponenty deklarované v modulu:

CreateEditBeneficiaryComponent

Formulář pro editaci nebo vytvoření kontaktní osoby a přiřazení k investici.

4.3.14 NotificationModule – modul pro notifikace

Tento modul spravuje uživatelské notifikace ohledně akcí prováděných systémem a jeho uživateli.

Komponenty deklarované v modulu:

NotificationComponent

Notifikační centrum, kde jsou zobrazeny všechny notifikace pro daného uživatele. Každá notifikace má své datum vytvoření, typ a popis. Notifikaci lze nastavit jako přečtenou.

4.3.15 FaqModule – modul pro troubleshooting

Modul obsahující komponenty, jenž poskytují nápovědu uživatelům nebo kontakt na podporu.

Komponenty deklarované v modulu:

FaqComponent

Komponent obsahující často kladené otázky a jejich řešení.

ContactFormComponent

Komponent obsahující formulář pro kontaktování podpory.

4.3.16 TrainingsModule – modul se školeními

Modul určen pro administrativu spojenou se školením obchodních zástupců.

Komponenty deklarované v modulu:

TrainingsComponent

Tabulka obsahující výpis všech vypsaných školení a přednášek. Z tabulky se lze přesměrovat na detail daného školení či přednášky.

TrainingDetailComponent

Detail školení, v němž vidíme přihlášené uživatele, počet volných míst, místo a čas konání. Je zde i formulář pro přihlášení.

4.3.17 Lazy-loading modulů

Ve výchozím nastavení jsou NgModuly načteny tzv. Eagerly-loaded, což znamená, že jakmile se aplikace načte, načtou se také všechny NgModuly, ať jsou okamžitě potřebné nebo ne. U velkých aplikací se spoustou pohledů je kvůli optimalizaci dobré zvážit implementaci Lazy-loadingu – návrhového vzoru, který NgModuly načte podle potřeb. Lazy-loading pomáhá udržovat počáteční velikost aplikace menší, což zase pomáhá zkrátit dobu načítání.

Abychom aktivovali Lazy-loading modulů je potřeba obohatit nadefinované cesty v aplikaci o parametr `loadChildren`, který je zobrazen na ukázce kódu 4.1 níže.

Ukázka kódu 4.1: Příklad použití `loadChildren` (zdroj: interní zdroj)

```
const routes: Routes = [
  {
      path: 'auth',
      loadChildren: () =>
import('./auth/auth.module').then(m => m.AuthModule)
  }
];
```

Všimněme si, že syntaxe Lazy-loadingu obsahuje parametr `loadChildren`, který používá vestavěnou syntaxi importu prohlížeče. Cesta importu je relativní cestou k modulu.

4.3.18 Měření rychlosti načítání modulů

Po optimalizaci spočívající v modularizaci zdrojového kódu frontendu, kterou popisuje kapitola 4.3, získala aplikace požadovanou škálovatelnost a rychlost vykreslování uživatelského rozhraní. Po implementaci modulů bylo provedeno měření rychlosti načtení přihlašovací obrazovky a rychlosti přecházení mezi pohledy. Měření bylo provedeno jak u starého CRM systému, tak i u nového. Časy byly měřeny pomocí vývojářských nástrojů zabudovaných ve webovém prohlížeči Chrome od společnosti Google. Aby výsledky nezkreslovala rychlost připojení k internetu, byla měření prováděna v lokální síti. Výsledky jsou zaznamenány v tabulce 4.1.

Tabulka 4.1: Měření rychlosti po optimalizaci návrhovým vzorem Lazy-loading (zdroj: interní zdroj)

Akce	**Rychlost a objem dat v původním CRM**	**Rychlost a objem dat v aktuálním CRM**
Načtení přihlašovací obrazovky	787 ms 1.5 MB	158 ms 496 KB
Přesměrování z domovské stránky na profil uživatele	1,82 s 6.1 MB	65 ms 60,6 KB

Jak vidíme v tabulce 4.1 rychlost načítání zdrojů mezi jednotlivými pohledy se razantně zvětšila. V původním CRM systému bylo dosaženo horších výsledků zejména díky faktu, že celý pohled musel být vygenerován na serveru a poté odeslán do klientova prohlížeče i se všemi externími zdroji. To v případě SPA není nutné. Při zobrazování profilu uživatele se pouze načetl korespondující ngModul a frontend vyslal HTTP požadavek pro získání informací o daném uživateli. Tyto informace ve velice krátkém čase obdržel ve formátu JSON.

