FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Zákaznický portál firmy ÚVT Internet, s.r.o.
Student:	Bc. Martin Horejš
Vedoucí:	Ing. Pavel Richter
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Seznamte se se stávajícím řešením a následně analyzujte, navrhněte, diskutujte, zvolte technologie a implementujte nové řešení. Výslednou aplikaci integrujte do prostředí firmy. Řešení bude splňovat níže uvedené požadavky:
1) Zobrazení profilu zákazníka.
2) Zákazník bude mít možnost odeslat požadavek na změnu osobních údajů.
3) Zobrazení aktivních služeb zákazníka a informace o jejich stavu.
4) Zobrazení vyúčtování/faktur jednotlivých služeb.
5) Výpis telefonních hovorů (v případě, že využívá této služby).
6) Možnost aktivace dalších služeb pro zákazníka.
7) Možnost zobrazení QR kódu pro mobilní platbu nezaplacených faktur.
8) Zobrazení přehledu informativních a provozních zpráv zaslaných zákazníkovi.
9) Multiplatformní přístup - řešení bude dostupné online jako webová aplikace.
10) Cizojazyčná lokalizace - aplikace bude podporovat lokalizace pro další jazyky.
11) Zabezpečení aplikace proti neoprávněnému vniknutí.
12) Zabezpečení aplikace proti úniku osobních údajů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 27. února 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Zákaznický portál firmy ÚVT Internet s.r.o.

Bc. Martin Horejš

Vedoucí práce: Ing. Pavel Richter

2. ledna 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu Pavlu Richterovi za vedení a firmě ÚVT Internet s.r.o. za poskytnutí příležitosti pro vznik této práce. Dále děkuji Tomáši Nováčkovi za jazykovou korekturu textu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 2. ledna 2019 .....

Odkaz na tuto práci

Horejš, Martin. *Zákaznický portál firmy ÚVT Internet s.r.o.*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Cílem této diplomové práce je vytvořit webovou aplikaci pro zákazníky firmy internetového poskytovatele ÚVT Internet s.r.o. Text této práce postupně rozebírá většinu částí softwarového vývoje od analýzy až po nasazení výsledného produktu. Výstupem práce je plně funkční webová aplikace spuštěna v reálném provozu, která zpřístupňuje zákazníkům správu odebíraných služeb a další související informace.

Klíčová slova webová aplikace, integrace systémů, Vue.js, Vuetify.js, Sails.js, Node.js, PHP

Abstract

The primary goal of this master's thesis is to create a web application for customers of the company ÚVT Internet s.r.o. The text of this thesis considers the whole process of software development beginning with an analysis and ending with the deployment of the final product. The outcome of this work is a fully functional web application deployed to the production environment that enables management of customer services and other personal information.

Keywords web application, system integration, Vue.js, Vuetify.js, Sails.js, Node.js, PHP

Obsah

Seznam obrázků

Seznam tabulek

Ukázky kódu

Úvod

V dnešní době je vyvíjen konkurenční tlak na firmy, aby ulehčily a zpříjemnily svůj vztah se zákazníkem pomocí různých informačních systémů. U společností zabývajících se moderními technologiemi je tento tlak ještě větší a jejich absence může znamenat zásadní konkurenční nevýhodu.

Společnost ÚVT Internet s.r.o., působící v Praze a Středočeském kraji, je poskytovatelem internetového připojení a dalších služeb, jako jsou hlasové služby či internetová televize. ÚVT Internet je zároveň zadavatelem této práce, protože potřebuje vytvořit nový informační systém pro své zákazníky. Tento systém by měl zákazníkům ulehčit úkony, které šlo doposud řešit pouze telefonicky či osobně. Zadáním je tedy vytvořit aplikaci pro zákazníky, která se funkcionalitou bude podobat například aplikacím hlavních mobilních operátorů v České republice, jako jsou Můj Vodafone, Moje O2 a Můj T-Mobile.

Cíle práce

Zadáním této práce je seznámení se se stávajícím řešením, analýza, návrh, diskuze použitých technologií a implementace nového řešení. Toto řešení musí být zároveň integrováno do softwarové infrastruktury zadavatele a také je nutné, aby splňovalo všechny požadavky v seznamu níže:

1. Zobrazení profilu zákazníka.
2. Zákazník bude mít možnost odeslat požadavek na změnu osobních údajů.
3. Zobrazení aktivních služeb zákazníka a informace o jejich stavu.
4. Zobrazení vyúčtování/faktur za jednotlivé služby.
5. Výpis telefonních hovorů (v případě, že zákazník využívá této služby).
6. Možnost aktivace dalších služeb pro zákazníka.

7. Možnost zobrazení QR kódu pro mobilní platbu nezaplacených faktur.
8. Zobrazení přehledu informativních a provozních zpráv zaslaných zákazníkovi.
9. Multiplatformní přístup – řešení bude dostupné online jako webová aplikace.
10. Cizojazyčná lokalizace – aplikace bude podporovat lokalizace do dalších jazyků.
11. Zabezpečení aplikace proti neoprávněnému vniknutí.
12. Zabezpečení aplikace proti úniku osobních údajů.

Tyto požadavky tvoří základ definice všech softwarových požadavků na nové řešení, které jsou definovány v kapitole Analýza. Aplikace bude dle požadavků ve webovém provedení a bude přístupná z více druhů zařízení, například z mobilního telefonu. Mimo dalších požadavků je nutné aplikaci zabezpečit proti neoprávněnému přístupu a úniku informací.

Text této práce popisuje v pěti kapitolách celý softwarový proces vývoje zadaného projektu. Začíná analýzou, která se zabývá stávajícím řešením, definicí požadavků, případů užití a především také doménovým modelem. Další kapitola návrhu aplikace diskutuje možnou podobu nového řešení, jehož implementace je popsána v navazující kapitole. Předposlední kapitola popisuje testování výsledné implementace, jejíž proces nasazení do reálného provozu je diskutován v poslední kapitole Nasazení aplikace.

Kapitola 1

Analýza

Analýza představuje rešerši problematiky a výstupy konzultací se zadavatelem, ÚVT Internet s.r.o. Po první konzultaci bylo definováno zadání této práce, avšak definované požadavky byly v průběhu vypracovávání ještě pozměněny, což je častou praxí softwarového procesu. Například „Sekce podpory zákazníka“ byla přesunuta na veřejné stránky společnosti *www.uvtnet.cz* a také byly přidány požadavky **FP.5**, **FP.6** a **FP.9**.

Kapitola začíná popisem stávajícího řešení zákaznického portálu společnosti ÚVT Internet s.r.o. Následuje definice konečných požadavků na nové řešení a popis případů užití. V další sekci je diskutováno pokrytí požadavků namodelovanými případy užití. Poslední část této kapitoly diskutuje doménový model aplikace.

1.1 Stávající řešení

Stávající řešení funguje jako samostatná webová aplikace, napsaná, stejně jako zadavatelův CRM systém (Opentis), v jazyce PHP. Funkcionalita stávající aplikace je velice strohá a funguje pouze jako provizorní řešení s původní myšlenkou dalšího vývoje a rozšíření.

1.1.1 Architektura aplikace

Aplikace je napsaná ve skriptovacím jazyce PHP a nevyužívá žádného existujícího frameworku tohoto jazyka. Architektura je totožná s tou ze systému Opentis firmy ÚVT Internet s.r.o., který vznikl jako rozšíření diplomové práce Ing. Pavla Richtera[1], vedoucího této práce.

I přes dobré rozdělení kódu aplikace se v ní vyskytují rysy, které by mohly při dalším vývoji nebo údržbě přinést problémy. Jedním z takových rysů jsou nedostatečně oddělené vrstvy aplikace, kde je ve skriptech implementace promíchána logika datové, aplikační a zobrazovací vrstvy.

Dalším problémem může být použití původního zastaralého PHP rozšíření MySQL[2] pro práci s databází, které již není oficiálně podporováno. Toto rozšíření rovněž neobsahuje pokročilou funkcionalitu pro zabezpečení databáze, například *SQL prepared statements*, které velmi dobře mitiguje zranitelnost proti útokům typu *SQL injection*[1].

Toto rozšíření PHP je navíc použité přímo v implementaci pro manipulaci doménového modelu místo pokročilejších technik, jako je například ORM neboli *objektově relační mapování*, což je technika napojení databáze do kódu aplikace. Tato technika zapouzdřuje volání API[2] databáze, čímž zamezuje vysokému počtu nutných úprav kódu při změně schématu databáze a umožňuje lepší přenositelnost mezi více databázovými systémy.

Nevyužití žádného PHP frameworku je další problém, který by v budoucím vývoji značně zhoršil udržitelnost aplikace. Frameworky se totiž obecně snaží o dobrý návrh architektury a zapouzdření jejich logiky, což s sebou přináší výhody snazší aktualizace na modernější algoritmy a technologie.

1.1.1.1 Integrace s CRM systémem Opentis

Systém Opentis je CRM systém napsaný v jazyce PHP a spravuje většinu zákaznických dat. Tato data je nutné sdílet s zákaznickým portálem, a je tedy nutná integrace těchto dvou systémů.

Integrace je navržena tak, aby při událostech měnících data v systému Opentis byly tyto změny propsány přes API do samostatné databáze stávající aplikace. Dochází tak k replikaci velké části dat z databáze systému Opentis do databáze zákaznického portálu.

Tento způsob integrace je bohužel velmi znevýhodněn stávající architekturou systému Opentis. Tato architektura nenabízí snadnou cestu, jak identifikovat konkrétní logiku systému Opentis řešící události, které bezprostředně modifikují data nutná pro zákaznický portál.

Je tedy velmi náročné přidat do systému Opentis kód pro replikaci dat do databáze stávající aplikace zákaznického portálu v případě, že nastane událost modifikující tato data. Další vývoj tímto směrem by byl velmi časově náročný a navíc je zde vyšší pravděpodobnost chybné implementace replikace, protože při změně v doménovém modelu systému Opentis je nutné tuto změnu promítnout také do zákaznického portálu. Zapomenutá změna doménového modelu nebo lehce odlišná implementace logiky by mohla snadno porušit integritu dat mezi oběma databázemi těchto systémů.

[1] SQL injection je technika útoku na úrovni databáze, kdy se pomocí nevalidovaných vstupů do aplikace útočník snaží zmanipulovat samotný SQL dotaz, a získat tak jinak nepřístupná data.

[2] API, neboli Application Programming Interface, je rozhraní pro programování, které definuje sadu procedur, funkcí a tříd využitelných při implementaci softwaru.[3]

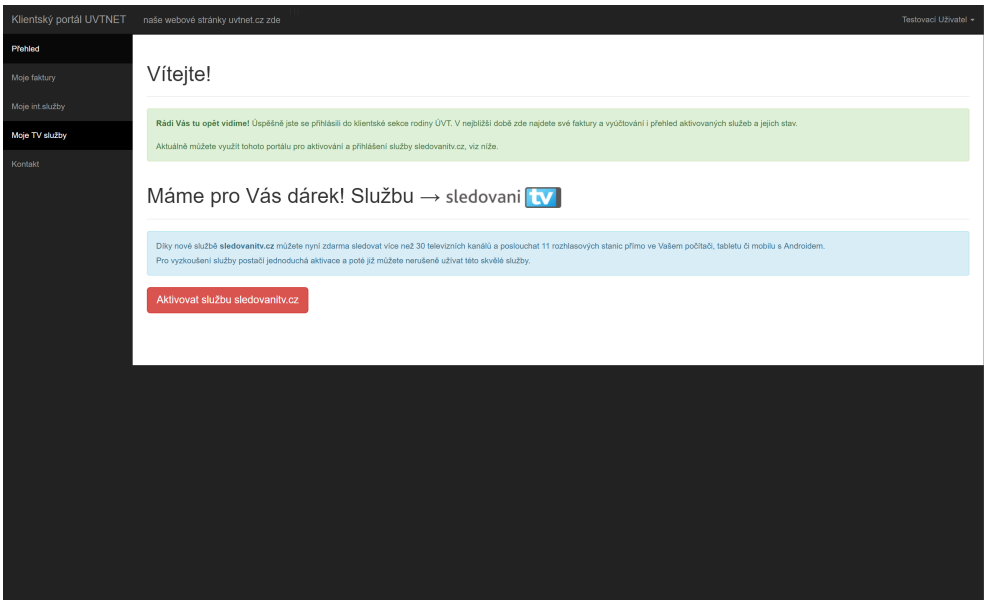
1.1.2 Funkcionalita

Stávající funkcionalita zákaznického portálu umožňuje přihlášení do aplikace, zobrazení profilu uživatele a možnost aktivování televizní služby *Sledovanitv*. Další rozšíření funkcionality bylo naplánováno, ale pro tuto činnost se nenašel dostatek zdrojů. Požadavky v zadání této práce vychází tedy především z těchto naplánovaných rozšíření stávajícího řešení.

Většina původní funkcionality je vidět na obrázku 1.1, který zachycuje výchozí obrazovku aplikace. Ostatní sekce aplikace, až na *Kontakt*, nejsou implementované a obsahují informaci, že jsou v přípravě.

1.1.3 Vzhled aplikace

HTML kód aplikace je generovaný PHP skripty a vzhled aplikace je řešen pomocí CSS knihovny Bootstrap. Je zde použito výchozí grafické téma této knihovny, které poskytuje velmi malou odlišitelnost od většiny aplikací používajících toto téma.



Obrázek 1.1: Výchozí stránka stávajícího řešení

1.2 Definice požadavků

Během konzultací se zadavatelem byly definovány požadavky na novou aplikaci. Od nich se přímo odvíjí rozsah aplikace a také cena výsledného produktu. Aktuální fáze vývoje produktu lze měřit množstvím implementovaných požadavků. To umožňuje zadavateli kontrolovat, v jakém stavu se vyvíjený projekt

nachází a jestli řešitel úspěšně postupuje k dokončení vývoje. Po druhé konzultaci se zadavatelem bylo nadefinováno 9 funkčních a 5 nefunkčních požadavků.

1.2.1 Funkční požadavky

Funkční požadavky stručně řečeno definují, co aplikace bude umožňovat. Jedná se tedy o seznam funkcionalit, které musí být v aplikaci naimplementované.

FP.1 Zobrazení profilu uživatele.

FP.2 Možnost odeslání požadavku na změnu osobních údajů zákazníka.

FP.3 Zobrazení služeb, které zákazník odebírá, a informace o jejich stavu.

FP.4 Zobrazení vyúčtování/faktur za jednotlivé služby.

FP.5 Zobrazení fakturačních údajů.

FP.6 Možnost zobrazení QR kódu pro mobilní platbu nezaplacených faktur.

FP.7 Výpis seznamů telefonních hovorů (v případě, že zákazník využívá telefonních služeb).

FP.8 Možnost aktivace dalších služeb pro zákazníka.

- Internetová televizní služba sledovani.tv.

FP.9 Zobrazení informačních zpráv zaslaných zákazníkovi.

1.2.2 Nefunkční požadavky

Nefunkční požadavky se zabývají nefunkčními atributy aplikace, jako jsou například bezpečnost, přístupnost či udržitelnost.

NP.1 Multiplatformní přístup – aplikace musí dobře fungovat na většině zařízení.

NP.2 Zabezpečení aplikace proti neoprávněnému vniknutí.

NP.3 Zabezpečení proti úniku osobních údajů.

NP.4 Integrace systému se stávajícím CRM systémem Opentis.

NP.5 Monitorování a pravidelné zálohování systému.

1.3 Případy užití

Případy užití se modelují jako detailní rozšíření funkčních požadavků a definují, jak bude aplikace používaná různými účastníky. Vzhledem k tomu, že je aplikace tvořená jenom pro zákazníky zadavatele, účastníkem všech případů užití je tedy pouze **Zákazník**.

Níže je vypsán seznam nadefinovaných případů užití.

UC.1 Zobrazení profilu.

UC.2 Vytvoření požadavku na změnu uživatelských údajů.

UC.3 Zobrazení požadavků.

UC.4 Vyhledání požadavku.

UC.5 Zrušení požadavku.

UC.6 Změna hesla.

UC.7 Zobrazení zpráv.

UC.8 Zobrazení vyúčtování.

UC.9 Vyhledání faktury.

UC.10 Stažení faktury.

UC.11 Zobrazení QR pro platbu.

UC.12 Zobrazení služeb.

UC.13 Aktivace televizní služby.

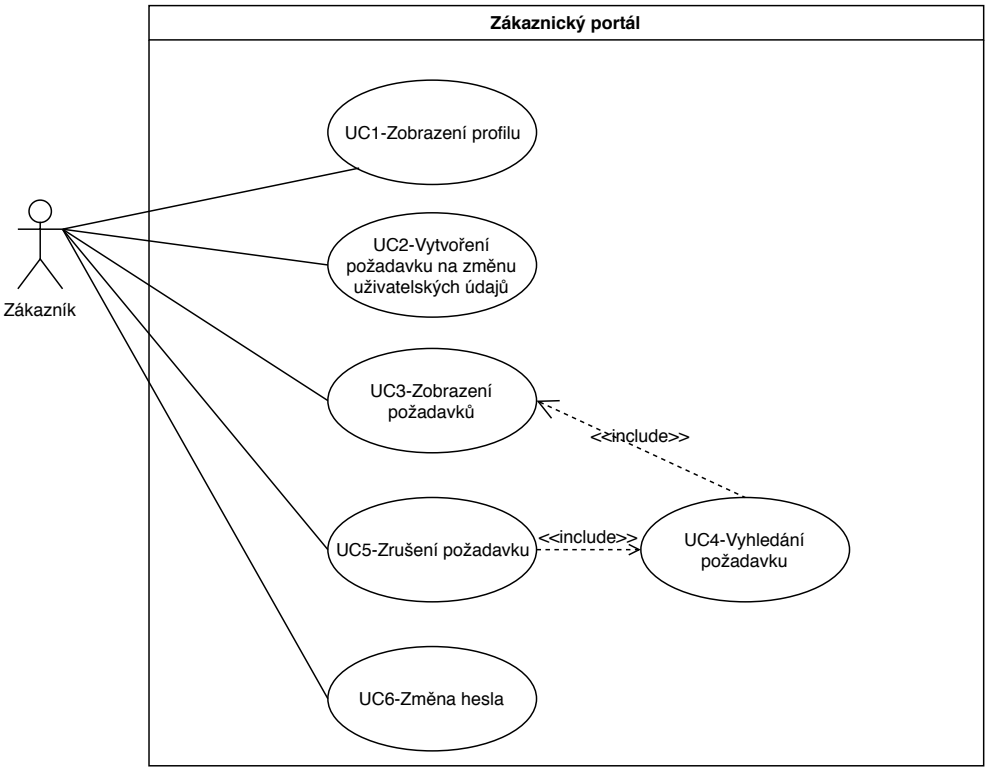
UC.14 Stažení výpisu hovorného.

UC.15 Zobrazení detailu služby.

UC.16 Změna jazyka aplikace.

UC.17 Resetování hesla.

Následující sekce rozdělují případy užití do kategorií a obsahují grafické znázornění těchto kategorií případů užití. Některé složitější případy také obsahují scénář průběhu, který eliminuje možné nejasnosti.



Obrázek 1.2: Případy užití, kategorie: Správa uživatele

1.3.1 Správa uživatele

1.3.1.1 UC.2 – Vytvoření požadavku na změnu uživatelských údajů

1. Případ užití začíná, když má zákazník zobrazený svůj profil a chce změnit/přidat údaje o sobě či další kontaktní osobě.