4.4 Optimalizace struktury

Mezi požadavky na optimalizaci původních funkcí, které jsou obsaženy v kapitole 3.6.2 se nachází požadavek na zrychlení práce s hierarchickou strukturou obchodních zástupců. Stávající řešení ukládání struktury do databáze neumožňuje funkci optimalizovat a je třeba implementovat nové řešení.

Nynější řešení spočívá v tom, že v tabulce, ve které jsou ukládáni uživatelé, má každý záznam v sloupci `owner_ID` svého bezprostředního nadřazeného obchodního zástupce. Tento přístup obsahuje několik problémů:

- Můžeme dotazovat pouze na dvě úrovně, protože mezi rodičem a dítětem existuje přímý vztah,
- Chceme-li se dále dotazovat hlouběji do hierarchie, musíme přidat více JOIN příkazů,
- Před opětovným vytvořením hierarchie musí aplikace načíst všechny řádky z databáze do aplikace, což zapříčiní větší zátěž pro paměťové zdroje,
- Udržování referenční integrity je obtížné, protože přesouvání nebo mazání záznamů vyžaduje řadu kroků.

Pro účely optimalizace těchto problémů byl zvolen návrhový vzor pro ukládání hierarchických dat Closure Tables. Tento přístup ukládá všechna metadata o hierarchii, tj. vztazích mezi rodičem a dítětem, do samostatné tabulky, čímž zajišťuje čisté oddělení od hlavní datové tabulky. Pomocí tohoto přístupu je tedy možné ukládat hierarchie neurčité hloubky.

4.4.1 Příprava databáze

Byla vytvořena samostatná tabulka `customer_structure`, která bude uchovávat vztah a hloubku vztahu rodič – dítě. Tabulka byla vytvořena migrací, která je ukázána na ukázce kódu 4.2.

Ukázka kódu 4.2: Migrace tabulky `customer_structure` (zdroj: interní zdroj)

create_customer_structure_table.php

```
public function up() {
   Schema::create('customer_structure', function (Blueprint
$table) {
   $table->unsignedInteger('ancestor_ID')->index();
   $table->unsignedInteger('descendant_ID')->index();
   $table->unsignedInteger('depth')->index()->default(0);
   $table->unsignedInteger('root')->index()->default(0);
   $table->unique(['ancestor_ID', 'descendant_ID']);
   $table->index(['ancestor_ID', 'descendant_ID','depth']);
            $table->index(['descendant_ID', 'depth']);
            $table->index(['depth', 'root']);
            $table->foreign('ancestor_ID')
                  ->references('customer_ID')
                  ->on('customer')
                  ->onDelete('cascade')
                  ->onUpdate('cascade');
            $table->foreign('descendant_ID')
                  ->references('customer_ID')
                  ->on('customer')
                  ->onDelete('cascade')
                  ->onUpdate('cascade');
        });
}
```

Sloupec `ancestor_ID` ukazuje na nadřazeného uživatele, `descendant_ID` na potomka a `depth` odkazuje na hloubku stromu. Sloupec `root` říká, zda se jedná o vrchol stromu.

Pro příklad, uložení jediného uživatel bez podřízených uživatelů v tabulce `customer_structure` je zobrazen v tabulce 4.2.

Tabulka 4.2: Uložení jediného uživatele bez potomků (zdroj: interní zdroj)

ancestor_ID	descendant_ID	depth
1	1	0

Stejný uživatel s jedním potomkem by byl uložen jako v tabulce 4.3.

Tabulka 4.3: Uložení uživatele s jedním potomkem (zdroj: interní zdroj)

ancestor_ID	descendant_ID	depth
1	1	0
1	2	1
2	2	0

Nyní nám stačí relativně jednoduché dotazy pro získání struktury daného obchodního zástupce.