2. Zákazník spustí akci pro změnu/přidání údajů.

3. Aplikace zobrazí formulář s předvyplněnými poli v případně editace a prázdnými poli v případě přidání údajů.

4. Zákazník vyplní povinná pole a odešle formulář.

5. Aplikace upozorní uživatele, zda se podařilo vytvořit požadavek.

1.3.1.2 UC.5 – Zrušení požadavku

1. Případ užití začíná, když má zákazník zobrazeny své požadavky na změny.

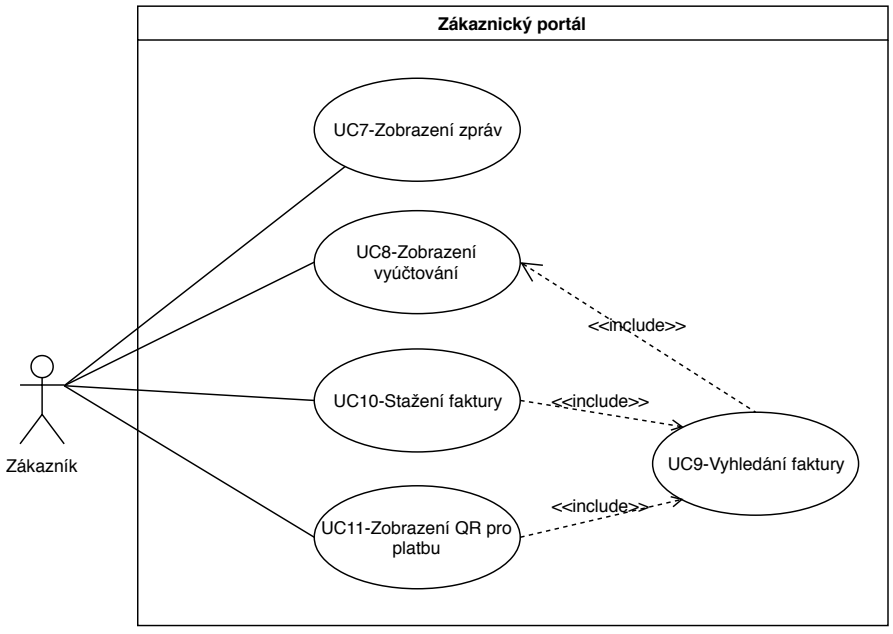
- V alternativním případě může zákazník začít případem užití **UC.3 – Zobrazení požadavků**.

2. Zákazník vykoná **UC4 – Vyhledání požadavku**.
3. Zákazník spustí akci zrušení požadavku.
4. Aplikace si vyžádá potvrzení o zrušení požadavku.
5. Aplikace upozorní uživatele, zda zrušení požadavku proběhlo úspěšně.

1.3.1.3 UC.6 – Změna hesla

1. Případ užití začíná, když má zákazník zobrazený svůj profil a chce změnit své heslo do aplikace.
2. Zákazník spustí akci změny hesla.
3. Aplikace zobrazí formulář pro změnu hesla.
4. Zákazník vyplní své staré heslo, nové heslo a znovu nové heslo pro kontrolu. Poté zákazník formulář odešle ke zpracování.
5. Aplikace upozorní uživatele, zda změna hesla proběhla v pořádku.

1.3.2 Správa vyúčtování

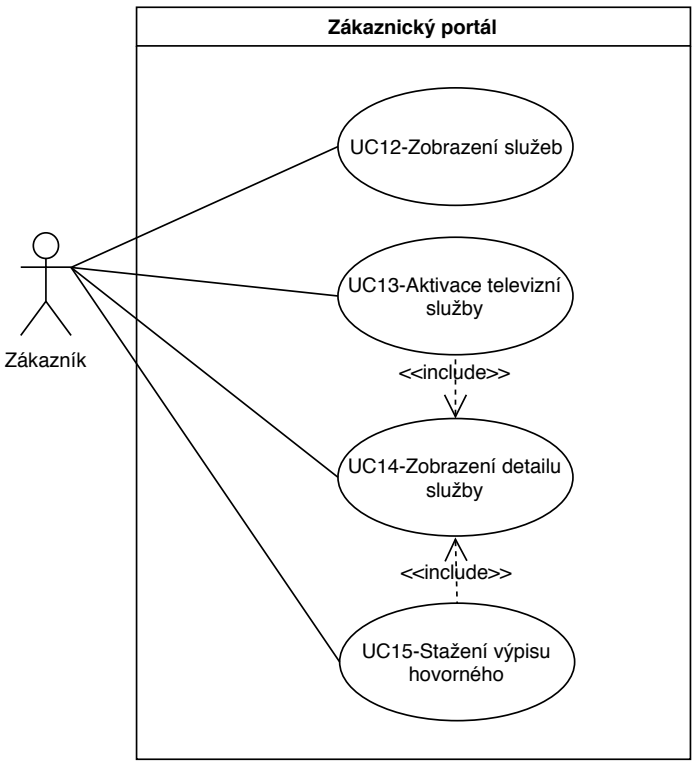


Obrázek 1.3: Případy užití, kategorie: Správa vyúčtování

1.3.2.1 UC.10 – Stažení faktury

1. Případ užití začíná, když má zákazník zobrazeny svoje faktury z vyúčtování.
    - V alternativním případě může zákazník začít případem užití **UC.8 – Zobrazení vyúčtování**.
2. Zákazník vykoná **UC.9 – Vyhledání faktury**.
3. Zákazník spustí akci stažení faktury.
4. Z aplikace se stáhne vybraná faktura.

1.3.3 Správa služeb



Obrázek 1.4: Případy užití, kategorie: Správa služeb

1.3.3.1 UC.13 – Aktivace televizní služby

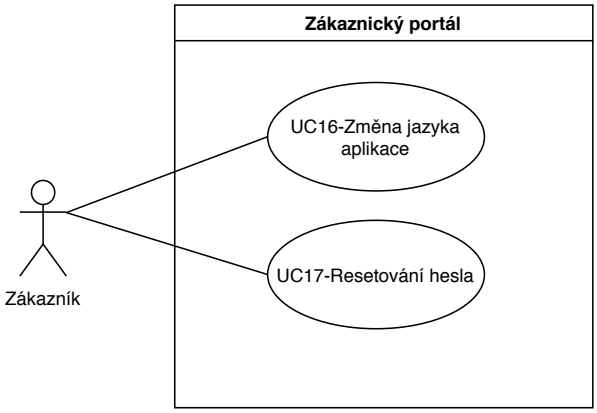
1. Případ užití začíná, když má zákazník zobrazený detail televizní služby.
    - V alternativním případě může zákazník začít případem užití **UC.14 - Zobrazení detailu služby**.

2. V případě, že zákazník tuto službu nemá aktivovanou, spustí její aktivaci.
3. Aplikace zpracuje požadavek na aktivaci.
4. Aplikace informuje uživatele, zda aktivace proběhla v pořádku.

1.3.3.2 UC.15 – Stažení výpisu hovorného

1. Případ užití začíná, když má zákazník zobrazený detail hlasové služby.
   - V alternativním případě může zákazník začít případem užití **UC.14 – Zobrazení detailu služby**.
2. Zákazník vybere období, za které chce stáhnout výpis hovorného, a spustí akci stažení.
3. Z aplikace se stáhne vybraný výpis.

1.3.4 Ostatní případy užití



Obrázek 1.5: Případy užití, kategorie: Ostatní případy užití

1.3.4.1 UC.17 – Resetování hesla

1. Případ užití začíná, když zákazník potřebuje resetovat svoje přihlašovací heslo do aplikace, aniž by byl přihlášený (tj. zákazník zapomněl své přihlašovací heslo).
2. Zákazník místo přihlášení spustí akci resetování hesla.
3. Aplikace zobrazí formulář pro resetování hesla.
4. Zákazník vyplní údaje pro resetování hesla.

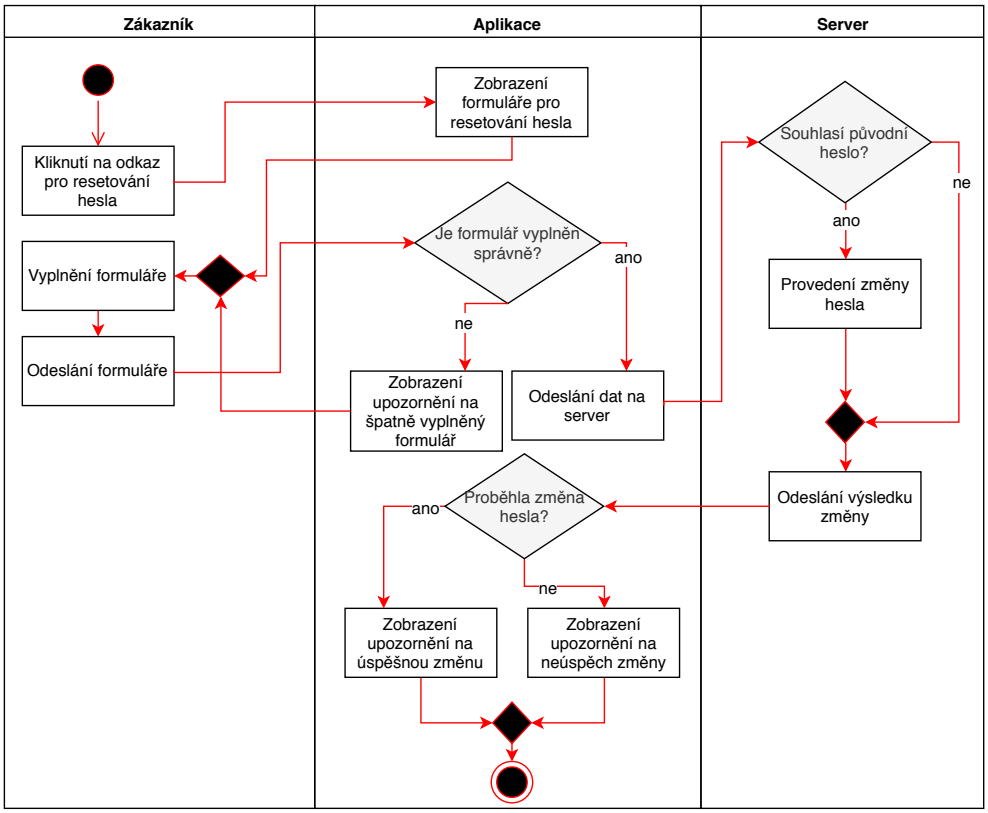
5. Aplikace v případě správných údajů odešle email s URL pro resetování hesla.

    - V alternativním případě se zákazníkovi zobrazí upozornění na špatně vyplněný formulář.

6. Zákazník toto URL otevře, dvakrát zadá svoje nové heslo a odešle formulář.

7. Aplikace požadavek zpracuje a informuje zákazníka, zda byl tento požadavek úspěšně vykonán.

Případ užití pro resetování zapomenutého hesla byl pro jeho složitost namodelován také jako diagram aktivit, který je zobrazen na obrázku 1.6. Diagram je rozdělen do tří částí, které reprezentují zóny odpovědnosti zákazníka, aplikace a serveru.



Obrázek 1.6: Aktivity diagram případu užití: UC.17 – Resetování hesla

1.4 Pokrytí funkčních požadavků případy užití

Aby bylo zaručené, že aplikace bude splňovat funkcionalitu definovanou zadavatelem, musí namodelované případy užití pokrýt všechny funkční požadavky definované v sekci 1.2.1. V následující tabulce 1.1 je v každém sloupci uveden alespoň jeden případ užití, tedy všechny funkční požadavky jsou opravdu pokryty.

Případy užití \ Funkční požadavky	FP.1	FP.2	FP.3	FP.4	FP.5	FP.6	FP.7	FP.8	FP.9
UC.1	✓				✓				
UC.2		✓							
UC.3						✓			
UC.4									
UC.5									
UC.6									
UC.7									✓
UC.8				✓	✓				
UC.9				✓					
UC.10				✓					
UC.11						✓			
UC.12			✓						
UC.13			✓					✓	
UC.14			✓						
UC.15							✓		
UC.16									
UC.17									

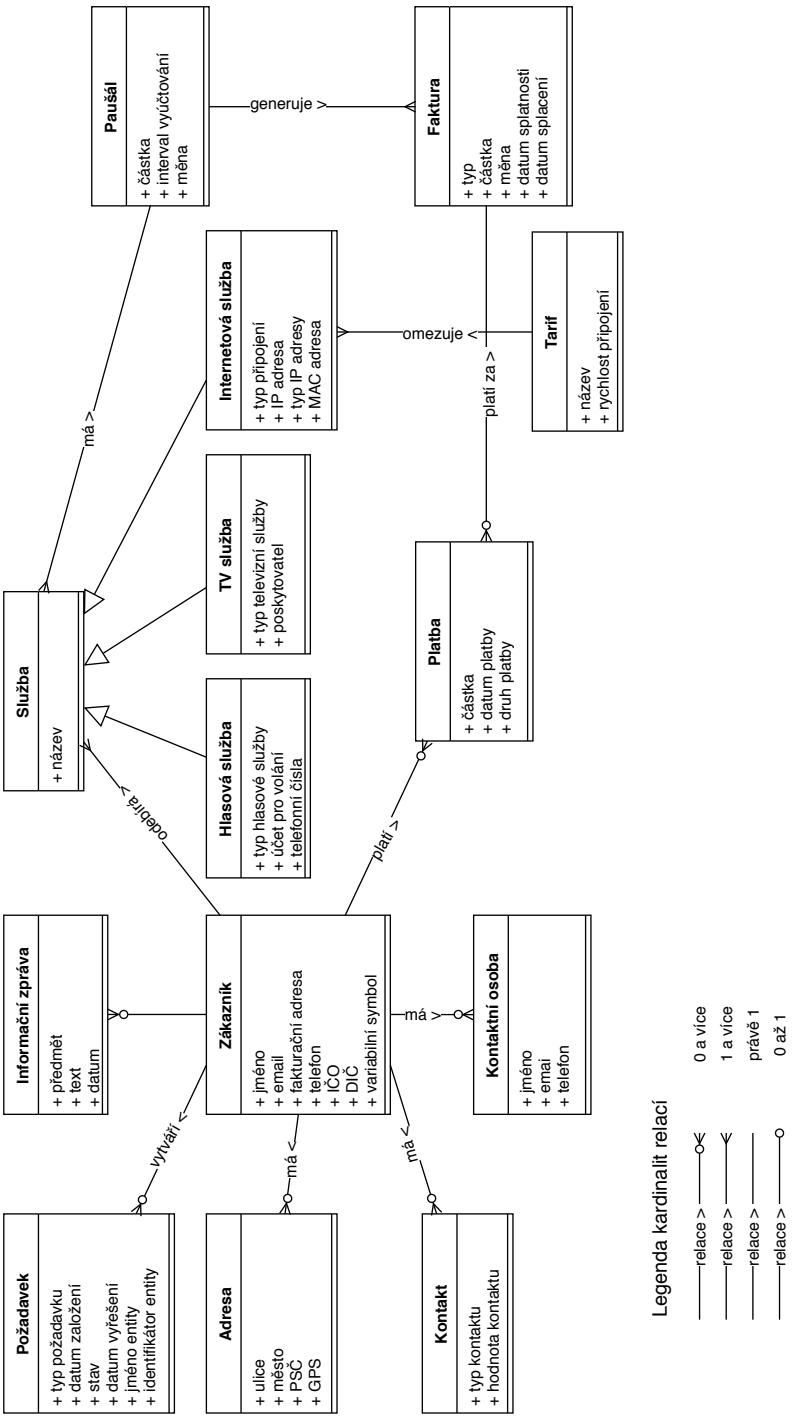
Tabulka 1.1: Pokrytí funkčních požadavků případy užití

1.5 Doménový model

Doménový model slouží především jako nástroj pro lepší pochopení aplikace a jejích funkcionalit. Zachycuje entity z reálného světa, které se vyskytují v aplikaci. Doménový model zobrazuje také atributy jednotlivých entit a relace existující mezi nimi. Relace jsou zobrazeny čarami mezi právě dvěma entitami a jejich směr je určen šipkou za jejím názvem. Kardinalita relací je znázorněna různým zakončením čar, která jsou vysvětlena v legendě na obrázku doménového modelu 1.7.

Jelikož jsou data ze systému Opentis sdílená s novou aplikací, tak její doménový model je v podstatě podmnožinou doménového modelu systému

Opentis. Opentis je velmi rozsáhlý systém a velikost této podmnožiny je co do počtu entit zhruba 3%.



Obrázek 1.7: Doménový model aplikace

1.5.1 Zákazník

Entita *Zákazník* je jedním ze základních stavebních kamenů doménového modelu aplikace. Zachycuje informace o zákazníkovi jako je například jeho jméno, telefon, IČO nebo variabilní symbol. *Zákazník* je v relaci s entitou *Služba*, což zachycuje stav, kdy *zákazník* odebírá jednu nebo více služeb. Další jeho důležitou relací je relace s *požadavkem*, kdy zákazník vytváří různé typy požadavků, jako může být změna osobních údajů, přidání *kontaktních osob*, změna tarifu *internetové služby* atp.

1.5.2 Služba

Entita *Služba* je další důležitou entitou doménového modelu. Služba musí být jednou ze tří typů, a to *hlasová služba*, *TV služba*, nebo *internetová služba*. Atributy *služby* se výrazně liší typ od typu, ale všechny tři typy jsou v relaci s *paušálem*, kdy je jedna či více služeb zastřešeno *paušálem* pro vyúčtování *služeb*.

1.5.3 Paušál

Entita *Paušál* obsahuje jednu či více *služeb*, pro které řeší pravidelné vyúčtování. Její atributy obsahují informace o intervalu vyúčtování, peněžní částce a měně, které jsou důležité pro generování *faktur*, což je znázorněno relací mezi těmito dvěma entitami.

1.5.4 Faktura

Entita *Faktura* reprezentuje daňový doklad o vyúčtování odebíraných *služeb zákazníka* za dané časové období. *Faktura* je v pravidelném časovém intervalu generována entitou *Paušál*.

1.5.5 Platba

Entita *Platba* představuje informaci o platbě *zákazníka* za odebírané *služby*. *Platba* je tedy v relaci s fakturou i *zákazníkem*, kdy zákazník může zadat nula a více plateb a *faktura* může být zaplacená nula a více *platbami*.

1.5.6 Požadavek

Entita *Požadavek* představuje požadavek na změnu nebo přidání další entity doménového modelu. Přidání respektive změny se můžou týkat například *kontaktů*, *kontaktních osob* nebo fakturačních údajů zákazníka. Plánované jsou taky změny *tarifů* daných *služeb*.

1.5.7 Informační zpráva

Entita *Informační zpráva* představuje zaslanou informační zprávu zákazníkovi. Takováto zpráva se může týkat například plánovaného výpadku služeb, speciální nabídky nebo dalších informací. Všechny informační zprávy jsou zasílány elektronickou poštou na fakturační email *zákazníka*.

Kapitola 2

Návrh

Tato kapitola popisuje návrh nového řešení na základě předešlé analýzy. Začíná diskuzí integrace nové aplikace se systémem Opentis a pokračuje popisem doménového modelu. Dále je prezentován návrh uživatelského rozhraní a softwarová architektura nového systému.