Ukázka kódu 4.3: Dotaz pro získání struktury obchodního zástupce s ID 31 (zdroj: interní zdroj)

```
select * from `customer`
inner join `customer_structure`
on `customer`.`customer_ID` =
`customer_structure`.`descendant_ID`
where `customer_ structure`.`ancestor_ID` = 31
and `customer_structure`.`depth` = 1
```

Jedinou nevýhodou tohoto přístupu, je potřeba udržovat tabulku `customer_structure` aktuální při každém vložení nebo odstranění uživatele. Ve srovnání s jinými přístupy je však zdaleka nejúčinnější.

4.4.2 Implementace na frontendu

Z pohledu frontendu byl komponent zobrazující tabulku se strukturou přepracován tak, že volá API endpoint pro výpis struktury přihlášeného uživatele pomocí nově vytvořené Closure tabulky. Do komponentu byla implementována i filtrace podle typu uživatele.

Po inicializaci komponentu je vyslán HTTP požadavek metodou GET pro získání struktury daného obchodního zástupce. V URL požadavku je obsažen identifikační řetězec uživatele tzv. slug. API vrací Observable s polem objektů Customer. Komponent nad tímto polem provede filtraci a vykreslí tabulku. Metodu pro filtraci uživatelů vidíme na ukázce kódu 4.4 níže.

Ukázka kódu 4.4: Metoda pro filtraci uživatelů (zdroj: interní zdroj)

```
  filterStructure(typeArray, refreshTable: boolean) {
    this.customers = this.customersBatch.filter(
      (customer: any) => {
      if (typeArray.includes('NO_ZERO_AMOUNT')) {
        return typeArray.includes(customer.type)
        && customer.own_sum_investment_amount > 0;
      }
      return typeArray.includes(customer.type);
    });
    if (refreshTable) {
      this.dtHelper.rerenderDataTable(
       this.dtElement,this.dtTrigger
      );
    }
  }
```

4.4.3 Měření rychlosti načítání struktury

Měření bylo prováděno pro strukturu od obchodního zástupce s ID 31, který je ve stromové struktuře velmi vysoko a jen v úrovni přímo pod sebou má 65 obchodních zástupců. Měření bylo opět provedeno skrze vývojářské nástroje prohlížeče Chrome. Postup spočíval v zobrazení struktury obchodního zástupce v původním CRM systému, kde se načítala z databáze pomocí sloupce `owner_ID` a poté zobrazením stejné struktury v novém CRM systému, kde se načítala z databáze pomocí návrhového vzoru Closure Table. Výsledky jsou zaznamenány v tabulce 4.4.

Tabulka 4.4: Měření rychlosti načítání struktury (zdroj: interní zdroj)

	Původní CRM (owner_ID)	Nové CRM (Closure Table)
Rychlost	8,26 s	2,43 s

4.5 Vylepšení správy plateb

Jelikož funkcionalita, která se starala o párování plateb přijatých na bankovním účtu s investicemi byla nevyhovující a objevovalo se v ní určité procento chybovosti, tak bylo rozhodnuto o využití nové služby od ČSOB Business Connector. Tato služba umožňuje automatizovanou komunikaci s bankou v podobě přenosu souborů, jako jsou výpisy z účtu, avíza a kurzovní lístky. Výhodou je, že služba umožňuje i přijímat dávky

platebních příkazů od klienta. Tato výhoda uspokojí požadavek vedení společnosti, které si přálo, aby nový CRM systém uměl automaticky vyplácet schválené žádosti o výběr.

4.5.1 Optimalizace příchozích plateb

Pro backendovou implementaci byla použita oficiální knihovna od ČSOB asisteam/csob-bc, která umožňuje pracovat s Business Connectorem z prostředí PHP zdrojového kódu. Místo parsování textů z emailů od banky se každé čtyři hodiny spouští PHP skript, jenž si stáhne pomocí Business Connectoru nové platby na bankovním účtu a přiřadí je k investicím, jelikož je klient při odesílání platby povinen zadat kód investice jako variabilní symbol. Skript podle variabilního symbolu vyhledá konkrétní investici a připíše do ní danou částku. Pokud se mu investici nepodaří nalézt, tak odešle informaci administrativnímu pracovníkovi, který situaci řeší dál.