2.1 Integrace dat se systémem Opentis

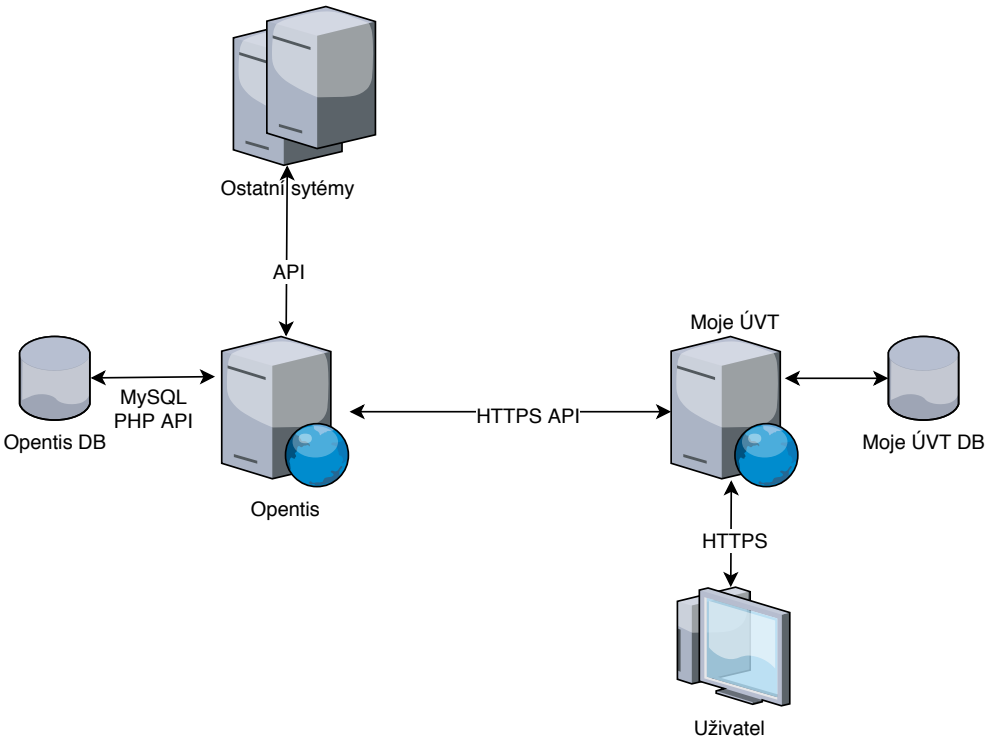
Integrace CRM systému Opentis a zákaznického portálu byla původně implementována tak, že se část dat systému Opentis replikovala do databáze aplikace. To bylo v sekci 1.1.1.1 ustanoveno jako neefektivní řešení, a proto tento návrh diskutuje jiný způsob integrace.

Integrace je navrhována tak, aby společná data obou systémů byla pouze v databázi systému Opentis a aplikace portálu k těmto datům přistupovala pouze pomocí zabezpečeného API, implementovaného na straně systému Opentis. Databáze nové aplikace bude obsahovat pouze údaje nutné pro přihlášení zákazníka, spolu s identifikátorem zákazníka v databázi systému Opentis. Tento přístup zajistí, že jakýkoliv pokus o přihlášení do portálu nebude zbytečně zatěžovat systém Opentis a pouze po validním přihlášení bude aplikace přistupovat přes API k datům zákazníka.

Při vzniku nového zákazníka v systému Opentis bude nutné, aby byly přihlašovací údaje zapsané i do databáze aplikace zákaznického portálu. Ta proto bude rovněž obsahovat API pro vytvoření přihlašovacích údajů nového zákazníka.

Aby se zabránilo nechtěnému úniku uživatelských dat, je nutné komunikaci přes API oběma směry zabezpečit tak, aby se k nim potenciální útočník nemohl dostat. Diskuze zabezpečovacích technik pro tuto komunikaci je uvedena v sekci 3.2.3.

Grafické znázornění výše popsané architektury je zobrazeno na obrázku 2.1, který také reprezentuje topologii komunikace mezi systémy a uživatelem.



Obrázek 2.1: Integrace systému Opentis a zákaznického portálu – Moje ÚVT

2.2 Databázový model

Databázový model přímo vychází z modelu doménového a obsahuje schéma databáze v nové aplikaci. Jak je vidět na obrázku 2.2, toto schéma obsahuje pouze jednu entitu *User*, jelikož zbytek entit doménového modelu je uchováván v databázi systému Opentis, jak bylo vysvětleno v sekci 2.1. Entita *User*

User		
PK	String	id
Unique	String	userId
	String	password
	Number	lastLogin
	String	resetPasswordToken

Obrázek 2.2: Databázový model v MongoDB

obsahuje pouze informace potřebné pro přihlášení a základní správu uživatele v aplikaci. Zbytek informací je uchován v systému Opentis jako entita *Customer*. Na obrázku 2.3 je vidět část zbývajících entit doménového modelu aplikace, který je uchován v databázi systému Opentis. Entity na obrázku nejsou

Obrázek 2.3: Část entit databázového modelu systému Opentis

propojeny žádnými čarami znázorňujícími relace, jelikož databáze systému Opentis používá databázi MySQL s MyISAM databázovým enginem, který nepodporuje cizí klíče. Veškeré relace jsou tedy řešeny na aplikační úrovni systému Opentis.

Entity uchovávané v systému Opentis navíc obsahují mnoho atributů, které jsou nepotřebné nebo irelevantní pro novou aplikaci, a proto je nutné při ná-

vrhu API pro integraci počítat s nutností filtrování těchto dat před přenosem.

2.3 Uživatelské rozhraní

Uživatelské rozhraní, neboli User Interface (UI), je jednou z nejdůležitějších částí aplikace. Uživatelské rozhraní rozhoduje o tom, zda uživatel bude moci snadno využít veškerou funkcionalitu aplikace, a proto je nutný nejen dobrý návrh, ale také otestování UI, které je diskutované v sekci 4.1.

2.3.1 Celkový vzhled

Jelikož má aplikace webovou formu, tak je tvořena stránkami ve webovém prohlížeči. Celkový vzhled aplikace tak můžeme rozčlenit do tří hlavních částí, a to záhlaví, tělo a zápatí stránky.

Implementační detaily uživatelského rozhraní jsou popsány v kapitole 3, kde se diskutují další rysy UI vycházející z použitého UI frameworku. Následující popisované části navrhovaného UI je možné vidět na obrázku wireframu 2.4 v sekci 2.3.2.

2.3.1.1 Záhlaví

Záhlaví stránky tvoří lišta obsahující logo aplikace, hlavní menu a správu uživatele. Toto rozvržení je praxí osvědčené a používá ho většina webových aplikací, což přispívá ke snazší orientaci uživatele při navigaci v aplikaci. Logo se nachází na levé straně a funguje také jako odkaz na výchozí stánku aplikace. Doprava pak pokračuje hlavní menu, které má horizontální formu. Jak bývá zvykem, při zobrazení aplikace na malém zařízení je menu transformováno do otevíratelné vertikální podoby. Na pravé straně lišty nalezneme odkaz na stránky zadavatele a vpravo od něj vertikální otevíratelnou nabídku pro správu uživatele, kde nalezneme odkaz na profil uživatele a také akce, jako je změna jazyka či odhlášení z aplikace.

2.3.1.2 Tělo

Obsah všech stránek nalezneme v těle stránky. Základními prvky těla obsahu tvoří dlaždice, tabulky, dialogy a varování.

Dlaždice
Většina komponent těla aplikace je tvořena dlaždicemi s různým obsahem. Uvnitř často nalezneme výpis dat formou seznamu s případnou ikonou či obrázkem. Dlaždicový styl zajišťuje dobré oddělení komponent, vyšší přehlednost a výbornou přizpůsobivost pro různě veliká zobrazovací zařízení, protože dlaždice se velmi snadno reorganizují do různých rozvržení.

Tabulky
Tabulky jsou využity pro výpisy velkého množství entit, jako jsou například faktury nebo informační zprávy.

Dialogy
Všechny formuláře a potvrzovací akce jsou navrženy formou otevíratelného dialogu. Dialog se otevře přes celou původní stránku a umožní interakci pouze s dialogem, dokud není uzavřen.

Varování
Varování či informační zpráva představuje prvek barevně ohraničeného pole s textem. Barva pak určuje, o jaký typ varování se jedná. Použité schéma barev je přebráno z praxe, kdy se nejčastěji používají následující barvy: zelená – úspěch, modrá – informační zpráva, oranžová – varování, červená – důležité varování/chyba.

2.3.1.3 Zápatí

Zápatí stránek bude tvořené lištou se standardními informacemi o stránce (rok vydání a jméno vydavatele) a také odkazem na kontakt, zpětnou vazbu zákazníka a změnu jazyka aplikace.

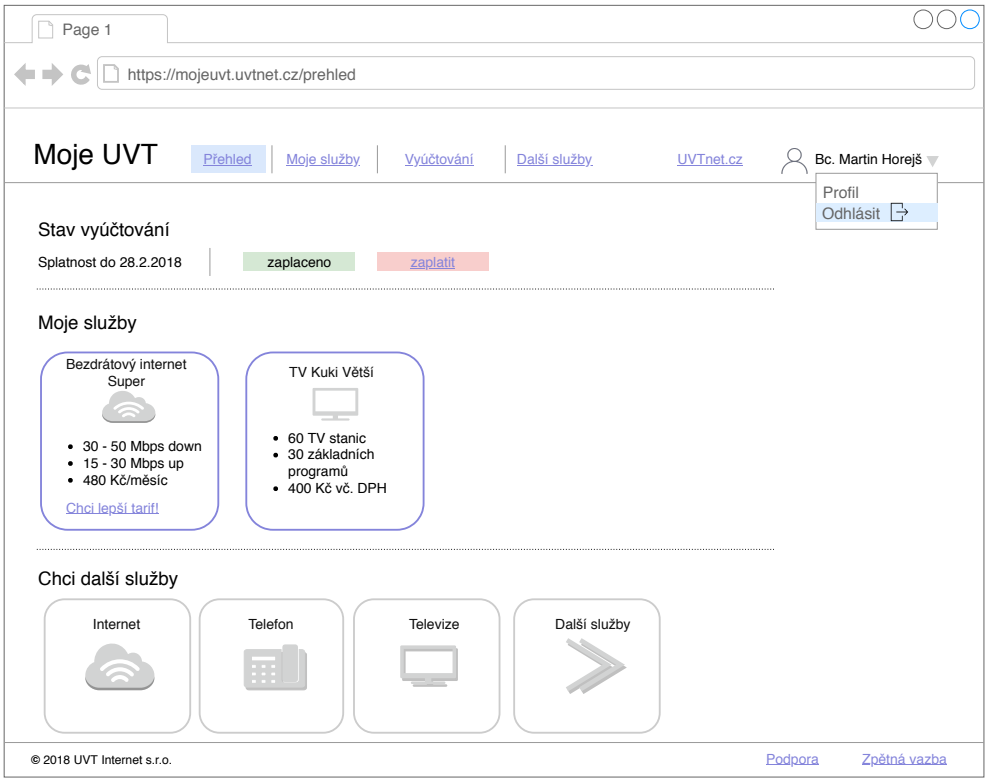
2.3.2 Wireframy obrazovek

Tvorba wireframů je jednou ze základních metod návrhu uživatelského rozhraní, která se zabývá především strukturou a rozvržením komponent UI aplikace. Každá obrazovka aplikace je rozkreslena podobně jako na obrázku 2.4. Wireframy jsou součástí projektové dokumentace a bylo nutné ho prodiskutovat se zadavatelem.

Na obrázku 2.4 vidíme návrh výchozí obrazovky. Nejedná se čistě o návrh rozložení prvků UI, protože prototyp obsahuje barevné prvky nebo ikony.

2.3.2.1 Barevnost

Barva a barevné prvky rozhodně patří mezi důležité faktory uživatelského rozhraní. V návrhu nedochází k malému kontrastu barev kvůli podobné intenzitě dvou odstínu jedné barvy a také nedochází k přílišnému kontrastu použitím doplňkových barev, které jsou například velmi nevhodné pro text. Návrh aplikace využívá především minimalistický design, který zvýrazňuje funkcionality a zlepšuje intuitivnost aplikace. Výsledný výběr barevného schématu ovšem závisí na vybraném UI frameworku v implementační části.



Obrázek 2.4: Wireframe: výchozí obrazovka

2.4 Architektura nové aplikace

Diskuze problematiky návrhu architektury se objevila už na začátku kapitoly 1. Tato sekce toto téma probírá hlouběji a rozebírá jednotlivé aspekty návrhu nové architektury.

2.4.1 Rozdělení aplikace

Stávající řešení funguje stejně jako mnoho starších webových aplikací, kde server aplikace zároveň generuje i podobu uživatelského rozhraní. Celá aplikace pak dostává monolitickou formu, kde UI je silně závislé na technologiích serverové části aplikace a grafické rozhraní je generované pomocí šablon s nutností vedlejšího klientského JavaScript kódu pro další funkcionalitu aplikace. Tento způsob implementace má historické důvody, kde starší webové prohlížeče jako Internet Explorer 7 nebyly výkonnostně schopné provádět složitější operace moderních webových aplikací. [4]

Dnešní webové prohlížeče jsou však velmi efektivní a výkonné u všech možných zařízení a tento archaický způsob vývoje lze opustit. Nové řešení bude tedy rozdělené na dvě části: frontend a backend. Frontendová část se bude

zabývat samostatnou implementací aplikace v uživatelově webovém prohlížeči. Backend aplikace je serverová část, která bude integrovaná s ostatními systémy zadavatele a bude provádět manipulaci a persistenci dat. Tato data budou filtrována a posílána přes nadefinované API do frontendu, kde s nimi uživatel může pracovat a v momentě nějakého požadavku bude frontend zpět komunikovat s backendem pomocí API. Frontend se tak stává nezávislý na technologiích backendu a může být implementován v úplně jiném programovacím jazyce.

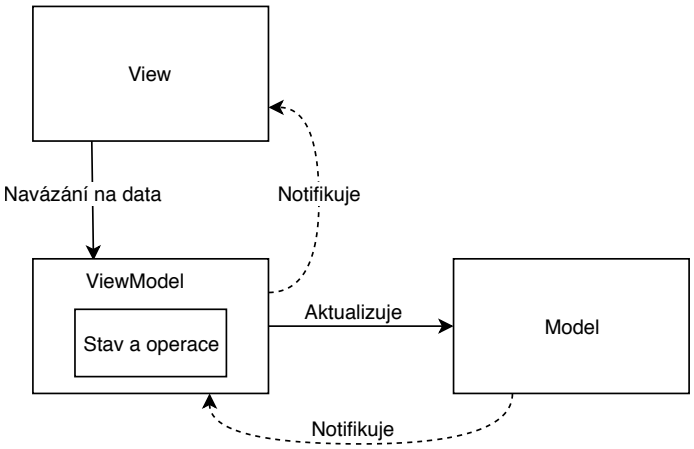
2.4.2 Návrhový vzor MVVM

Softwarová architekura MVVM (Model–View–ViewModel), podobně jako architektury MVC nebo MVP (Model–View–Controler/Presenter), rozděluje systém do tří hlavních vrstev s různou zodpovědností. Toto oddělení vrstev zajišťuje přehledný kód a tím napomáhá udržitelnosti dalšího vývoje aplikace.

Model je nejnižší vrstvou v tomto návrhovém vzoru. Jedná se o implementaci doménového modelu. Vyměňuje data s backendem a obsahuje business a validační logiku[5].

View je vrstva definující strukturu, rozložení a vzhled uživatelského rozhraní[5].

ViewModel je vrstva ležící mezi vrstvami model a view. Obsahuje aktuální stav modelu a obousměrně navazuje data ze stavu modelu přímo do vrstvy view.



Obrázek 2.5: Schéma modelu MVVM [5]

Myšlenka obousměrného navázání dat do vrstvy View se ukázala jako velmi praktická u odděleného vývoje frontendu single-page aplikací, a proto MVVM inspirovalo mnoho moderních webových frameworků, jako například *KnockoutJS*[6], *Angular*[7] nebo *Vue.js*[8].

V diskuzi původního řešení v sekci 1.1 bylo ustanoveno, že stávajícímu řešení chybí jasné oddělení těchto vrstev, které MVVM návrhový vzor softwarové architektury řeší. To bude při výběru frontendového frameworku jedno z důležitých kritérií.

2.4.3 API pro komunikaci frontendu s backendem aplikace

Aplikace bude potřebovat dvě API:

(i) API pro propojení frontendu s backendem.

(ii) API pro integraci backendu se systémem Opentis.

Architektura API pro komunikaci backendu a frontendu aplikace kompletně závisí na výběru frameworku backendu, a proto je diskutována až v sekci 3.1.3. U API pro integraci backendu se systémem Opentis je ovšem návrh API žádoucí.

2.4.4 API pro integraci se systémem Opentis

API pro integraci je naopak úplně nezávislé na technologiích frameworku, jelikož systém Opentis žádný framework neimplementuje, a proto je nutné připravit jeho návrh, který bude později od základů implementován v jazyce PHP na straně systému Opentis.

Jelikož systém Opentis funguje jako webový server interpretující skriptovací jazyk PHP, tak návrh ponechává nativní protokol této konfigurace – HTTP respektive HTTPS[3]. Dnešní API pro aplikace komunikující přes HTTP protokol většinou používají dva styly komunikace: REST a RPC. Tyto styly se od sebe zásadně liší ve svých konceptech. Nicméně není nutné držet se striktně jednoho stylu a API může koncepty těchto odlišných architektur libovolně kombinovat.

2.4.4.1 Koncept REST

REST neboli *representational state transfer* je koncept API, který funguje na bázi manipulace se zdroji. Se zdroji je možné manipulovat pouze standardními metodami, resp. „akcemi“ HTTP protokolu, jako jsou *GET, POST, DELETE* a další. Zdroje jsou identifikované pomocí URI. Například URI */zakaznik/1* identifikuje zdroj *zákazník* s identifikátorem ID 1.

Koncept REST je vhodný pro CRUD operace (create, read, update, delete) manipulaci doménového modelu a je výhodné ho implementovat, pokud aplikace potřebuje právě CRUD operace pro všechny entity doménového modelu. V ukázce kódu 2.1 je příklad použití HTTP REST API, kde je použita

[3]HTTP neboli Hypertext Transfer Protocol je protokol pro komunikaci s webovými servery[9]. Jeho rozšíření HTTPS (HTTP Secure) do původního protokolu přidává šifrování této komunikace[10].

```
1 POST /zakaznik/1/zpravy HTTP/1.1
2 Host: api.example.com
3 Content-Type: application/json
4
5 {"zprava": "Ahoj!"}
```

Ukázka kódu 2.1: Příklad použití HTTP API na bázi REST

metoda *POST*, která vytváří novou instanci zdroje definovaného pomocí URI. V tomto případě je to zdroj *zpráva*, které patří *zákazníkovi* s identifikátorem ID 1.

REST navíc nativně podporuje filtrování zdrojů, jako by bylo například zavolání `GET /zakaznik?vek=25&limit=2`, které by vrátilo kolekci o velikosti maximálně 2 zákazníků, kteří jsou ve věku 25 let.

2.4.4.2 Koncept RPC

RPC neboli *Remote Procedure Call* je další koncept návrhu API. Na rozdíl od REST RPC funguje na bázi operací a není limitován předdefinovaným seznamem „sloves“. RPC v podstatě volá vzdálené metody systému na druhé straně API. V ukázce kódů 2.2 je příklad, jak by mohlo vypadat použití RPC

```
1 POST /pridejZpravuZakaznikovi HTTP/1.1
2 Host: api.example.com
3 Content-Type: application/json
4
5 {"zakaznikId": 1, "zprava": "Ahoj!"}
```

Ukázka kódu 2.2: Příklad použití HTTP API na bázi RPC

HTTP API se stejným výsledkem jako použití konceptu REST v ukázce 2.1. Z ukázky je vidět, že toto API je explicitní, co se týče volání a i osoba neznalá tohoto API je schopna domyslet, co toto volání způsobí.

2.4.4.3 Porovnání konceptů API

Pro vytvoření optimálního návrhu API nového řešení aplikace je nutné vzít v potaz kapitolu Analýza a diskutovat možné výhody a nevýhody rysů, ať už jen jednoho či obou z konceptů.

Rozsah a funkcionalita API
Ze sekce 1.5 je známo, že doménový model aplikace je poměrně malý. Z funkčních požadavků ze sekce 1.2.1 také plyne, že většina funkcionality je zobrazovacího charakteru. Vytváření se týká pouze entity *Požadavek*. Dále je nadefinován také požadavek na aktivaci dalších služeb.

Žádná entita z doménového modelu tedy nepotřebuje implementovat všechny CRUD operace. Z tohoto hlediska se použití konceptu REST zdá být zbytečné a mnoho „akcí“ HTTP by tak nebylo implementováno. Tento problém u konceptu RPC už z jeho podstaty nevzniká.

Použití konceptu RPC by navíc u takto řídkého API zlepšilo jeho explicitnost a predikovatelnost jednotlivých volání.

Systém Opentis
Ze sekce 1.1.1.1 je známo, že systém Opentis nepoužívá žádný framework, ani vrstvu ORM a na místo toho volá přímo API databáze MySQL. Doménový model navíc není konzistentně naimplementován a některé jeho integrace se systémy třetích stran jsou volané v externích skriptech.

Tento problém zásadně znemožňuje použití konceptu REST, ve kterém je zachycení akcí vyžadujících volání rozhraní třetích stran problematické. Taková implementace by znamenala zásadní zásah do původní implementace systému Opentis.

Na rozdíl od toho u konceptu RPC tento problém nenastává, jelikož taková operace by se dala právě nadefinovat pomocí vzdálené funkce, která by ji provedla.

Dalším problémem použití konceptu REST je modifikace dat použitím HTTP metody *PUT*. Tato metoda umožňuje modifikovat jakákoliv data určeného zdroje, což povede k velmi komplikované implementaci (mnoho podmínek a kontrol všech přijatých dat), jelikož zdroje obsahují velmi mnoho atributů, přičemž jen velmi málo z nich je nutné modifikovat pro funkcionalitu nové aplikace. Implementace by tak musela řešit všechny možné případy při změnách těchto dat, nebo změny některých atributů úplně vynechat, což by zapříčinilo nekonzistentnost takového REST API.

RPC tento problém nemá, jelikož je založené na implementaci těchto izolovaných operací s daty a implementuje tak pouze potřebné akce na rozdíl od v tomto smyslu abstraktněji definovaného REST modelu. RPC se tak podobá více modelu: „nejdříve vše zakaž a poté žádoucí požadavky povol“ oproti RESTu: „nejdříve vše povol a poté nežádoucí požadavky zakaž“.

2.4.4.4 Návrh API

Návrh API se drží spíše rysů RPC, konkrétně JSON-RPC, než RESTu kvůli výše zmiňovaným výhodám. Nicméně obsahuje více zakončení URL podle zdroje, kterého se volání týká, což je inspirované modelem REST. V ukázce kódu 2.3 je vidět volání nového API, které provede uložení zpětné vazby od zákazníka do databáze systému Opentis. Forma volání je založena na standardu JSON-RPC, jehož příklad je zobrazen v ukázce kódu 2.4. Tento standard pracuje zásadně s formátem JSON[4] a skoro výhradně s HTTP metodou *POST*.

[4]JSON, neboli JavaScript Object Notation, je jednoduchý, textový a jazykově nezávislý formát pro výměnu dat[12].

```
//request
POST /api?r=customer HTTP/1.1
Host: example.com
Content-Type: application/json
API-Token: dGhpcyBpcyBvbmx5IGEgdGVzdCB0b2tlbg

{"method": "sendCustomerFeedback",
 "params": {
  "id_customer": 1,
  "message": "this is the feedback",
  ...
 }
}

//response
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2018 12:28:53 GMT
```

Ukázka kódu 2.3: Příklad použití navrhovaného HTTP API.

```
-> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<- {"jsonrpc": "2.0", "result": 19, "id": 1}

-> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse error"
    }, "id": null}
```

Ukázka kódu 2.4: Volání standardu JSON-RPC [11]

V těle zprávy standardu JSON-RPC se nachází atributy pro verzi standardu a ID zprávy, které jsou pro nové API nepotřebné[5], a byly tak z návrhu vynechány. Odpovědi nového API na požadavky mohou na rozdíl od standardu JSON-RPC nabývat libovolné formy. Stav odpovědi je totiž definován HTTP stavovým kódem, jako je například `200 OK`, a výsledek odpovědi je pak přímo v těle zprávy, které může mít libovolnou formu, či být zcela prázdné. Tento rozdíl je zásadní oproti JSON-RPC a umožňuje tak odpovídat na požadavky i jiným obsahovým typem než je JSON, což může být například dokument PDF, obrázek JPEG a další.

2.4.4.5 Zabezpečení API

Nové API musí být zabezpečené proti neoprávněnému použití, a proto je třeba zaručit, že všechny zaslané požadavky budou nějakou formou autorizované pro provedení. Mimo restrikcí na síťové vrstvě pomocí firewallu návrh počítá s autorizací pomocí tokenu. Tento token je náhodný řetězec bezpečné délky,

[5] Požadavek a jeho odpověď jsou k sobě automaticky spárovány na úrovni HTTP serveru, a proto není nutné je párovat podle ID.

jež zná pouze strana, která je autorizována k využití tohoto API a zasílá ho v HTTP hlavičce *API-Token* jak je vidět v ukázce 2.3.

Kapitola 3

Implementace

Tato kapitola popisuje výběr technologií pro implementaci aplikace podle návrhu z předchozí kapitoly. Dále jsou diskutovány další rysy vybraných technologií a implementačních detailů. Veškeré ukázky kódů počínaje touto kapitolou a dále budou v anglickém jazyce, protože i celá implementace nové aplikace je psána v angličtině podle zaběhnutých programátorských konvencí.

3.1 Výběr technologií

Vzhledem k rozdělení vývoje aplikace do dvou oddělených větví frontendu a backendu v sekci 2.4.1 se nabízí vybrat stejnou technologii a programovací jazyk pro obě části. To přispěje k lepší čitelnosti kódu a snadnějšímu vývoji, jelikož programátor nebude muset přemýšlet v kontextech odlišných technologií. Dále je také možné obě větve vyvíjet ve stejném prostředí se stejnou správou balíčků či knihoven.

Frontend aplikace je oddělen od backendu a jeho implementace je spouštěná ve webovém prohlížeči zákazníka. Dnes prakticky všechny moderní webové prohlížeče interpretují klientský JavaScript kód. Z tohoto plyne, že frontend aplikace ve webovém prohlížeči musí být implementován v jazyce JavaScript, respektive v některém z jeho frontendových frameworků, jako je například Angular, React nebo Vue.js.

Použitím technologie Node.js pro backend je zachován jazyk implementace obou vývojových větví, čímž je dosáhnuto výhod popsaných výše. Pro ještě lepší synchronizaci technologií vývoje je také vybrána technologie MongoDB – databáze na bázi JSON dokumentů.

3.1.1 Node.js

Node.js je moderní runtime prostředí jazyka JavaScript pro vytváření síťových aplikací s dobrou škálovatelností, která je umožněna díky neblokujícímu asynchronnímu vykonávání událostí v jednom vláknu. Implementace protokolu

HTTP je jednou z hlavních částí Node.js, což z něj dělá dobrého kandidáta na implementaci webových frameworků a knihoven[13].

Node.js je postaven na projektu V8[14], což je velmi výkonný open-source JavaScript engine od společnosti Google, který je napsán v jazyce C++[15].

Instalace Node.js je k dispozici v různých distribucích pro operační systémy Linux, macOS i Windows. Spustitelný binární soubor má název *node* a příklad spuštění aplikace v Node.js může vypadat například takto: `node app.js`, kdy se spustí proces *node*, který začne interpretovat jazyk JavaScript ze souboru s cestou *app.js*. Node.js je také možné spustit samostatně bez argumentů pouze jako `node`, což otevře interaktivní konzoli.

Příklad jednoduché aplikace pro Node.js je v ukázce kódu 3.1, která spustí lokální webový HTTP server naslouchající na portu 3000 a na veškeré požadavky odpovídá textem „Hello World“. Na řádku 12 je také vidět asynchronní technika Node.js, kde se zavolá spuštění naslouchání serveru a této asynchronní metodě je zároveň předána tzv. callback funkce, která je spuštěná poté, co tato metoda skončí. Z tohoto vyplývá, že proces interpretace této ukázky by po zavolání metody `server.listen` nečekal na výsledek a ihned pokračoval v provedení dalších případných instrukcí na řádku 15 a dále.

```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at ${hostname}:${port}`);
14 });
```

Ukázka kódu 3.1: Příklad jednoduché aplikace v Node.js [13]

3.1.1.1 npm

npm neboli *Node Package Manager* je výchozí open-source nástroj pro správu balíčku Node.js. *Npm* operuje s největším registrem softwaru na světě a umožňuje jednoduché vyhledání a zakomponování znovupoužitelného kódu do nových projektů[16]. Použití tohoto nástroje je velmi snadné, jelikož je nainstalován jako součást Node.js. Volání se provádí pomocí příkazové řádky například takto: `npm install lodash`. Tento příkaz spustí instalaci balíčku *lodash* v aktuálním adresáři příkazové řádky.

Veškeré závislosti a další informace o projektu využívajícího *npm* jsou definované v souboru *package.json*, který je umístěn v kořenovém adresáři takového projektu. Příklad jak takový soubor může vypadat je v ukázce kódu 3.2.

```
{
  "name": "new-application",                //název aplikace
  "version": "0.0.1",                       //aktuální verze aplikace
  "description": "New application",         //popis
  "author": "John Doe <doe@example.com>",   //autor aplikace
  "private": true,                          //privátní projekt?
  "scripts": {                              //spouštěcí skripty npm
    "test": "mocha ...",
  },
  "dependencies": {                         //závislosti projektu
    "deepmerge": "^2.1.1",
    "lodash": "^4.17.10",
    "moment": "^2.22.2",
  },
  "devDependencies": {                      //záv. projektu pro vývoj
    "mocha": "^5.2.0",
    "nyc": "^12.0.2",
  }
}
```

Ukázka kódu 3.2: Příklad souboru package.json

3.1.2 MongoDB

MongoDB je open-source databázový systém, který podporuje distribuovaný model a data ukládá do kolekcí, které nemají pevné schéma. Předpokládá se u nich, že jejich data budou mít podobný tvar. Tyto vlastnosti řadí MongoDB do kategorie NoSQL databází, které se konceptuálně odlišují od klasických SQL relačních databázových systémů.

MongoDB ukládá záznamy dat do dokumentů formátu podobnému JSON objektů. Tyto dokumenty jsou kolekcemi dvojic: klíč, hodnota. Hodnoty dokumentů odpovídají nativním typům mnoha programovacích jazyků[17]. Dokumenty mají možnost skládání, což značně snižuje počet nutných dotazů pro získání dat oproti standardním relačním databázím, které musí provést několik relativně drahých join operací. Příklad složených dokumentů je v ukázce kódů 3.3, kde je vidět, že hodnota klíče `name` je další dokument. Pro přístup k datům vloženého dokumentu se používá následující tečková dereference: `contact.phone.number`.

Skládání objektů nemusí být vždy vhodné, jelikož může docházet k opakování stejných záznamů, a navíc je velikost jednoho dokumentu omezena na 16 megabajtů [18]. Z tohoto důvodu lze použít stejného konceptu jako u relačních

```
1 {
2   ...
3   _id: 2323232323,
4   name: { first: "Alan", last: "Turing" },
5   birthday: "2017-02-01",
6   contact: { phone: { type: "cell", number: "111-222-3333" } },
7   children: [123456789, 234567890]
8   ...
9 }
```

Ukázka kódu 3.3: Příklad dokumentu v MongoDB

databází a místo vloženého dokumentu použít identifikátor na odkazovaný dokument. Tato možnost je ukázána na řádku 7 ukázky 3.3, kde osoba, kterou ukázka zachycuje, je rodičem dalších osob s identifikátorem _id 123456789 a 234567890.

Identifikátor _id je primární klíč každého dokumentu, který není při tvorbě záznamu povinný, nicméně v takovém případě je vygenerován automaticky[19]. MongoDB vytváří unikátní index nad tímto klíčem, což zajišťuje, že není možné vložit do stejné kolekce dva dokumenty se stejným _id[20].

Propojení aplikací založených na jazyku JavaScript, jako je například Node.js, s MongoDB se díky velmi podobné notaci formátu JSON a tečkové dereferenci vnořených objektů stává výhodou. Implementace komunikace databázové vrstvy s backendem aplikace je pak z pohledu programátora jednoduší a přirozenější.

3.1.3 Sails.js

Vzhledem k tomu, že osobně nemám příliš mnoho zkušeností s backend webovými frameworky v Node.js, bylo při výběru důležité, aby měl framework výbornou dokumentaci a podporu komunity, což by přispělo k rychlému nalezení řešení potenciálních problémů při implementaci. Z tohoto důvodu byl nejprve vybrán framework *Express.js*, který je mezi základními frameworky Node.js nejpopulárnější[21] a je velmi dobře zdokumentovaný[22]. *Express.js* ovšem poskytuje spíše nízkoúrovňové API, zatímco implementace aplikace vyžaduje pokročilejší technologie, jako ORM nebo generování z emailových šablon. Z tohoto důvodu byl nakonec vybrán framework *Sails.js*, který je jak založen na *Express.js*, tak poskytuje právě potřebné funkcionality pro tuto implementaci.

Sails.js je webový framework pro backend postavený na technologii Node.js. Jeho základ stojí také na webovém frameworku Express.js a knihovně pro komunikaci přes protokol WebSocket – Socket.io.

Sails je také kompatibilní s jakoukoliv databází díky vlastnímu ORM nástroji Waterline, který toto umožňuje díky obecnému namapování doménového modelu na databázi. Díky namapování doménového modelu umí Sails také automaticky generovat REST API pro jeho manipulaci. Výběr frontend

technologie také není omezen, jelikož Sails je agnostický vůči všem frameworkům a je tedy kompatibilní s například Angular, React, Android, Vue.js a dalšími frontendy.

3.1.3.1 Popis ukázkové struktury aplikace Sails

Struktura aplikace Sails je předem definovaná a její zjednodušená ukázková forma je na obrázku 3.1.

```
api ..... adresář s logikou aplikace
  controllers ..... adresář s implementací API
    UserController.js ..... kontroler s akcemi uživatele
  helpers ..... adresář s implementací pomocných služeb
  models ..... adresář s definicí modelu a propojení databáze
    User.js ..... entita User s definovaným ORM mapováním
  policies ..... adresář s implementací autorizačních filtrů
    isAuthorized.js ..... kontrola autorizace zákazníka
assets ..... statický obsah: obrázky, kód frontendu v JavaScriptu atp.
config ..... adresář se soubory obsahující konfiguraci
  routes.js ..... mapování cest URL na akce
  policies.js ..... mapování filtrů (policies) na akce
views ..... adresář s šablonami pro generování HTML kódu
```

Obrázek 3.1: Zjednodušená struktura backendu

Modely
Modely jsou v Sails definice entit doménového modelu. Obsahují popis atributů, které umožní automatické mapování ORM na databázi aplikace. V ukázce 3.4 je příklad pro entitu *User*, kde jsou nadefinované atributy jméno, datum narození a email. Modely mají automaticky vygenerované metody pro jejich manipulaci jako například `User.find()`. Dále je možné také definovat vlastní konkrétnější metody pro složitější práci s entitami v databázi. Takové metody se poté nacházejí ve stejném souboru jako definice atributů.

```
1 //User.js
2 attributes: {
3   name: { type: 'string', required: true },
4   birthDay: { type: 'number', required: true },
5   email: { type: 'string', allowNull: true },
6 },
7 async function customMethod(){...}
```

Ukázka kódu 3.4: Příklad modelu ve frameworku Sails

Kontroler a akce

Kontroler je v Sails základní jednotkou definující logiku aplikace a nachází se v adresáři *controllers*. Obsahuje akce, které se provádí v momentě, kdy přijde požadavek od uživatele.

V ukázce 3.5 je vidět příklad takové akce. Jedná se o asynchronní funkci, která přijímá dva parametry. Tyto parametry jsou objekty reprezentující požadavek uživatele (request) a odpověď na tento požadavek (response). V této akci se z požadavku uživatele převezme parametr *userId*, podle kterého se pak v databázi na řádku 3 hledá odpovídající uživatel. Akce pak odpoví pozdravem *Hello user!* podle toho, jestli takový uživatel v databázi je a v opačném případě zašle HTTP kód 404 reprezentující odpověd „nebylo nalezeno“.

```
1 async function welcome(request, response) {
2   const userId = request.param('userId');
3   var user = await User.findOne({ id: userId });
4   if(user) return response.send("Hello user!");
5   return response.notFound();
6 }
```

Ukázka kódu 3.5: Příklad akce ve frameworku Sails

Filtry a omezení

Ve složce *policies* jsou definované filtry příchozích požadavků, které jsou vhodné například k autorizaci příchozího požadavku.

V ukázce 3.6 je příklad takového filtru. Tento filtr povolí zpracování příchozích požadavků z IP adresy 8.8.8.8 a ostatní odmítne s HTTP kódem 403 „zamítnuto“. Tyto filtry jsou pak mapované na akce v souboru *policies.js*, kde ať už pro celé kontrolery nebo pouze jejich vybrané akce lze zapnout několik různých filtrů.

```
1 function isAuthorized (req, res, next) {
2   const ip = req.ip
3   if (ip !== "8.8.8.8") return res.forbidden()
4   return next()
5 }
```

Ukázka kódu 3.6: Příklad akce ve frameworku Sails

URL cest

Soubor *routes.js* obsahuje pravidla pro mapování URL cest na akce kontrolerů. Toto mapování definuje, jaké URL cesty aplikace budou dostupné a jakou budou provádět akci. Mapování se skládá z trojic hodnot: HTTP metoda, URL a mapovaná akce ve formátu „kontroler/akce“. Příklad mapování může vypadat následovně `'get /user/welcome': 'user/welcome'`.

Statický obsah aplikace
Ve složce *assets* se nachází statický obsah aplikace. Může se jednat například o obrázky, videa, hudbu či klientský JavaScript kód. Právě zde se bude nacházet implementace frontendu aplikace, protože frontend je reprezentován klientským JS kódem.

3.1.4 Vue.js

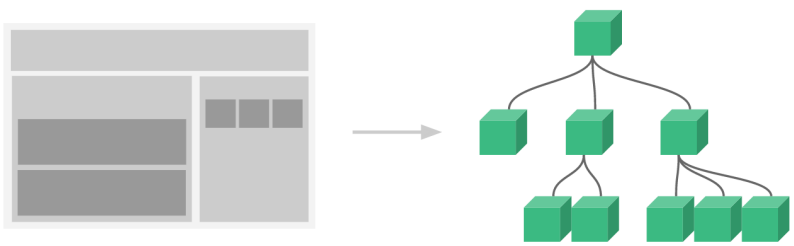
Frontendové web frameworky jako Angular, React nebo Vue jsou více méně srovnatelné co do možností vývoje a jejich funkcionalit. Vzhledem k mé velmi malé zkušenosti s vývojem frontendu v těchto pokročilých frameworcích jsem osobně vybral takový, který byl velmi srozumitelný pro začátečníka a který se dal snadno a poměrně rychle naučit. Vue má oproti Angularu a Reactu mnohem příjemnější učící křivku[23] a spolu s výbornou dokumentací[24] protkanou mnoha ukázkami kódu, je proto velmi vhodnou volbou pro začínajícího programátora v JavaScript frontend frameworcích.

Vue.js je progresivní webový framework určen pro tvorbu uživatelského rozhraní. Jádro této knihovny je soustředěno na zobrazovací vrstvu, ale spolu s dalšími nástroji je Vue schopno vytvořit sofistikované single-page aplikace.[25]

Single-page aplikace znamená, že po prvotním načtení webové stránky se již při libovolné akci nebude stránka aplikace nahrávat znovu, ale mezi backendem a aplikací se přes API na pozadí posílají pouze data nutná pro provedení této akce. Samotná stránka se pak znovu nenačítá, ale dynamicky se aktualizují pouze prvky aplikace ve webovém prohlížeči, které byly touto akcí ovlivněny.

3.1.4.1 Koncept komponent

Vue je založeno na konceptu skládání komponent. To umožňuje tvorbu rozsáhlých aplikací tvořených pomocí malých, zapouzdřených a znovu použitelných komponent. Skládání komponent skoro každého typu aplikace vytváří pomyslný strom komponent, jako je vidět na obrázku 3.2.