Ukázka kódu 4.5: Vyhledávání investice podle variabilního symbolu (zdroj: interní zdroj)

```
$variableSymbol = $ap->getVariableSymbol();
$investment = $this->
investmentRepository->findInvestmentByVS($variableSymbol);

if ($investment == null) {
Log::channel('csob')->debug('Payment::['.$ap->
bank_account_payment_ID.']::investment_not_found_by_vs::try
ing_by_message');
$message = trim($ap->getMessage());
$investment = $this->
investmentRepository->findInvestmentByCode($message);
}
```

Do procesu bylo implementováno i zaznamenání všech událostí (logování) pro řešení případných problémů. Od implementace Business Connectoru pro příchozí platby se již neděly problémy s párováním investic, protože za aplikačního rozhraní banky chodí přesná data ve striktně stanoveném formátu. Jediným problémem v této implementaci je běžné selhání lidského faktoru, kdy klient opomene vyplnit variabilní symbol.

4.5.2 Implementace odesílání dávek platebních příkazů

Jakmile klient požádá o výběr ze své investice, administrativní pracovník je povinen tento výběr zkontrolovat a přiřadit mu stav. Pokud je výběr schválen je nastaven stav na „schváleno“. Aby se dávky platebních příkazů nemuseli tvořit ručně, byl napsán PHP skript, který každý den iteruje skrz žádosti o výběr ve stavu „schváleno“ a rozdělí je do dávek po dvaceti platbách. V rámci bezpečnosti a kontroly nejsou dávky hned

odeslány, ale zobrazí se jako výpis v CRM systému. Tento výpis kontroluje management společnosti a pokud je vše v pořádku, mohou v rámci systému potvrdit jejich odeslání.

Pro účely ukládání platebních dávek byly vytvořeny dvě nové tabulky v databázi. První tabulka, která se jmenuje `outgoing_payment`, reprezentuje jednotlivé odchozí platby. Druhá tabulka, pojmenována `outgoing_payment_batch`, reprezentuje jednotlivé platební dávky. Každá platební dávka může mít až dvacet odchozích plateb, proto je mezi tabulkami relace 1:N.

Aby mohl management v CRM systému platební dávky spravovat, bylo potřeba na frontendu vygenerovat dva nové komponenty. Jeden se stará o výpis všech vytvořených platebních dávek a druhý zobrazuje detail dávky a informace o obsažených platbách.

První komponent obsahuje pouze tabulku, která si po své inicializaci vyžádá data z backendu. Backend vrátí výpis nejnovějších platebních dávek v systému. Metoda pro získání dávek je zobrazena na ukázce kódu 4.6 níže.

Ukázka kódu 4.6: Metoda pro získání dávek (zdroj: interní zdroj)

```
  getPaymentsBatches() {
    this.ngxService.start();
    this.bankService.getBankAccountPaymentBatches()
    .subscribe((batches: BankAccountPaymentBatch[]) => {
      this.paymentsBatches = batches;
      this.dtTrigger.next();
      this.ngxService.stop();
    }, error => {
      this.ngxService.stop();
      swal({
        type: 'error',
        toast: true,
        text: 'Dávkové soubory se nepodařilo načíst',
        timer: 2500
      });
    });
  }
```

Po kliknutí na platební dávku je uživatel přesměrován na komponent zobrazující detail dávky. Po načtení je odeslán HTTP požadavek na server, který vrací podrobné informace o dávce a platbách v ní obsažených. Management provede kontrolu dávky. Platby může přidávat nebo i odstranit. Po dokončení kontroly lze dávku odeslat kliknutím

na tlačítko. Tlačítko vyšle HTTP požadavek metodou POST, v jehož těle se nachází ID platební dávky. Server dávku zpracuje a odešle do Business Connectoru.

4.6 Implementace asynchronní fronty pro procesy běžící na pozadí aplikace

Aby byli uživatelé aplikace odstíněni od náročných procesů, které by mohly být spuštěny asynchronně na pozadí, byla použita asynchronní fronta. Asynchronní frontu v prostředí frameworku Laravel implementujeme pomocí funkcionality Laravel Queues.