Obrázek 3.2: Strom komponent [26]

Jednotlivé komponenty jsou definované v souborech s příponou *.vue*. V ukázce 3.7 je příklad definice kořenové komponenty s názvem *MyApp*. Soubor komponenty se skládá ze tří částí: šablona, skript a vzhled.

Šablona je HTML kód mezi značkami `<template></template>`, který tvoří rozložení komponenty. Na řádku 4 je vidět skládání komponent, kde se do šablony vloží jiná komponenta s názvem *MyList*.

Skript aplikace obsahuje hlavní logiku komponenty. Na řádku 9 je vidět importování již zmíněné komponenty *MyList*, která je později lokálně registrovaná v této komponentě na řádku 13. Pole `props` obsahuje názvy atributů HTML, přes které se předávají data z rodičovské komponenty. Například na řádku 4 je vložena komponenta *MyList* s atributem *items*, do kterého je předán objekt `listItems` z dat, která jsou spravovaná komponentou *MyApp*.

```
1  <template>
2    <div id="app">
3      <h1>My App!</h1>
4      <MyList :items="listItems"/>
5    </div>
6  </template>
7
8  <script>
9    import MyList from './components/MyList.vue'
10
11   export default {
12     name: "MyApp",
13     components: { MyList },
14     props: [],
15     data: ()=>({
16       listItems: [ {id: 1, text: "Text1"}, {id: 2, text: "Text2"} ]
17     }),
18     methods: {}
19   }
20 </script>
21
22 <style lang="css">
23   #app {
24     max-width: 400px;
25   }
26 </style>
```

Ukázka kódu 3.7: Příklad definice kořenové komponenty ve Vue

3.1.4.2 Vue Router

Vue Router je třída z frameworku Vue, která řeší routování URL stránek instance aplikace Vue. Routování probíhá podle URL cest, ke kterým se namapuje komponenta. Vue také obsahuje komponentu *router-view*, která zobrazuje odpovídající namapovanou komponentu podle aktuální URL cesty.

Část konfigurace Vue routeru aplikace je v ukázce 3.8, kde jsou namapovány cesty na komponenty přihlašovacího formuláře a přehledu po přihlášení uživatele. Na řádku 11 je cesta `/login` definována jako veřejná metaatributem

`public`. To umožní uživatelům zobrazit přihlašovací formulář, ještě než jsou autentifikovaní. Ostatní cesty, které nejsou definované jako veřejné, jsou nepřihlášeným uživatelům nepřístupné a budou přesměrovány zpět na přihlašovací formulář. To je zajištěno mechanizmem na řádcích 23 až 33.

```
1  import Dashboard from '../components/Dashboard'
2  import LoginForm from '../components/LoginForm'
3  let router = new Router({
4    mode: 'history',
5    routes: [
6    {
7      path: '/login',
8      name: 'login',
9      component: LoginForm,
10     meta: {
11       public: true
12     }
13   },
14   {
15     path: '/dashboard',
16     name: 'dashboard',
17     component: Dashboard
18   }
19   ...
20   ]
21 })
22 // authenticated only
23 router.beforeEach((to, from, next) => {
24   if (to.matched.some(record => record.meta.public !== true)) {
25     if (window.localStorage.getItem('token')) {
26       next()
27     } else {
28       router.push({name: 'login'})
29     }
30   } else {
31     next()
32   }
33 })
```

Ukázka kódu 3.8: Příklad konfigurace Vue routeru

3.1.4.3 VueTranslate

VueTranslate je plugin pro framework Vue.js, který řeší internacionalizaci aplikace. HTML šablony Vue komponent mohou být pomocí tohoto pluginu použity pro překlad textu. Překlad textů je určen slovníkem definovaným přímo v JavaScript kódu, nebo v importovaném souboru ve formátu JSON.

Ukázka, jakým způsobem je tento plugin využíván, je vidět v ukázce kódu 3.9. Ukázka definuje komponentu *TranslateExample*, která obsahuje pouze

HTML element `span` s přeloženým textem „welcome“. Výsledek překladu závisí na slovníku definovaném v členské proměnné `locales` ukázkové komponenty. Jazyk překladu je daný řádkem 16, který ho nastavuje na `cs`, tudíž bude použit český slovník. Nakonec je ve vybraném slovníku hledán klíč `welcome`, který má v českém slovníku definovanou hodnotu jako „Vítejte“, což je i výsledek překladu ukázky.

```
1  <template>
2  <spant>{{t('welcome')}}</span>
3  </template>
4
5  <script>
6  import Vue from 'vue';
7  import VueTranslate from 'vue-translate-plugin';
8  Vue.use(VueTranslate); //register the plugin
9  export default {
10   name: 'TranslateExample',
11   locales: {
12     cs: {welcome: "Vítejte"},   //czech locale
13     de: {welcome: "Willkommen"} //german locale
14   },
15   created () {
16     this.$translate.setLang("cs") //setting the locale to cs -> result is Vítejte
17   }
18 }
19 </script>
```

Ukázka kódu 3.9: Příklad použití VueTranslate pluginu

3.1.5 Material design

V sekci 2.3 byl specifikován návrh uživatelského rozhraní aplikace, ale finální vzhled aplikace nikoliv. Pro tento návrh byl v implementaci použit standardizovaný vzhled *material design*.

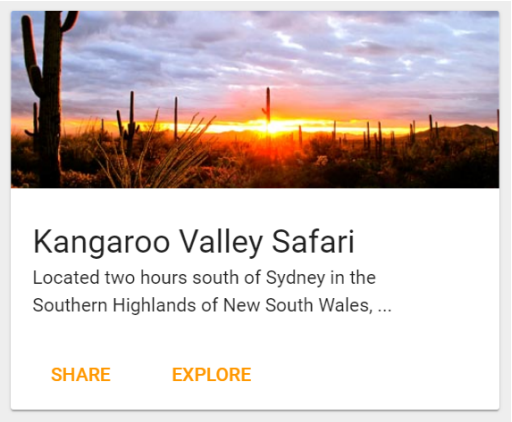
Material design je vizualizační jazyk vyvinutý společností Google, který definuje, jak by mělo vypadat uživatelské rozhraní. Tyto směrnice obsahují doporučení pro typografii, velikosti komponent, vzdálenosti mezi prvky či použití stínů pro navození plastického efektu prvku. Základním cílem tohoto standardu je imitovat materiály z reálného světa s přidanou funkcionalitou světa softwaru. Lidský mozek je díky tomu schopný lépe a rychleji pochopit uživatelské rozhraní a jeho funkcionalitu[27].

3.1.6 Vuetify

Vuetify je framework implementující material design, který vystavuje API obsahující širokou řadu komponent. Tento framework je postavený na technologii Vue.js, což umožňuje velmi snadné rozšíření tohoto frameworku do vlastních komponent Vue.js. Na obrázku 3.3 je výsledek Vuetify ukázky kódu 3.10.

```
<v-card>
  <v-img src="https://cdn.vuetifyjs.com/images/cards/desert.jpg" />
  <v-card-title primary-title>
    <h3 class="headline mb-0">Kangaroo Valley Safari</h3>
    <div>Located two hours south of Sydney in the <br>Southern Highlands of New
        South Wales, ...</div>
  </v-card-title>
  <v-card-actions>
    <v-btn flat color="orange">Share</v-btn>
    <v-btn flat color="orange">Explore</v-btn>
  </v-card-actions>
</v-card>
```

Ukázka kódu 3.10: Příklad použití komponent Vuetify[28]

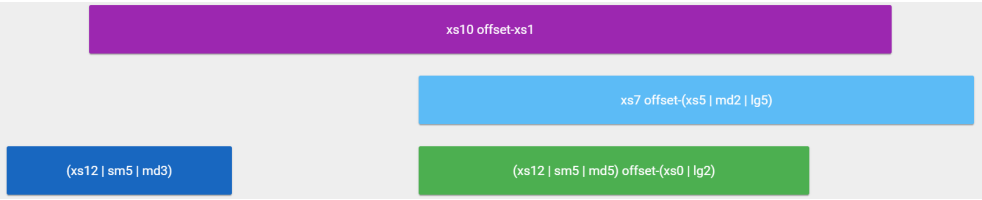


Obrázek 3.3: Výsledný vzhled ukázky 3.10[28]

Ukázka je velice stručná co do délky kódu, zatímco výsledek vypadá mnohem sofistikovaněji. V další fázi psaní této ukázkové komponenty by stejně jako v ukázce 3.7 přibyly sekce `<style>` a `<script>`, které by definovaly další vzhled či chování komponenty například při interakci s viditelnými tlačítky *SHARE* a *EXPLORE*.

3.1.6.1 Systém mřížky

Vuetify používá pro výpočet pozice zobrazení komponent systém mřížky. Tato mřížka horizontálně rozděluje místo komponenty na 12 bodů. Velikost komponenty je tak definovaná například HTML atributem `xs12`, který definuje velikost přes všech 12 bodů. Prefix `xs` znamená, že je tato velikost nastavena pro velmi malá zařízení. Vuetify definuje tyto prefixy od malých až po velká zařízení: `xs, sm, md, lg, xl`. Na obrázku 3.4 je zobrazen tento systém mřížky v akci spolu s dalšími pokročilými pravidly, jako je offset.



Obrázek 3.4: Systém mřížky frameworku Vuetify [29]

3.2 API pro integraci se systémem Opentis

Z návrhu v sekci 2.4.4 je známo, že API systému Opentis musí být naprogramováno v zaběhlé struktuře tohoto systému. Zpracování návrhu tohoto API, které bylo strukturálně navrženo obdobně pomocí kontrolerů, jako ve frameworku Sails a dalších, využívajících tento koncept.

3.2.1 Struktura souborů API

Na obrázku 3.5 je zobrazena souborová struktura naimplementovaného API. Jediný vstupní bod do API je soubor *index.php*, který podle vlastností požadavku na API rozděluje požadavky na odpovídající handlery v adresáři *handlers*. Soubor *Config.php* obsahuje různá nastavení API co se týče autorizace a dalších vlastností.

```
api ..................................................... adresář s API
├── index.php ........................................... vstupní bod do api
├── handlers ............................ adresář s implementací handlerů
│   ├── Handler.php ..................................... abstraktní handler
│   ├── CustomerHandler.php ................ handler s akcemi zákazníka
│   ├── CustomerMessageHandler.php . handler s akcemi zpráv zákazníka
│   ├── CustomerRequestHandler.php .. handler s akcemi požadavků zák.
│   ├── InvoiceHandler.php ..................... handler s akcemi faktur
│   ├── ServiceHandler.php ..................... handler s akcemi služeb
│   ├── SledovaniTvHandler.php ... handler s akcemi služby SledovaniTV
│   └── SubscriptionHandler.php ............. handler s akcemi paušálů
└── config
    └── Config.php ..................... pomocná třída pro nastavení API
```

Obrázek 3.5: Struktura API systému Opentis

3.2.2 Proces zpracování požadavku na API

V ukázce 3.11 je vypsán soubor *index.php*, který funguje jako vstupní bod do tohoto API. Ostatní soubory nelze spustit přímo, ale musejí být importované, což je zaručeno podmínkou `if (count(get_included_files()) == 1) exit;`.

Na řádku 3 se provádí inicializace konfigurace provedení skriptu, poté se na řádku 5 autorizuje nebo zamítne příchozí požadavek. V druhém případě je vyhozena výjmka se statusem `CODE_FORBIDDEN` „zamítnuto“, která je zpracovaná v bloku `catch` na řádku 25, a na řádku 28 je na takový požadavek odeslána odpověď s odpovídajícím chybovým kódem HTTP.

Při autorizovaném požadavku pokračuje skript dále ve vyhodnocování na řádku 8, kde je z URL parametru `r` vyčten název handleru, který má daný požadavek zpracovat. Na řádku 11 je poté vybrán odpovídající handler a jsou mu předána příchozí data z požadavku a konečně po dalších validacích je na řádku 19 zavolána odpovídající metoda ve stylu podobnému RPC, jak bylo navrženo v sekci 2.4.4.

```
1  <?php
2  include_once("config/Config.php");
3  Config::initialize();
4  try {
5    if (!Config::authorizeIncomingRequest())
6      throw new Exception('Forbidden', Handler::CODE_FORBIDDEN);
7    $URI = [];
8    $URI = explode('/', trim(array_key_exists('r', $_GET) ? $_GET['r'] : '
         ', '/'));
9    $data = json_decode(file_get_contents('php://input'));
10   $resource = strval($URI[0]);
11   $handler = Handler::getChildHandler($resource, $data);
12
13   if (!$handler || !count($URI)) {
14     throw new Exception('Bad request', Handler::CODE_GENERAL_ERROR);
15   } else {
16     if (empty($data->method) || empty($data->params) || empty($data->
           meta))
17     throw new Exception('Bad request', Handler::CODE_GENERAL_ERROR);;
18     Handler::log($data->meta->ip . ' - ' . $data->meta->action);
19     $handler->invokeMethod();
20   }
21
22 } catch (ErrorException $e) {
23   Handler::log($e->getMessage());
24   Handler::sendError(Handler::CODE_INTERNAL_ERROR);
25 } catch (Exception $e) {
26   Handler::log($e->getCode() . ': ' . $e->getMessage());
27   $code = $e->getCode() ? $e->getCode() : Handler::CODE_GENERAL_ERROR;
28   Handler::sendError($code, $e->getMessage());
29 }
```

Ukázka kódu 3.11: Vstupní bod do API – index.php

Metoda `invokeMethod` třídy *Handler* poté pomocí reflexních metod jazyka PHP ověří, zda volaná funkce API existuje ve vybraném potomkovi třídy *Handler* a zdali obsahuje správný počet parametrů pro korektní zavolání této

funkce. Po veškerých kontrolách je odpovídající funkce zavolána a je odeslána výsledná odpověď na příchozí požadavek.

3.2.3 Zabezpečení

Co se týče zabezpečení, implementace API spoléhá na protokol HTTPS proti odposlechnutí dat a autorizace probíhá pomocí tokenu, který je tvořen bezpečně dlouhým náhodným řetězcem, který je zasílán v HTTP hlavičce `API-TOKEN`. Další vrstvou autorizace tohoto API je funkcionalita HTTP – *Basic access authentication*[6], která je zapnutá pro systém Opentis. V pravidlech této funkcionality pak existuje výjimka pro IP adresu backendu aplikace.

3.3 Frontend aplikace

Jak bylo řečeno v sekci 3.1, frontend aplikace bude implementován technologií Vue.js s pomocí komponentového frameworku Vuetify. Základní rozvržení aplikace se velmi podobá doporučenému rozvržení tohoto frameworku a můžeme ho vidět v ukázce 3.12. Na řádku 6, 12 a 18 jsou do tohoto rozvržení vloženy tři hlavní části aplikace.

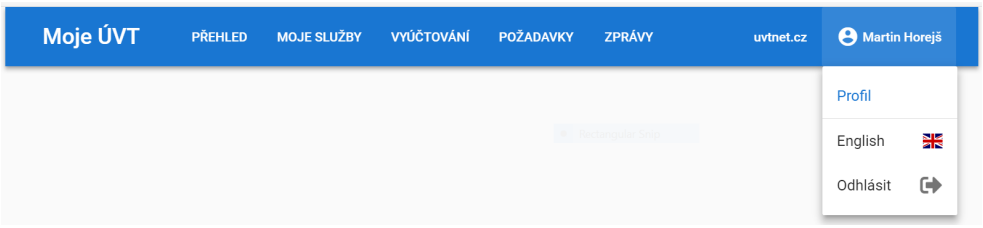
```
1  <template>
2    <v-app>
3      <v-container fluid grid-list-xl text-xs-center class="py-0 px-3" style="
           z-index: 1">
4        <v-layout row wrap>
5          <v-flex xs12 sm12 offset-sm0 lg10 xl8 offset-xl2 offset-lg1 class="pt-0">
6            <menu-navigation/>
7          </v-flex>
8        </v-layout>
9        <v-content>
10         <v-layout row wrap>
11           <v-flex xs12 sm12 offset-sm0 lg10 xl8 offset-xl2 offset-lg1 class="pt-0
                 pb-5">
12             <router-view/>
13           </v-flex>
14         </v-layout>
15       </v-content>
16       <v-layout row wrap>
17         <v-flex sm12 offset-sm0 lg10 offset-lg1 class="pa-0 pt-1">
18           <foot/>
19         </v-flex>
20       </v-layout>
21     </v-container>
22   </v-app>
23 </template>
```

Ukázka kódu 3.12: Výsledné základní rozvržení komponent aplikace – soubor App.vue

[6]HTTP Basic access authentication je metoda autorizace HTTP požadavku, kde klient posílá serveru v hlavičce HTTP autorizační údaje[30].

3.3.1 Menu aplikace

Hlavní menu aplikace je tvořeno komponentou `MenuNavigation` (v HTML `<menu-navigation/>`), která implementuje navigaci aplikace. Toto menu je horizontálního stylu a nachází se v horní části stránky. Na obrázku 3.6 je detail této komponenty zobrazen. Na tomto detailu je vidět, že navigace obsahuje



Obrázek 3.6: Komponenta hlavního menu aplikace

odkazy na všechny hlavní sekce, které byly definované v případech užití v sekci 1.3, a také odkaz na stránky zadavatele: *uvtnet.cz*. Menu dále také obsahuje podnavigaci pro zobrazení profilu uživatele a změnu jazyka aplikace. Tato podnavigace se zobrazí při najetí kurzoru myši na odkaz se jménem uživatele.

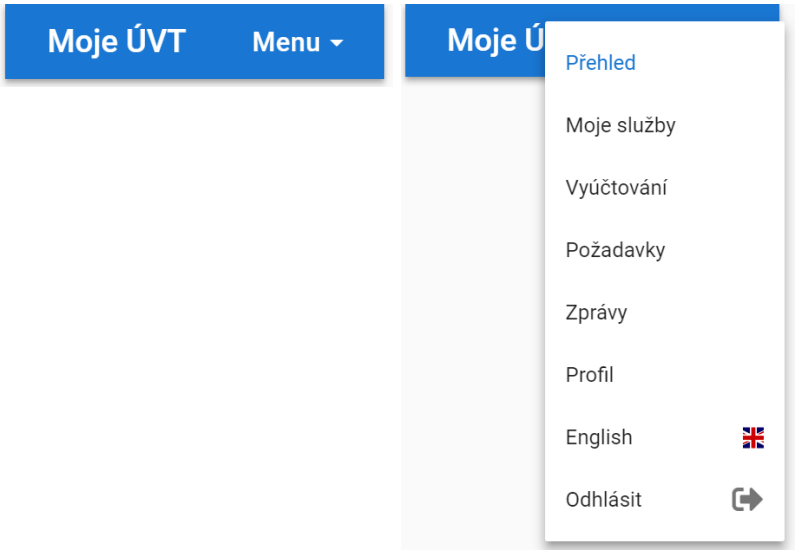
Jednotlivé odkazy na sekce také mohou obsahovat ikonu s upozorněním, jak je možné vidět na obrázku 3.8 u tlačítka *Zobrazit detail*. Toto upozornění se může týkat například nezaplaceného vyúčtování, nové informační zprávy či možnosti aktivace služby. Aby bylo možné jednoduše zjistit, kde se zákazník v aplikaci právě nachází, je odkaz na aktuálně zobrazenou sekci odlišen od ostatních sekcí jiným odstínem modré barvy.

3.3.2 Přizpůsobivost zobrazení

Pro splnění požadavku **NP.1** – přístupnost aplikace na většině zařízení, je nutné počítat s různými velikostmi displejů zařízení uživatelů. Framework Vuetify tento problém sám o sobě výborně řeší svým systémem mřížky, jak bylo vysvětleno v sekci 3.1.6.1. Spolu s dalšími atributy Vuetify je možné definovat složitější chování pro různá zobrazení.

V ukázce 3.12 zachycující rozvržení aplikace jsou použity atributy jako `xs12` a `offset-lg1`, které definují velikost a offset komponent na různých velikostech displejů. Dalšími atributy použitými uvnitř komponenty `MenuNavigation` je `hidden-sm-and-down` / `hidden-md-and-up`, které danou komponentu skryje na displejích `sm` (malé a střední tablety) a menších, respektive `md` (větší tablety a laptopy) a větších. Touto funkcionalitou je vyřešena změna hlavního menu aplikace z horizontálního menu na rozbalovací vertikální menu, které je nutné pro zachování přístupnosti aplikace například na mobilních displejích. Malé menu se zobrazí zákazníkům se zařízeními velikosti `sm` a menší. Toto menu pro menší displeje je zobrazeno na obrázku 3.7.

Většina komponent aplikace je navíc tvořena z menších komponent Vuetify nazývaných *karty*. Karty jsou v podstatě kontejnery pro další prvky a umožňují jednoduché přeskupení obsahu stránky při zvětšení či zmenšení velikosti displeje. Karty jsou většinou skládané vedle sebe do řady a případně pokračují na novém řádku, pokud jich je mnoho. Při zmenšení displeje se pak délka řádku zmenšuje, až je celý obsah stránky zarovnán do jednoho sloupce, což umožňuje snadné prohlížení stránky na mobilním telefonu.



(a) Zavřené menu (b) Otevřené menu

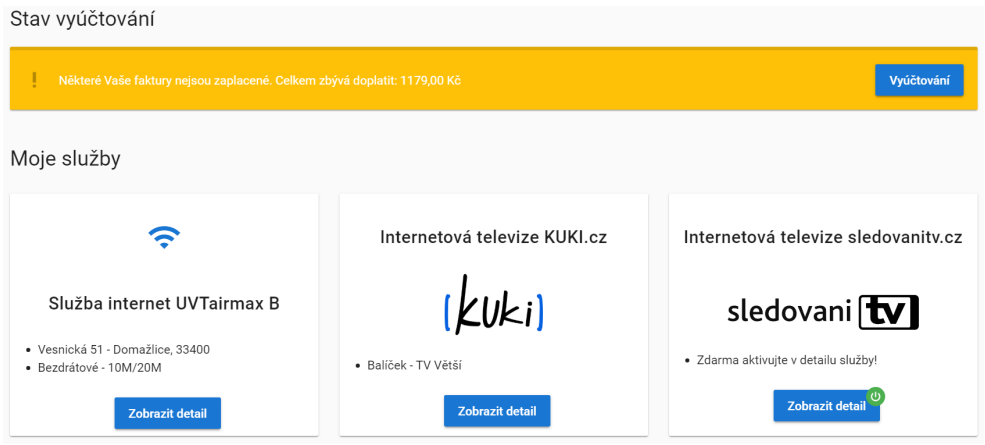
Obrázek 3.7: Komponenta hlavního menu pro malé displeje

3.3.3 Obsah stránky

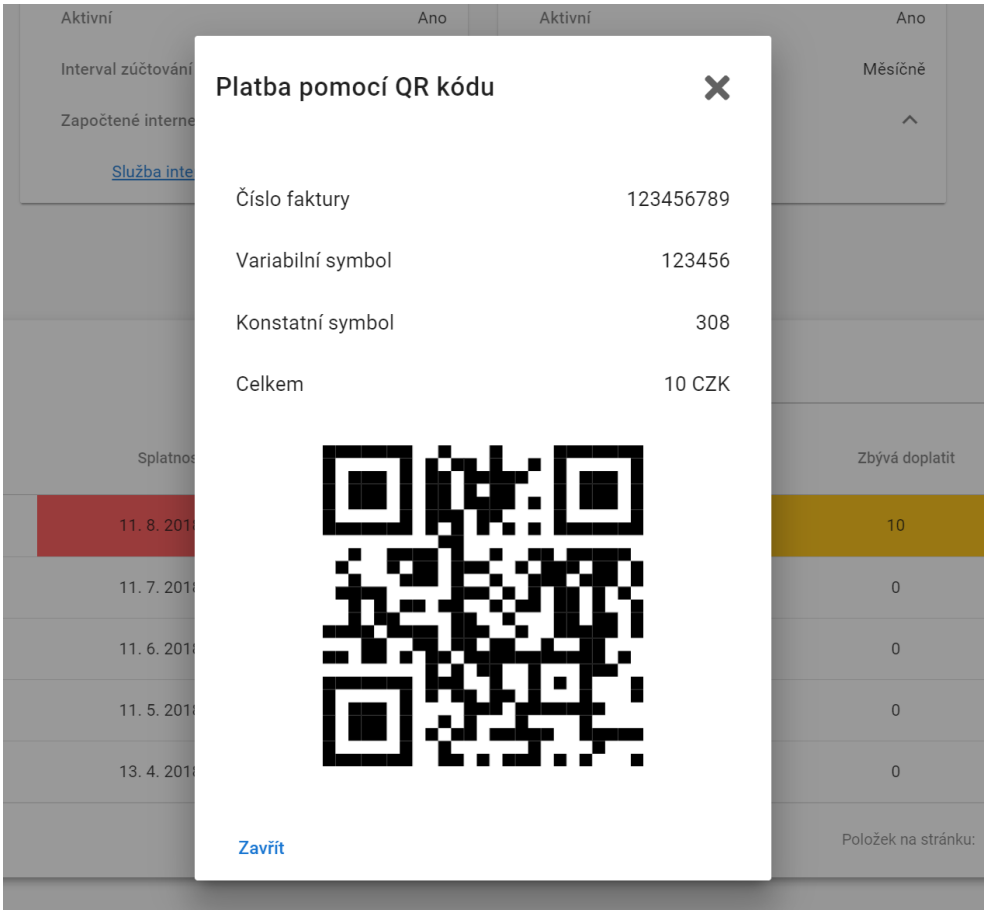
Obsah stránky se na rozdíl od menu liší v závislosti na tom, kde se uživatel v aplikaci aktuálně nachází. Komponenta `RouterView` vybere podle mechanizmu popsaném v sekci 3.1.4.2 komponentu, která se má zobrazit.

3.3.3.1 Přehled

Přehled zobrazuje nejdůležitější informace, jako aktuální stav vyúčtování a aktivní služby. Na obrázku 3.8 je tato komponenta zobrazena. V horní části je stav vyúčtování, který informuje uživatele, zda jsou všechna jeho vyúčtování zaplacena. V případě, že jsou vyúčtování zaplacena, je tato sekce znázorněna barvou zelenou. V opačném případě je tato sekce zvýrazněna barvou oranžovou, respektive červenou, pokud je navíc nějaká faktura po splatnosti. Pod stavem vyúčtování je zobrazen seznam služeb zákazníka v kartách se stručnými informacemi.



Obrázek 3.8: Komponenta sekce přehledu aplikace



Obrázek 3.9: Komponenta dialogu pro QR mobilní platbu

3.3.3.2 Moje služby

Sekce *Moje služby* obsahuje stejné dlaždice jako zobrazené služby v sekci *Přehled*. Jediná změna je, že výpis těchto služeb je roztříděn podle druhu služby. Takové zobrazení je praktické hlavně pro zákazníka s mnoha službami.

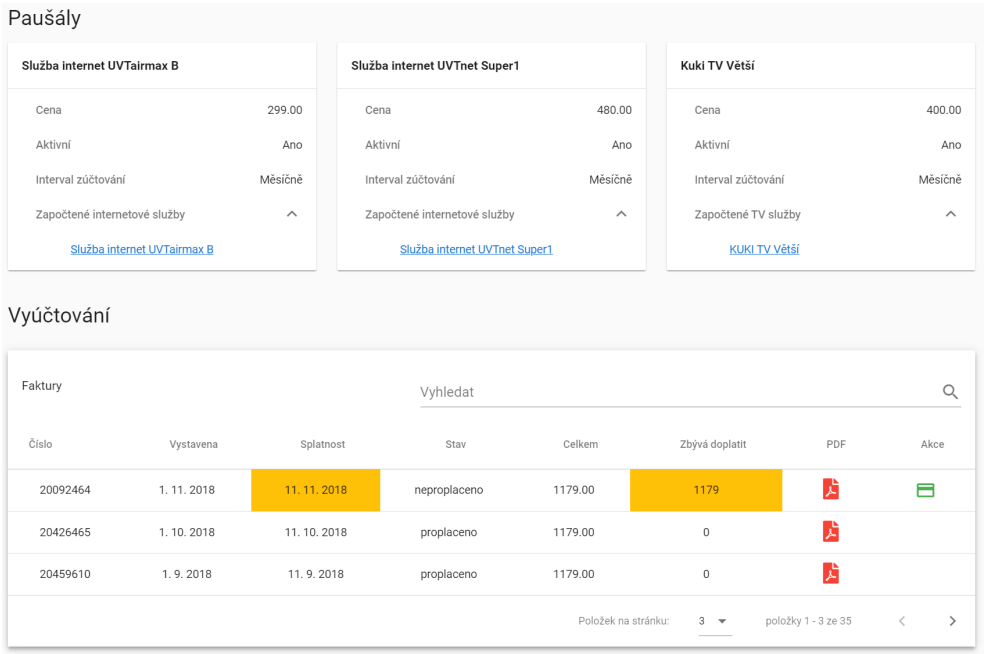
3.3.3.3 Vyúčtování

Na obrázku 3.10 je zobrazena komponenta sekce vyúčtování. V horní části je seznam paušálů, které jsou podkladem pro pravidelné vyúčtování. Pod tímto seznamem je tabulka s jednotlivými fakturami, kde si pro každé vyúčtování může uživatel stáhnout PDF. Pokud není vyúčtování zaplacené, tak má možnost kliknutím na zelenou ikonu platební karty otevřít dialog s QR kódem pro mobilní platbu. Nezaplacená vyúčtování mají zároveň barevně zvýrazněna políčka se splatností a částkou stejným způsobem jako v sekci Přehled. Zvýraznění je vidět na obrázku 3.10.

Dialog mobilní platby pomocí QR kódu je ukázán na obrázku 3.9. Informace, jako jsou částka, číslo účtu, variabilní symbol a měna, jsou zakódovány do řetězce SPAYD (Short Payment Descriptor). Tento řetězec by například pro platbu 10 Kč se zprávou *UKAZKA* na účet *CZ1234000000001234567890* vypadal následovně:

`SPD*1.0*ACC:CZ1234000000001234567890*AM:10.00*CC:CZK*MSG:UKAZKA`

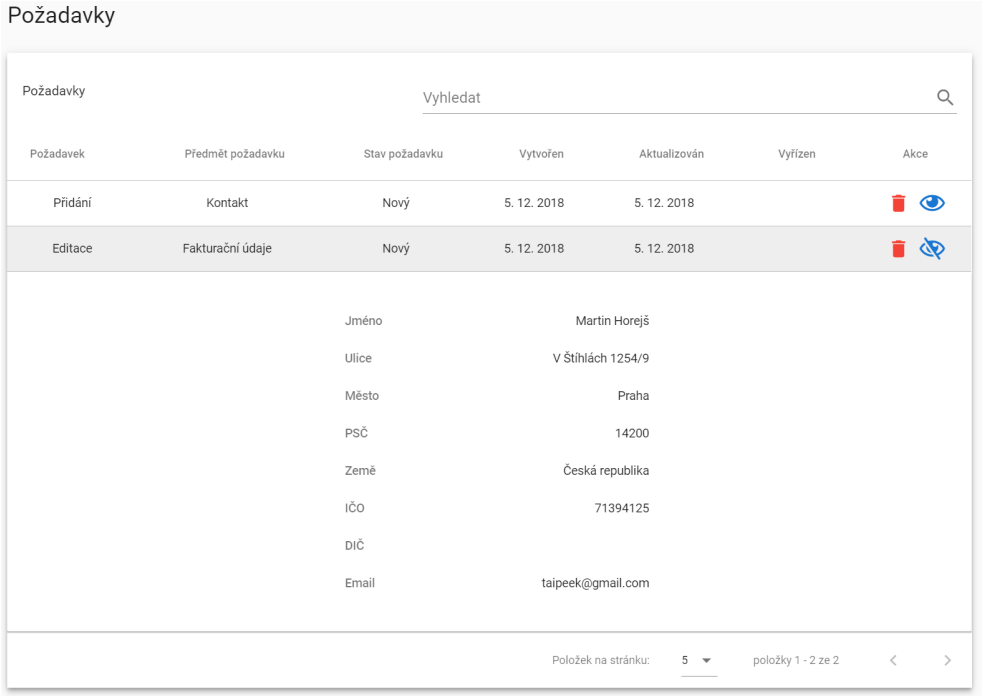
Takovýto řetězec je převeden do QR kódu a po načtení aplikací pro mobilní platby je platba připravena k odeslání.



Obrázek 3.10: Komponenta sekce vyúčtování

3.3.3.4 Požadavky

Sekce požadavků obsahuje podobnou tabulku jako sekce Vyúčtování. Řádky této tabulky jsou tvořeny požadavky uživatele s datem vytvoření, poslední aktualizací a uzavřením. Detail jednotlivých požadavků je možné zobrazit kliknutím na řádek nebo na ikonu ve sloupci *Akce*, kdy se řádek tabulky vertikálně roztáhne a zobrazí detail požadavku. Expandování detailu požadavku je vidět na obrázku této komponenty 3.11.



Obrázek 3.11: Komponenta sekce požadavků

3.3.3.5 Zprávy

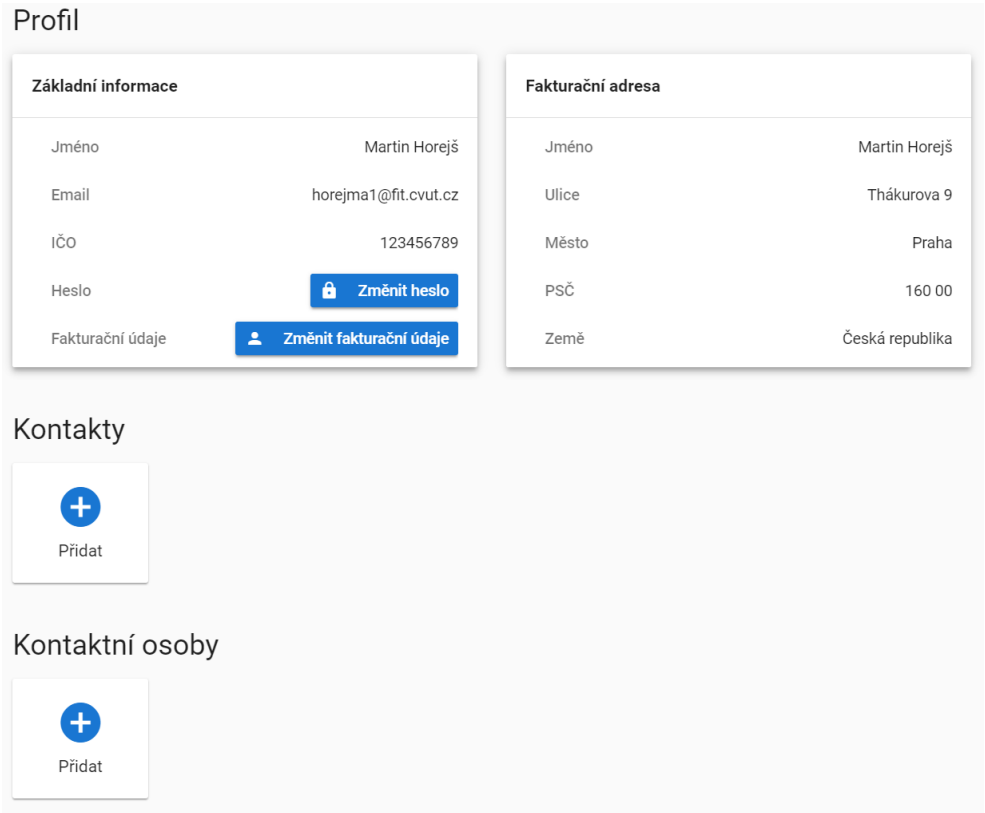
Sekce zprávy je také tvořena podobnou tabulkou jako v sekci vyúčtování, ale obsahuje informační zprávy poslané uživateli. Tyto zprávy jsou modře zvýrazněné, pokud nejsou přečteny. Zobrazení celé zprávy funguje stejně jako u tabulky v sekci požadavky, tedy kliknutím na řádek dané zprávy.

3.3.3.6 Profil uživatele

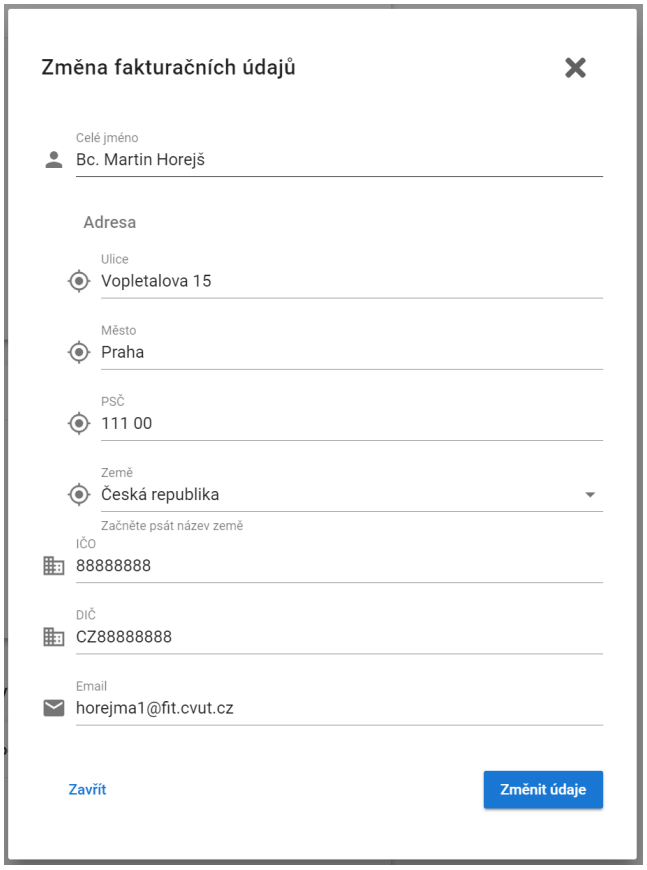
Sekce profilu uživatele obsahuje informace o uživateli. Na obrázku 3.12 je ukázka této sekce zobrazena. Sekce tvořena zejména fakturačními údaji o uživateli, jako jsou: jméno, adresa, IČO, variabilní symbol, email. Stránka profilu uživatele zobrazuje také další dvě podsekce, a to *Kontakty* a *Kontaktní osoby*.

Kontakty uživatele mohou být různých typů: telefon, další adresa, email, bankovní účet a webová stránka. U kontaktní osoby se definuje jméno, telefon a email. Obě tyto entity z doménového modelu pak lze editovat a přidávat. Pro přidání uživatel klikne na tlačítko přidat, které otevře dialog s formulářem, který po vyplnění odešle.

Další akce, které může uživatel provést v sekci Profil, jsou změna hesla a změna fakturačních údajů. Při kliknutí na tlačítka těchto akcí je otevřen dialog s odpovídajícím formulářem pro změnu. Dialog s formulářem pro změnu fakturačních údajů je vidět na obrázku 3.13.