Laravel Queues poskytují sjednocené API napříč celou řadou různých systémů pro řízení front, jako jsou Beanstalk, Amazon SQS, Redis nebo dokonce relační databáze. Fronty umožňují odložit zpracování časově náročného úkolu, například odeslání e-mailu, na pozdější čas. Odložení těchto časově náročných úkolů na pozadí drasticky urychluje webové požadavky na naši aplikaci.

Jelikož budou fronty využívány prozatím na výpočty provizí a odesílání emailu, není nutné používat služby třetích stran, jako například Beanstalk. Pro tento use case nám postačí naše existující relační databáze.

Chceme-li tedy použít databázi pro řízení front, budeme potřebovat tabulku k uložení úloh. Strukturu tabulky zobrazuje migrace na ukázce kódu 4.7 níže.

Ukázka kódu 4.7: Migrace tabulky `jobs` (zdroj: interní zdroj)

```
public function up() {
   Schema::create('jobs', function (Blueprint $table) {
      $table->bigIncrements('id');
      $table->string('queue')->index();
      $table->longText('payload');
      $table->unsignedTinyInteger('attempts');
      $table->unsignedInteger('reserved_at')->nullable();
      $table->unsignedInteger('available_at');
      $table->unsignedInteger('created_at');
   });
}
```

Třídy pro jednotlivé úlohy jsou velmi jednoduché a obvykle obsahují pouze metodu handle, která se volá, když je úloha zpracována ve frontě. Třída úlohy pro odesílání emailů vypadá následovně.

Ukázka kódu 4.8: Úloha pro odesílání emailů (zdroj: interní zdroj)

```
class SendEmail implements ShouldQueue
{
  use Dispatchable, InteractsWithQueue, Queueable;

  protected $emailObject;
  protected $recipientInfo;

  public function __construct($recipientInfo, $emailObject)
    {
        $this->emailObject = $emailObject;
        $this->recipientInfo = $recipientInfo;
    }

  public function handle()
    {
     Mail::to(
$this->recipientInfo->email,
$this->recipientInfo->name)->send($this->emailObject);
    }
}
```

Po napsání třídy jí můžeme v kódu odeslat do fronty pomocí metody `dispatch()`. Argumenty předané metodě `dispatch()` budou převzaty konstruktorem dané úlohy.

Laravel obsahuje tzv. Queue Worker, který bude zpracovávat nové úlohy, jakmile jsou přidány do fronty. Queue Worker se spouští pomocí speciálního příkazu. Po jeho spuštění bude vykonávat všechny nové úlohy zařazené do fronty, dokud nebude ručně zastaven.

Je třeba mít na paměti, že Queue Worker je proces s dlouhou životností a ukládá si zdrojový kód aplikace do paměti. V důsledku toho si Queue Worker nevšimne změn v našem zdrojovém kódu. Během procesu nasazení nové verze aplikace se tedy nesmí zapomenout všechny Workery restartovat.

4.7 Implementace elektronických podpisů

Pro implementaci elektronických podpisů na frontendové straně CRM systému byl vytvořen nový komponent a umístěn do existujícího modulu pro práci s dokumenty – `DocumentModule`. Komponent reprezentuje dialogové okno s podpisovou plochou, na kterou lze psát buďto myší nebo dotykovou tužkou na mobilních zařízeních. Komponent

musí být navrhnut jako obecný a znovupoužitelný, neboť bude použit na podepisování vícera typů dokumentů.

Komponent je v rámci anotace `@ngModule` umístěn v poli `entryComponents`, neboť se na něj nedotazujeme selektorem, ale přímo instancí třídy. Komponent vykreslujeme, když uživatel klikne na tlačítko „Podepsat“. Toto tlačítko spustí metodu, jenž provede inicializaci komponentu.

Ukázka kódu 4.9: Metoda pro inicializaci komponentu (zdroj: interní zdroj)

```
this.modalService.open(SignaturePadModalComponent, {size:
'lg'});
```

Funkcionalitu podpisového pole zajišťuje vnořený komponent `<signature-pad></signature-pad>`, jehož vstupním parametrem je nastavení podpisového pole. V nastavení se specifikují parametry, jako například barva písma, šířka a výška podpisového pole, šířka tahu pera v pixelech apod.

Po dokončení podpisu je spuštěna funkce `drawComplete()`, ve které uložíme nakreslený podpis do formátu base64 a spolu s identifikačním číslem dokumentu je odeslán na server, kde je s dokumentem spojen a uložen do databáze. Implementaci metody `drawComplete()` můžeme vidět na ukázce kódu 4.10 níže.

Ukázka kódu 4.10: Implementace metody `drawComplete()` (zdroj: interní zdroj)

```
drawComplete() {
  this.signature = this.signaturePad.toDataURL();
  if (this.signer === 'CUSTOMER') {
  this.file.customerSignature$.next(this.signature);
  } else {
  this.file.representativeSignature$.next(this.signature);
  }
}
```

4.8 Implementace Progressive Web Apps

Z důvodů popsaných v kapitole 3.6.1 se v této podkapitole budeme zabývat implementací PWA do našeho nového frontendu. Funkcionality PWA se implementují skrze tzv. Service Workers.

Service Worker je skript, který pracuje na pozadí a komunikuje s téměř všemi moderními prohlížeči. Service Workers fungují skrze HTTPS a pracují stejným způsobem jako Web Workers. Progresivní webová aplikace považuje Service Workers

za primární technologii. Umožňuje hlubokou integraci platformy, jako je podpora offline, synchronizace na pozadí, ukládání do mezipaměti a push notifikace.

Service Workers rozšiřují tradiční model vývoje webových aplikací a umožňuje jim poskytovat uživatelské pohodlí se srovnatelným výkonem jako u nativně nainstalovaných aplikací. Přidáním Service Workera do Angular aplikace je jedním z kroků pro přeměnu na PWA. Angular implementaci PWA silně podporuje již od verze 5 a poskytuje vývojářům propracované nástroje pro tento use case.

4.8.1 Přidání PWA funkcí

Pro přidání Service Workera a ostatních důležitých zdrojových souborů pro PWA je nejprve nutné spustit následující příkaz.

Ukázka kódu 4.11: Příkaz pro přidání Service Workera (zdroj: interní zdroj)

```
ng add @angular/pwa
```

Nejdůležitějším krokem, který příkaz vykonal, je přidání Service Workera. Service Worker může zachytit požadavky na server a vrátit výsledky v mezipaměti, kdykoli je to možné. To znamená, že aplikace by měla fungovat i když jsme offline.

Přidáním podpory PWA se také vytvořily ikony aplikací různých velikostí, které nahradíme logem společnosti. Nakonec byl do souboru src/manifest.json přidán manifest webové aplikace. Manifest poskytuje prohlížeči informace, které potřebuje k instalaci aplikaci lokálně na zařízení uživatele.

Dále se vygeneroval soubor `ngsw-config.json`, který je zodpovědný za Service Workery.

Ukázka kódu 4.12: Soubor `ngsw-config.json` (zdroj: interní zdroj)

ngsw-config.json

```
{
  "$schema": "./node_modules/@angular/service-
worker/config/schema.json",
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",
          "/manifest.webmanifest",
          "/*.css",
          "/*.js"
        ]
      }
    }, {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/assets/**",

"/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
        ]
      }
    }
  ]
}
```

4.8.2 Implementace notifikací

Stejně jako v jiných případech, by aplikace měla působit zdvořile a přátelsky. Je povinností požádat koncového uživatele o povolení oznamování skrze push notifikace.

Ukázka kódu 4.13: Metoda pro získaní oprávnění k odesílání notifikací (zdroj: interní zdroj)

```
await Notification.requestPermission();
```

Pokud si uživatel přidá náš CRM systém na plochu a souhlasí s odesíláním notifikací, budeme je pro něj v kódu tvořit skrze Notification API. Na ukázce kódu 4.14

si ukážeme implementaci základní notifikace, která je odeslána uživateli po vytvoření investice.