Obrázek 3.12: Komponenta profilu uživatele

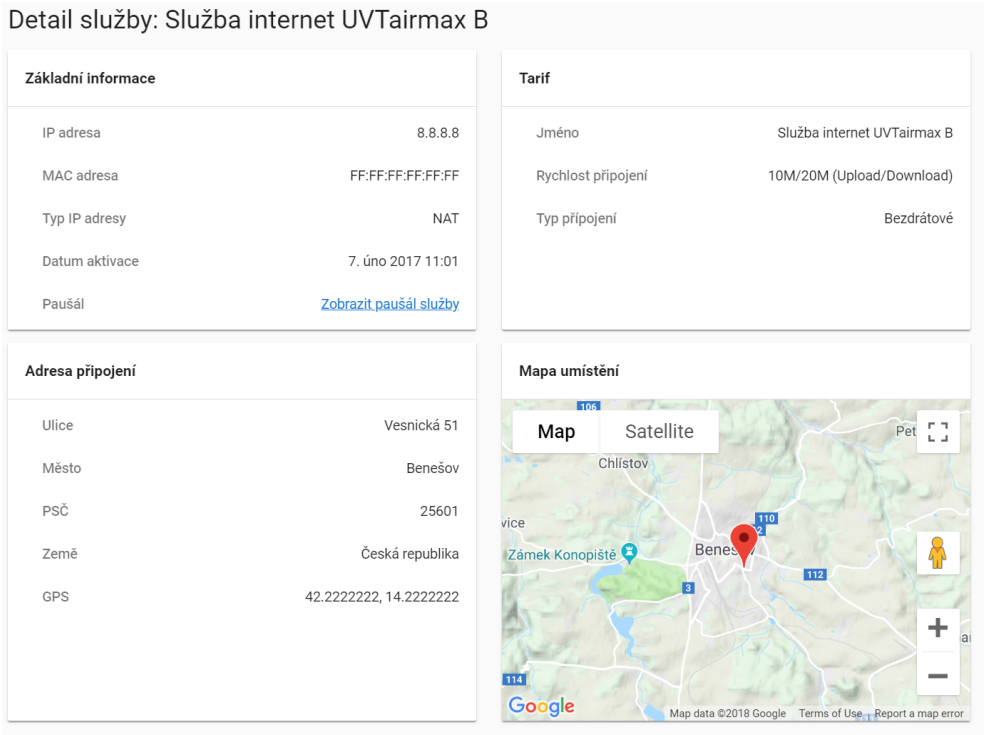


Obrázek 3.13: Dialog s formulářem pro změnu fakturačních údajů zákazníka

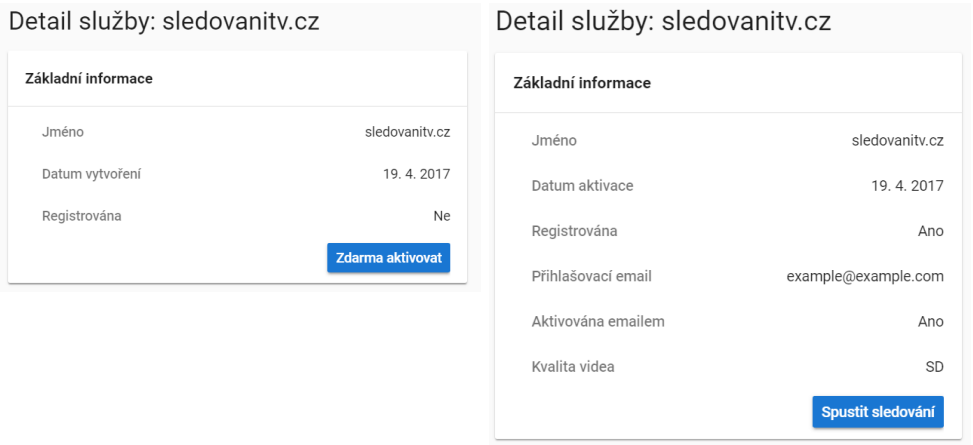
3.3.3.7 Detail služby

Na obrázku 3.14 je zobrazena komponenta detailu služby. Tato sekce obsahuje 4 karty zobrazující základní informace, tarif služby, adresu připojení a mapu s umístěním adresy připojení. Tento obrázek zachycuje detail internetové služby. V aplikaci je možné zobrazit také služby televizní a služby hlasové.

Na obrázku 3.15 je zobrazen detail televizní služby poskytovatele sledovanitv.cz, který nabízí internetovou televizi ve spolupráci se zadavatelem ÚVT Internet. Zákazník musí nejprve službu aktivovat u poskytovatele (zachycuje obrázek 3.15a), což může udělat kliknutím na tlačítko *Zdarma aktivovat*, kdy je přesměrován k poskytovateli této služby a tam provede dokončení registrace. Po tomto kroku je možné službu plně využívat (zachycuje obrázek 3.15b) a je možné zahájit sledování televize z aplikace tlačítkem *Spustit sledování*.



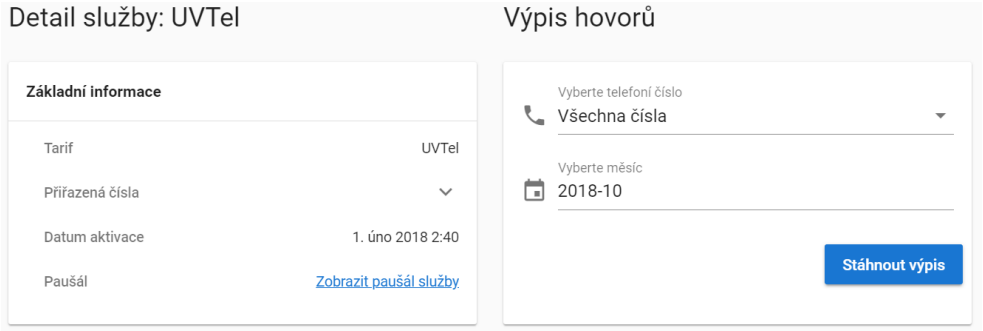
Obrázek 3.14: Komponenta detailu internetové služby



(a) Před aktivací (b) Po aktivaci

Obrázek 3.15: Komponenta detailu televizní služby

Na obrázku 3.16 je zobrazena komponenta detailu hlasové služby. Tato komponenta obsahuje jednu kartu s informacemi o službě a další kartu s formulářem pro stažení měsíčního výpisu volaní. K dispozici jsou hromadné výpisy pro všechna čísla nebo samostatné výpisy pro jednotlivá čísla.



Obrázek 3.16: Komponenta detailu hlasové služby

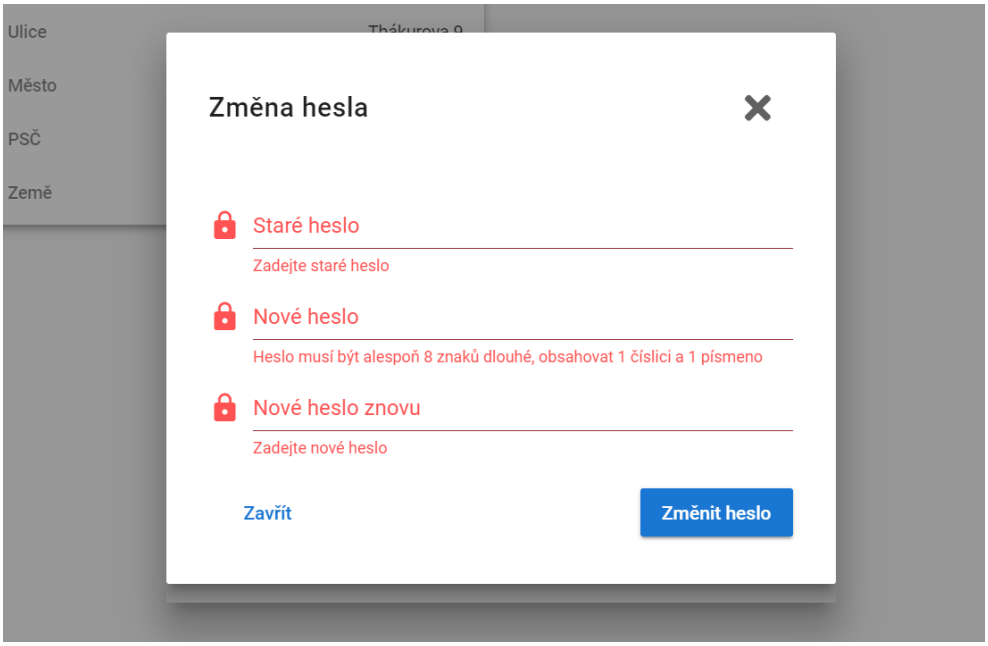
3.3.4 Zápatí stránky

Zápatí stránky je tvořeno komponentou `Foot`, která obsahuje odkazy jako jsou kontakt, změna jazyka a formulář pro zpětnou vazbu. Tato komponenta je zobrazena na obrázku 3.17. Zápatí je na stránce přítomné vždy, tedy i na přihlašovací stránce.

Obrázek 3.17: Komponenta zápatí aplikace

3.3.5 Dialogy pro formuláře a akce

V sekci 2.3.1.2 je implementace formulářů a akcí uživatele navrhnuta jako zobrazovací dialog. Na obrázku 3.18 je vidět takový dialog s formulářem pro změnu hesla. Formulář obsahuje políčka pro vyplnění, která se ihned validují a poskytnou případnou nápovědu, jakou formou musí být vyplněna. Dialogy obsahují tlačítko křížek v pravém horním rohu pro jejich zavření. Při odeslání těchto formulářů je zobrazen indikátor probíhajícího požadavku jako modrá horizontální čára a při dokončení je zobrazena hláška informující uživatele o úspěchu či neúspěchu. Hláška je podle toho zvýrazněna zeleně, respektive červeně.



Obrázek 3.18: Komponenta dialogu pro změnu hesla

3.3.6 Data a komunikace s backendem

Data potřebná pro práci frontendu jsou uložena pomocí nástroje *Vuex*, což je nástroj, který řídí stav aplikace Vue. Vue samotné spoléhá na propagaci dat z rodičovských komponent do potomků, ale tento způsob nemusí být vždy vhodný, například pokud více komponent potřebuje stejná data. *Vuex* se zaměřuje právě na tento problém a v podstatě řeší vrstvu modelu ve frontendu aplikace. V této vrstvě je zároveň naimplementovaná komunikace s backendem, která je zprostředkovaná HTTP knihovnou *vue-resource*.

V ukázce 3.13 je zobrazena implementace entity *Faktura* v knihovně *Vuex*. Tato knihovna definuje výchozí stav entity, mutace pro změnu stavu a akce s touto entitou. Výchozí stav entity je nastaven jako prázdné pole. Jediná mutace této entity provádí přiřazení nového pole a akce řeší komunikaci s backendem a volání odpovídající mutaci stavu. Jakákoliv změna stavu entity by měla být provedena pomocí mutace, což zajistí, že stav bude omezen pouze na několik forem, což umožní jednodušší řešení potíží při implementaci. Takto definovaný stav entity ve *Vuex* pak lze injektovat do libovolné komponenty.

3.3.7 Struktura projektu

Jelikož je frontend vyvíjen odděleně od backendu má projekt svoji vlastní adresářovou strukturu. Na obrázku 3.19 je zobrazena struktura s nejdůležitějšími body aplikace, které byly diskutovány v celé sekci 3.3.

```
1  import Vue from 'vue'
2  export default {
3    namespaced: true,
4    state: {
5      invoices: []
6    },
7    mutations: {
8      setInvoices (state, data) {
9        state.invoices = data
10     }
11   },
12   actions: {
13     getInvoices (context) {
14       return new Promise((resolve, reject) => {
15         Vue.http
16         .get('/api/invoice')
17         .then(({body}) => {
18           context.commit('setInvoices', body)
19           resolve()
20         })
21         .catch(error => reject(error))
22       })
23     },
24     resetInvoices (context) {
25       context.commit('setInvoices', [])
26     }
27   }
28 }
```

Ukázka kódu 3.13: Ukázka implementace entity *Faktura* pomocí knihovny *Vuex*

3.4 Backend aplikace

V sekci 3.1.3 byl pro implementaci backendu aplikace vybrán framework Sails.js. Pomocí akcí obsažených v kontrolerech je implementováno API, na které je navázaný frontend. Tyto akce většinou neimplementují přímo samotnou business logiku aplikace, ale přeposílají požadavky do integrovaného systému Opentis, kde je tato logika umístěna. Mimo to backend řeší další aspekty aplikace, jako autorizace, management uživatelských účtů, zabezpečení a jiné, které jsou diskutovány v sekcích níže.

Struktura projektu backendu Sails.js je prakticky stejná jako ta na obrázku 3.1, a proto zde nebude znovu uvedena. Implementace obsahuje navíc další kontrolery, akce a pomocné služby. Finální struktura je v podstatě vidět v ukázce Pokrytí kódu testováním 4.2.

```
README.md ..... popis a informace o tomto projektu
vue.config.js ..... konfigurace sestavení a vývoje aplikace
package.json ..... soubor s definicemi závislostí
src ..... zdrojové kódy implementace
    main.js ..... vstupní bod aplikace
    App.vue ..... kořenová komponenta
    assets ..... adresář se statickým obsahem
    components ..... složka s komponentami
        Billing.vue
        Dashboard.vue
        Foot.vue
        LoginForm.vue
        ...
    locales ..... lokalizační slovníky aplikace
        ActionLocale.js
        BillingLocale.js
        ...
    mixins ..... pomocné funkcionality pro komponenty
    plugins
        vuetify.js ..... nastavení zakomponování frameworku Vuetify.js
    router
        router.js ..... definice URL cest aplikace
    state ..... implementace stavu aplikace
tests ..... testy
    e2e
        specs ..... definice e2e testů
```

Obrázek 3.19: Struktura frontendu aplikace

3.4.1 Autorizace uživatelů

Klasické přihlášení do webových aplikací bývá implementováno pomocí ukládání stavu relace mezi uživatelem a aplikací, který je označován jako *session*. Tato metoda vyžaduje perzistentní ukládání těchto dat na straně serveru.

Autorizace a proces přihlášení uživatelů do nové aplikace jsou implementovány pomocí jiné technologie – *JSON Web Token*[31]. Tato technologie nevyžaduje žádné úložiště a funguje na principu kryptografie. Místo udržování session je při úspěšném ověření hesla uživatele vystaven *JWT* token, který je tvořen třemi částmi: hlavičkou, tělem a podpisem. Hlavička určuje typ a algoritmus podpisu, tělo může obsahovat libovolná data ve formátu JSON. Hlavička a tělo jsou podepsány vybraným kryptografickým algoritmem pro podpis. Tento podpis je připojen na konec tokenu, jehož všechny části jsou

nakonec zakódovány Base64[7] kódováním. Finální tvar řetězce tokenu má formát `havička.tělo.podpis`. Podepsání je umožněno díky klíči, který je v držení backendu aplikace. Tímto klíčem jsou také podpisy tokenů zpět ověřovány, což umožňuje autorizaci požadavků uživatele, kdy při každém požadavku uživatele je přidána hlavička HTTP obsahující autorizační token, který je na serveru ověřen. Tělo tokenu obsahuje identifikátor uživatele a další data, jako je platnost tokenu, které backend může použít pro další funkcionalitu.

3.4.2 Management uživatelských účtů

Aplikace je integrována se systémem Opentis, který obsahuje všechny údaje o uživateli. Nicméně přihlašovací údaje uživatelů jsou udržovány přímo v databázi aplikace, a proto musí existovat způsob, jak tato data řídit ze systému Opentis, z důvodů popsaných v následující sekci 3.4.2.1. Backend obsahuje kontroler a akce přímo pro tento účel. Tyto akce vyžadují speciální autorizaci ze strany Opentisu, která je opět založena na technologii *JWT*. Mimo akcí tvorby a smazání uživatelských účtů podporuje backend akci krátkodobého přihlášení do uživatelského účtu autorizovaným zaměstnancem ÚVT Internet, což umožňuje přímé řešení potíží v aplikaci přímo se zákazníkem. Tato akce je z bezpečnostních důvodů monitorována a historie těchto přihlášení zaměstnance je uchovávána.

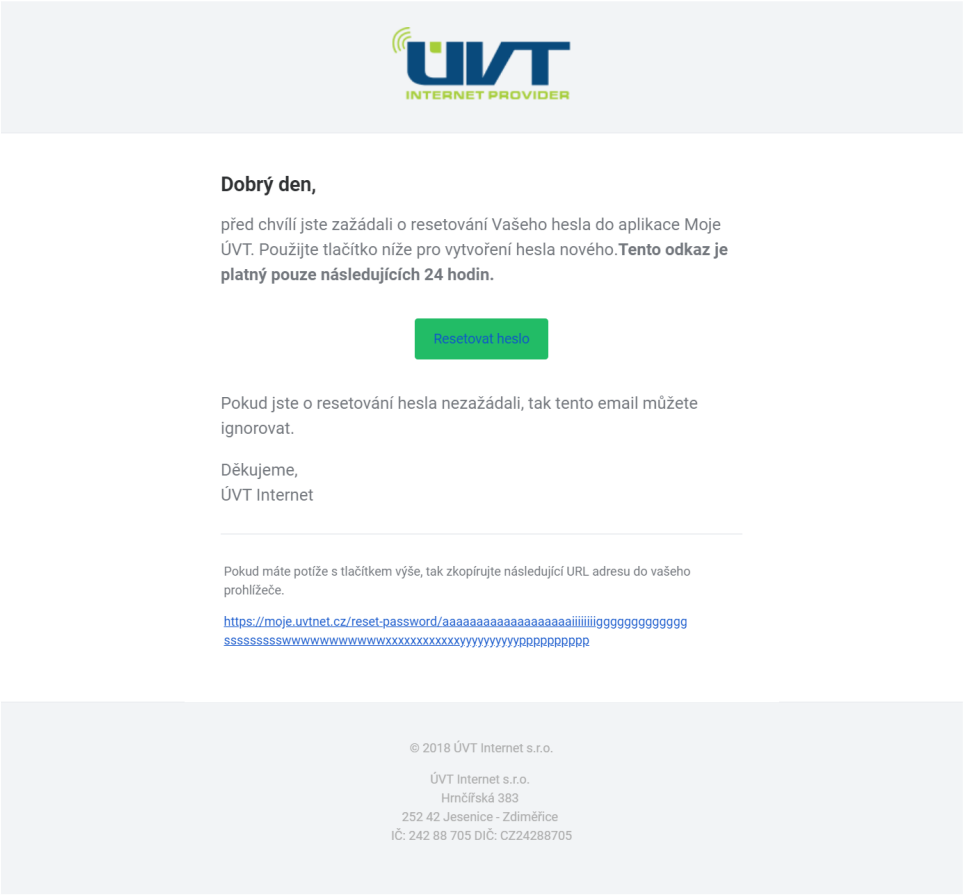
3.4.2.1 Proces vytvoření uživatelského účtu

Záznam entity *Zákazník* je vytvořen v systému Opentis zpravidla po podpisu smlouvy o poskytnutí služeb. Poté je ze systému Opentis zavoláno API backendu aplikace pro tvorbu nového uživatelského účtu. Při tvorbě účtu je uložen do databáze aplikace záznam a uživateli je zaslán aktivační email. Email obsahuje odkaz se speciálním *JWT* tokenem, který odkazuje na tvorbu nového hesla pro tento účet, po jehož tvorbě se může uživatel přihlásit do aplikace.

3.4.3 Zasílání emailů

Několik akcí, jako jsou resetování hesla (obrázek 3.20) a aktivace nového účtu, vyžaduje zasílání emailů uživatelům. Zasílání emailů je řešeno přes emailový účet *moje@uvt.cz*, ke kterému se backend připojí přes protokol SMTP. Tělo zprávy je ve formátu HTML, které je vygenerováno ze šablony na straně backendu pomocí funkce `sails.renderView('templateName', templateData)`, kde *templateName* je jméno šablony a *templateData* obsahuje data pro správné vyplnění šablony s odpovídající jazykovou lokalizací.

[7] Base64 je kódování dat, kdy jsou data zakódována po šesti bitech. Hodnoty mezi 0 až 63 jsou pak kódovány do abecedy malých a velkých písmen spolu s číslicemi a speciálními znaky: `+`, `/` a výplňovým znakem `=` [32].