Ukázka kódu 4.14: Základní notifikace (zdroj: interní zdroj)

```
async function showNotification() {
 const result = await Notification.requestPermission();
      if (result === 'granted') {
      const noti = new Notification('Nová investice', {
                  body: 'Byla vytvořena nová investice.',
                  icon: 'investment.png'
            });
            noti.onclick = () => swal('success');
      }
}
showNotification();
```

Pokud máme povolení zasílat oznámení, tvoření notifikací je relativně jednoduché a využívá se při něm konstruktoru třídy `Notification`. Konstruktoru můžeme přidat parametry, jako jsou název nebo konfigurační objekt, obsahující možnosti pro vytvoření oznámení.

Přidání PWA funkcí do exitující aplikace není zas tak složitým procesem, jak by se mohlo zdát. Nejtěžší na této implementaci je správné ukládání odpovědí ze serverového API do mezipaměti. Tato funkcionalita se bude vylepšovat, až v budoucích verzích CRM systému. Pro uspokojení požadavku bude prozatím stačit možnost přidání aplikace na plochu mobilního zařízení a základní notifikace.

5 Závěr

Po nasazení nové verze CRM systému jsme se setkali převážně s kladnými ohlasy od uživatelů. Klienti chválili jednoduchost a obchodní zástupci byli potěšeni zejména razantním zrychlením nejpoužívanějších funkcí. Zvyk je však železná košile a někteří věkově starší klienti společnosti se hůře adaptovali na nový minimalistický design. První týdny od spuštění řešili pracovníci supportu hlavně poradenství klientům, kteří nemohli najít některé interaktivní prvky aplikace.

Je pozitivní, že se po nasazení nevyskytovalo moc chyb a jednalo se spíše o občasný jev spojený s migrací původních dat do nové databáze. Vzhledem ke komplexitě celé aplikace a relativně krátkému vývojovému cyklu, se jedná o pozitivní výsledek. Na vyskytnuté chyby bylo reagováno velmi pružně, efektivně a byly vyřešeny v co nejkratším možném termínu. Jedinou skutečností, která kazí celkový dojem z jinak dobře zvládnutého vývoje je, že termín vydání nové verze systému byl z důvodu technických problémů posunut o třicet kalendářních dní. Problémy se týkaly hlavně migrace dat z původní databáze do databáze nové, jelikož se značně změnil i datový model a databázové schéma.

Jelikož cíl práce byl splněn a nový systém je rychlejší, snadno škálovatelný a modularizovaný, tak nastala doba, kdy klíčoví uživatelé systému navrhovali implementace dalších funkcí, které by usnadnili vykonávání procesů spojených s chodem společnosti. Za tímto účelem zavedl management společnosti tzv. technické porady, které se od doby nasazení nového systému konají v pravidelných intervalech. Porad se zúčastňují nejvýše postaveni obchodní zástupci, management společnosti a jeden člen IT oddělení. Obchodní zástupci mají prostor navrhnout úpravy nebo implementace nových funkcí do systému a probrat se zástupcem IT oddělení, zda jsou úpravy technicky realizovatelné. Management je na poradě od toho, aby posoudil, zda navrhnuté úpravy korespondují se strategickými cíli společnosti.

Ačkoli byl cíl diplomové práce splněn v plném rozsahu, práce na CRM systému zdaleka nekončí. Aplikace bude nadále procházet postupnými úpravami, neboť její fungování je klíčové pro funkci hlavních, ale i některých podpůrných procesů společnosti. V současné době se pracuje na dalších vývojových sprintech, které mají do systému

přinést možnost správy plateb ze zahraničí, další jazykové mutace a podporu automatického testování zdrojového kódu.

Framework Angular je velmi progresivní a společnost Google přináší vývojářům novou verzi každých šest měsíců. Do budoucna si kladu za cíl udržovat frontend nového CRM systému vždy na aktuální stabilní verzi Angularu, i přes pravděpodobnou nutnost refaktoringu starších funkcí.

Seznam použité literatury

Odborná kniha

ANDERSON, David J. *Kanban: Successful Evolutionary Change for Your Technology Business.* NZ: Blue Hole Press, 2010. ISBN 978-0984521401.

BEAN, Martin. *Laravel 5 Essentials*. Birmingham: Pack Publishing, 2015. ISBN 978-1785283017.

BEVACQUA, Nicolas. *Practical modern Javascript*. Sebastopol, CA: O'Reilly Media, 2017 ISBN 9781491943533id.

FREEMAN, Adam. *Essential TypeScript: From Beginner to Pro*. New York: Apress, 2019. ISBN 978-1-4842-4979-6.

FREEMAN, Adam. *Pro Angular 6 Third Edition.* New York: Apress, 2018. ISBN 978-1484236482.

LEHTINEN, Jarmo. *Aktivní CRM: řízení vztahu se zákazníky*. Přeložil Alena SVOZILOVÁ. Praha: Grada Publishing, 2007. ISBN 978-80-247-1814-9.

MIKOWSKI, Michael S. a Josh C. POWELL. *Single page web applications: JavaScript end-to-end.* Shelter Island, NY: Manning, 2014. ISBN 9781617290756.

MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.

ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. Brno: Computer Press, 2014. ISBN 978-80-251-4194-6.

Elektronické dokumenty a ostatní

COPES, Flavio. *The React Handbook*. In: FreeCodeChamp [online]. January 2019 [cit. 2020-03-29]. Dostupné z: https://www.freecodecamp.org/news/the-react-handbook-b71c27b0a795/

COPES, Flavio. *The Vue Handbook: a thorough introduction to Vue.js.* In: FreeCodeChamp [online]. July 2018 [cit. 2020-03-29]. Dostupné z: https://www.freecodecamp.org/news/the-vue-handbook-a-thorough-introduction-to-vue-js-1e86835d8446/

DONALD, Claire, *An Introduction To: Agile Software Development* [online]. In: *Medium*. 30 November 2017 [cit. 2020-03-29]. Dostupné z: https://medium.com/shecancode/an-introduction-to-agile-software-development-914339dcec66

Agile 101, *Agile Alliance* [online]. [cit. 2020-03-29]. Dostupné z: https://www.agilealliance.org/agile101/

Introduction to modules, *Angular.io* [online]. [cit. 2020-03-29]. Dostupné z: https://angular.io/guide/architecture

Introduction to components, *Angular.io* [online]. [cit. 2020-03-29]. Dostupné z: https://angular.io/guide/architecture-components

Introduction to services and dependency injection, *Angular.io* [online]. [cit. 2020-03-29]. Dostupné z: https://angular.io/guide/architecture-services

Observables & RxJS, *Angular.io* [online]. [cit. 2020-03-29]. Dostupné z: https://angular.io/guide/observables

What is Kanban?, *Digité* [online]. [cit. 2020-03-29]. Dostupné z: https://www.digite.com/kanban/what-is-kanban/

Seznam zkratek

AOT	Ahead of Time
API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
CRM	Customer Relationship Management
CRON	Commands Run Over Night
CSS	Cascade Style Sheets
CSV	Comma Separated Values
ČSOB	Československá obchodní banka
DI	Dependency Injection
DOM	Document Object Model
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identification
IT	Information Technology
JIT	Just in Time
JSON	JavaScript Object Notation
JSX	JavaScript XML
KB	Kilobyte
KPI	Key Performance Index
MB	Megabyte
MIT	Massachusetts Institute of Technology
MPA	Multi Page Application
MVC	Model View Controller

MVVM	Model View ViewModel
OVH	Online Virtual Hosting
PDF	Portable Document Format
PHP	Hypertext Preprocessor
PWA	Progressive Web Application
QR	Quick Response
SASS	Syntactically Awesome Style Sheets
SEO	Search Engine Optimalization
SLA	Service Level Agreement
SPA	Single Page Application
SPAYD	Short Payment Description
SW	Software
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VPS	Virtual Private Server
XP	Extreme Programming

Prohlašuji, že

- jsem byl(a) seznámen(a) s tím, že na mou diplomovou (bakalářskou) práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- 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 Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou (bakalářskou) práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová (bakalářská) práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO. Souhlasím s tím, že bibliografické údaje o diplomové (bakalářské) práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou (bakalářskou) práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 21.4.2020

Pešat

.....

Bc. Ondřej Pešat

Seznam příloh

Příloha 1 : Grafické znázornění modulů

Příloha 1: Grafické znázornění modulů