Obrázek 3.20: Ukázkový email pro resetování zapomenutého hesla

3.4.4 Integrace dat

V ukázce kódu 3.14 je většina implementace pomocného modulu pro volání API systému Opentis. Volání probíhá pomocí HTTP knihovny *request* pro Node.js. Z ukázky je vidět, že volání probíhá podle schématu návrhu tohoto API v sekci 2.4.4.

3.4.5 Zabezpečení

Pro splnění nefunkčních požadavků **NP.2 – Zabezpečení aplikace proti neoprávněnému vniknutí** a **NP.3 – Zabezpečení proti úniku osobních údajů** musejí být do aplikace implementovány odpovídající zabezpečovací techniky.

Použití protokolu HTTPS je jednou z nejdůležitějších technik zabezpečení webové aplikace. Tento protokol šifruje veškerou komunikaci mezi webovým serverem a uživatelem, a tím zamezuje odposlechu citlivých informací.

```
const rp = require('request-promise-native')
const r = require('request')
module.exports = {
  url: sails.config.custom.opentisApiURL,
  apiKey: fs.readFileSync(path.resolve(__dirname, '...')).toString(),
  request (req, resource, method, params, syncRequest) {
    let options = {
      method: 'POST',
      headers: {'API-TOKEN': this.apiKey},
      body: {method: method, params: params, id_customer: params.
          id_customer || null, meta: {...}},
      json: true,
      resolveWithFullResponse: true
    }
    let u~= new URL(this.url)
    u.search = `r=${resource}`
    return syncRequest === true ? r(u, options) : rp(u, options)
  }
}
```

Ukázka kódu 3.14: Ukázka implementace OpentisAPI.js

Aplikace proto používá tento protokol s certifikátem ověřeným u certifikační autority Let's Encrypt[33].

Server webové aplikace je zároveň chráněn striktním firewallem a jeho databáze je chráněná heslem. Uživatelská hesla jsou hashována silným *bcrypt*[34] algoritmem s vysokým počtem kol opakování, který zaručuje časově neúnosné prolamování těchto hesel při hypotetickém úniku dat z databáze.

V databázi aplikace není uloženo mnoho uživatelských údajů. Je to v podstatě pouze ID uživatele a hash hesla, kde zbytek informací je uložen v databázi systému Opentis. Integrace s Opentisem je také zabezpečena protokolem HTTPS a také speciální autorizací diskutovanou v sekci 3.2.3.

Další důležitou technikou zabezpečení je validace příchozích dat od uživatele, například při odesílání formulářů. Backend aplikace se nemůže spoléhat na to, že frontend sám validuje formuláře, protože útočník tuto validaci může jednoduše obejít a odeslat data přímo na backend. Validace formulářů frontendu je spíše pro zlepšení přístupnosti aplikace, než pro zvýšení zabezpečení, a proto je nutné validovat data na backendu aplikace a zamítnout požadavky, které neodpovídají povoleným formátům.

Další nutností je ochrana proti SQL injection, kdy se útočník pokouší manipulovat s dotazy do databáze. Tuto hrozbu lze mitigovat pomocí předpřipravených parametrizovaných dotazů nebo použitím funkcí pro *escapování*[8]

[8]Escapování proměnné identifikuje znaky, které systém považuje za klíčové a mají speciální význam. Takové znaky jsou například `"`, `'`, `;`, `#`. Těmto znakům je předřazen escapovací znak (většinou to bývá `\`) a tím přicházejí o své systémové vlastnosti a stávají se součástí řetězce.

proměnných, které jsou vkládané do řetězce dotazu. Aplikace využívá obě tyto techniky.

3.5 Prostředí pro vývoj aplikace a její standardy

Vývojové a testovací prostředí spolu s verzovacím systémem kódu hraje důležitou roli při implementaci aplikace a zaručuje vyšší kvalitu dodávaného softwaru. Další aspekty jako dodržování formátování a struktury kódu či použití standardů formátu také přispívají k vyšší kvalitě a výrazně zjednodušují pozdější spolupráci dalších osob.

3.5.1 Vývojové prostředí

Veškerý kód související s touto prací byl napsaný v editoru Visual Studio Code, který nabízí podporu mnoha jazyků prostřednictvím pluginů, které obsahují i další funkcionality, jako je napovídání a kontrola syntaxe kódu. S pomocí tohoto nástroje se rychlost implementace nové aplikace značně zrychlila v porovnání s klasickým editorem bez těchto funkcionalit a pomůcek.

3.5.2 Standard formátu kódu

Pro jazyk JavaScript existuje několik standardů formátu kódu, které se zabývají formátováním a strukturou kódu. Jak pro backend tak pro frontend byl použit standard StandardJS, který například definuje kritéria odsazení textu dvěma mezerami, neužíváním středníků, jednoduchých uvozovek pro psaní textových řetězců a další[35].

Aby byl standard dodržován, tak je v celém projektu kontrolován zdrojový kód nástrojem ESLint, což je open-source nástroj pro kontrolu JavaScript kódu, který leze nainstalovat s pomocí nástroje NPM: `npm install esling -g`. Soubory typu *.js* lze poté zkontrolovat příkazem `eslint example.js` [36] nebo lze opět využít pluginu ESlint pro Visual Studio Code.

Dodržování standardu formátu zdrojového kódu zlepšuje jeho čitelnost i přehlednost a může předejít chybám vzniklým například špatným uzávorkováním.

3.5.3 Verzovací systém Git

Používání verzovacího systému je jedním z nejdůležitějších procesů při vývoji softwaru. Zadavatel používá jako verzovací systém platfomu GitLab, která je postavena na verzovacím nástroji Git.

Veškeré změny a implementace dílčích částí aplikace byly postupně verzované a jsou tedy zpětně k dispozici pro porovnání změn mezi verzemi či nutného vrácení kódu na starší verzi kvůli změně, která způsobila nesprávné chování aplikace. Kód je také zálohován, a tím se předchází možné ztrátě, která

by způsobila finanční ztrátu zadavatele (případně vykonavatele v závislosti na jejich smluvním vztahu). Další výhodou je možný vývoj na více počítačích, bez nutnosti složitého kopírování kódu.

3.5.4 Testovací prostředí

Zadavatel provozuje v souběhu hlavní a vývojovou verzi jejich systému Opentis. Pro otestování reálné integrace mezi systémem Opentis a nové aplikace bylo vytvořeno i testovací prostředí pro novou aplikaci, která se co nejvíce podobá reálnému prostředí nasazení. Toto prostředí bylo zintegrované s testovací verzí systému Opentis, což umožnilo plynulý vývoj s průběžným testováním, a tím zajistilo lepší testování integrace a zvýšilo i kvalitu implementace.

3.6 Výsledek implementace

Implementace proběhla bez větších problémů a jejím výstupem jsou tři části: API systému Opentis, frontend a backend aplikace. Během vývoje byla implementace průběžně testována, což je diskutované v kapitole 4. Po prvním nasazení aplikace proběhlo přidání další funkcionality a menších úprav na základě požadavků a připomínek zadavatele.

3.6.1 Další vývoj

Jednou z plánovaných funkcionalit aplikace byla možnost změny tarifu u odebíraných služeb zákazníka. Takový požadavek závisí na automatizaci nabídky tarifů pro danou službu, která by se odrážela například od adresy umístění takové služby. Bohužel toto mapování z dané služby na množinu tarifů ještě není k dispozici v systému Opentis. Jakmile bude toto mapování hotové, tak bude tato funkcionalita prioritou pro další implementaci.

Kapitola 4

Testování

Cílem testování aplikace je odhalit chyby a nedostatky, které snižují její kvalitu. Lze tak předejít mnoha problémům při nasazení a údržbě aplikace. Aplikace byla podrobena několika typům testování, které se zaměřovaly na různé aspekty.

4.1 Uživatelské testování

Uživatelské testování (Usability testing) se zaměřuje na přístupnost a použitelnost aplikace a jejího UI. Testování probíhá následovně: testovací uživatel dostane scénář s pokyny, jaké akce má vykonat, zatímco dozor uživatele pozoruje a dělá poznámky o průběhu testování. Během celého procesu je také snímána obrazovka aplikace pro další možnou analýzu.

Testování proběhlo na stolních počítačích s obrazovkami s rozlišením 1920× 1080 pixelů.

4.1.1 Scénář

U uživatelského testování je důležité, aby testovací scénář neobsahoval příliš detailních informací pro testovacího uživatele. Mohlo by pak dojít k tomu, že aplikace nebude dostatečně otestována, jelikož uživatel by slepě plnil detailní instrukce místo toho, aby ho uživatelské rozhraní a jeho intuice dovedla ke správné akci.

Obvyklou praxí je vytvořit testovací persony s různými scénáři. V systému figuruje pouze jedna role – zákazník. Ta zde byla rozdělena do dvou person. Jedna persona zastupuje typického zákazníka s internetovou a telefonní službou, zatímco druhá persona je zákazník, jakožto firma, s mnoha internetovými a hlasovými službami.

Následující testovací scénáře byly použity pro 6 různých testovacích uživatelů s jinými zkušenostmi práce s výpočetní technikou.

4.1.1.1 Persona zastupující fyzickou osobu – S1

1. Otevřete si webový prohlížeč a jděte na stránku `https://moje.uvtnet.cz/`
2. Přihlaste se do aplikace jako uživatel s klientským číslem *803260* a heslem *admin*.
3. Zobrazte si své vyúčtování a stáhněte si PDF faktury č. *17207776*.
4. Otevřete svůj profil a proveďte následující kroky.
   a) Změňte si heslo na *testheslo1*.
   b) Dále si změňte svůj fakturační email na *example@example.com* a ve fakturační adrese zemi na Českou republiku.
   c) Přidejte libovolnou kontaktní osobu.
   d) Přidejte libovolný kontakt.
5. Zkontrolujte, že všechny tyto požadavky byly zaznamenány v sekci *Požadavky* a požadavek na změnu fakturačních údajů zrušte.
6. Otevřete si sekci Moje služby a zobrazte si detail své televizní služby *sledovanitv.cz*.
7. Zobrazte si detail své druhé televizní služby *KUKI* a zjistěte své Kuki ID přihlašovací údaje.
8. Odhlaste se z aplikace.

4.1.1.2 Persona zastupující firmu – S2

1. Otevřete si webový prohlížeč a jděte na stránku `https://moje.uvtnet.cz/`
2. Změňte jazyk zobrazení na angličtinu.
3. Přihlaste se do aplikace jako uživatel s klientským číslem *107489* a heslem *admin*.
4. Změňte jazyk zpět na češtinu.
5. Stáhněte si dubnový výpis volání z čísla *227022022* z telefonní služby, pod kterou toto číslo spadá.
6. Stáhněte si březnový výpis volání z telefonní služby, která spadá pod volaný účet *testAccount*.
7. Odhlaste se z aplikace.

4.1.2 Výsledky uživatelského testování

Výsledky jsou shrnuté v tabulce 4.1 obsahující úspěšnost[9] testování. Ve výsledcích je uvedena nejen úspěšnost provedení instrukcí obou scénářů, ale také čas jejich provedení.

Jméno uživatele	Úspěšnost	Zkušenost uživatele	Čas (min:sec)	Problémové instrukce
Šárka	89%	vysoká	14:17	S1–4a, S2–5
Anna	74%	nižší	14:41	S1–4b, S1–4c, S1–4d, S2–5, S2–2
Martin	95%	střední	11:54	S2–2
Míša	89%	střední	13:22	S1–7, S2–5
Jan	95%	vyšší	10:09	S2–5
Radek	95%	vyšší	10:34	S2–5

Tabulka 4.1: Výsledky uživatelského testování za oba scénáře

Uživatelské testování dopadlo úspěšně. Testování proběhlo hned po dokončení prvního prototypu aplikace, a poskytlo tak nové pohledy na úpravy a další vývoj. Nejčetnějším problémem byl bod 5 v druhém scénáři, kdy komponenta dialogu pro výběr měsíce neuložila výsledek, pokud ji uživatel zavřel jiným způsobem něž tlačítkem pro potvrzení výběru. Většina testerů zde vybrala měsíc a dialog zavřela a tudíž měsíc nebyl správně vybrán. Tato chyba byla později opravena, aby k problémům již nedocházelo.

Dalším zlepšením na základě tohoto testování bylo přidání tlačítka křížku do pravého horního rohu modálního dialogu. V původním návrhu toto tlačítko pro uzavření dialogu nebylo zakomponováno. Toto vylepšení navrhla uživatelka Šárka, která měla se zavřením těchto dialogů potíže.

4.2 Jednotkové testy

Jednotkové testy se používají pro testování jednotek, což jsou izolované části kódu s určitou funkcionalitou. Tyto jednotky jsou testovány s předem danými vstupními parametry a mají definované chování a výstup pro každý z těchto vstupů. Tyto testy kontrolují správnost algoritmu a umožňují identifikovat možné chyby v chování aplikace[37].

Při vývoji je důležité tyto testy spouštět pokud dojde ke změně kódu, aby se ověřilo, že provedené úpravy nezpůsobily nežádoucí regrese v systému. V případě změny funkcionality či chování je nutné naopak upravit tyto testy. Spouštění těchto testů je zpravidla velmi rychlé, protože jsou jednotky tes-

[9] Úspěšnost je počítána jako počet kroků scénáře, se kterými tester neměl potíže/celkový počet kroků scénáře.

továny izolovaně od zbytku aplikace, což nebrzdí průběh testování přidanou režií[37].

Testování pomocí jednotkových testů bylo použito na testování služeb (services) aplikace. Tyto služby řeší zapouzdřenou logiku aplikace, jako je například manipulace s JSON Web Tokeny pro přihlašování či další autorizace aplikace.

4.3 Integrační testování

Cílem integračního testování je odhalit chyby v komunikaci a propojení určitých komponent systému. V tomto případě bylo testováno API backendu aplikace, které komunikuje se systémem Opentis, kde volá různé funkce. API backendu aplikace je implementováno pomocí skriptů zvaných *Controller*, které obsahují funkce, pro něž byly v rámci integračního testování vytvořeny samostatné testy.

Jelikož je komunikace těchto komponent přenášena HTTP protokolem, byl použit nástroj *supertest*[38], který umožňuje automatizaci a snadné psaní takovýchto testů. V ukázce kódu 4.1 je vypsán test pro *AppController*, který

```
1  const supertest = require('supertest')
2  const assert = require('assert')
3  const fs = require('fs')
4
5  describe('AppController', () => {
6    describe('response test', () => {
7      it('should get index.html', (done) => {
8        supertest(sails.hooks.http.app)
9        .get('/')
10       .expect(200, done)
11     })
12     it('should get index.html', (done) => {
13       supertest(sails.hooks.http.app)
14       .get('/anything-not-api-related')
15       .expect(res => {
16         const app = `${__dirname}/../../../assets/index.html`
17         let file = fs.readFileSync(app, {encoding: 'utf8'})
18         assert(file.length === res.text.length)
19       })
20       .expect(200, done)
21     })
22   })
23 })
```

Ukázka kódu 4.1: Příklad integračního testování s nástrojem *supertest*

kontroluje, zda funguje poskytování souboru *index.html*, který je zavaděčem frontendové části aplikace v uživatelově prohlížeči. Zároveň se v ukázce na řádku 18 kontroluje, zda délka tohoto souboru odpovídá délce originálního

souboru uloženého na disku, což ověřuje, že byl soubor přenesen k uživateli celý.

Pro každý *Controller* backendu existuje obdobný test, který testuje všechny jeho metody namapované na nějakou cestu URL.

4.4 Pokrytí kódu testováním

Při automatizovaném testování (Integračního a Unit testování) lze zjistit procento kódu, které je pokryté testy. Tato metrika vypovídá o tom, jak veliká část aplikace je otestovaná a zda jsou testy napsané pro všechny možné případy větvení kódu, které mohou nastat. Tuto metriku měří u testování nástroj *Istanbul* z balíčku *nyc*[39].

V ukázce kódu 4.2 je výstup běhu jednotkového a integračního testování (sekce 4.2 a 4.3). Oboje testování proběhlo v testovacím frameworku *Mocha*[40].

```
-------------------------------|----------|----------|----------|----------|
File                           |  % Stmts | % Branch |  % Funcs |  % Lines |
-------------------------------|----------|----------|----------|----------|
All files                      |    73.14 |    59.38 |    82.76 |    79.14 |
 api/controllers               |    66.56 |    56.03 |    79.31 |    73.41 |
    AppController.js           |      100 |      100 |      100 |      100 |
  CustomerMessageController.js |    74.25 |       50 |    83.33 |    80.23 |
  CustomerRequestController.js |    82.93 |       50 |      100 |    89.47 |
  InvoiceController.js         |    78.95 |       50 |    66.67 |    83.33 |
  LoginController.js           |    65.12 |    63.27 |    88.89 |    76.71 |
  ServiceController.js         |    82.61 |    81.82 |    66.67 |    90.48 |
  SledovaniTVController.js     |    91.67 |       50 |      100 |      100 |
  SubscriptionController.js    |    83.33 |       50 |      100 |    90.91 |
  UserController.js            |    61.25 |     47.5 |    83.33 |    64.79 |
 api/models                    |       85 |      100 |      100 |       85 |
  User.js                      |       85 |      100 |      100 |       85 |
 api/policies                  |    80.95 |       50 |    83.33 |    94.44 |
  OpentisAuthorization.js      |       75 |       50 |      100 |       90 |
  isAuthorized.js              |    81.25 |       50 |      100 |      100 |
  preventBruteforce.js         |    85.71 |       50 |    66.67 |    92.31 |
 api/services                  |    89.47 |     82.5 |    84.62 |    92.96 |
  CryptographyService.js       |    91.43 |    88.46 |      100 |      100 |
  GoogleRecaptchaAPI.js        |      100 |       50 |      100 |      100 |
  MailService.js               |    44.44 |        0 |        0 |    44.44 |
  OpentisAPI.js                |      100 |      100 |      100 |      100 |
  ResponseErrorHandler.js      |      100 |       75 |      100 |      100 |
  SledovanitvAPI.js            |      100 |      100 |      100 |      100 |
  TokenService.js              |      100 |      100 |      100 |      100 |
```

Ukázka kódu 4.2: Pokrytí kódu testováním

4.5 End-to-End testování

End-to-End, zkráceně E2E, testování se zaměřuje na testování celé aplikace včetně integrace s ostatními systémy a skrze konečné rozhraní webového pro-

hlížeče prochází funkcionalitu aplikace. Testy jsou automatizované a prováděné v reálném prostředí internetového prohlížeče, který simuluje skutečného uživatele. E2E testování pokrývá širokou škálu funkcionality aplikace a velmi přispívá k výsledné kvalitě testovaného produktu.

Pro E2E testování aplikace byl použit nástroj *Nightwatch.js*, což je testovací nástroj pro *Node.js* umožňující jednoduché psaní E2E automatizovaných testů[41]. *Nightwatch.js* komunikuje spolu s nástrojem *Selenium WebDriver*, který zprostředkovává automatizaci uživatelských akcí v internetovém prohlížeči[42].

V ukázce kódu 4.3 je vypsán E2E test, který zkontroluje přihlašovací stránku, přihlásí se, zkontroluje úvodní stránku a provede změnu zobrazeného jazyka aplikace. Test je složen z několika oddělených částí, které jsou spouštěny seshora dolů. Tyto části zpravidla obsahují instrukce pro provedení nějaké akce, čekaní na provedení a kontrolu stavu aplikace, zda odpovídá očekávané formě.

Ve čtvrté části ukázkového testu s názvem *Change Locale on Dashboard Page* se na řádku 25 přesune kurzor myši na uživatelské menu aplikace a počká 500 ms na zobrazení položky pro změnu jazyka. Na řádku 26 proběhne kontrola, zda je element pro změnu jazyka viditelný na stránce. Na řádku 27 je instrukce pro kliknutí na element pro změnu jazyka na češtinu a řádky 30 až 33 kontrolují, zda je aplikace správně přeložena tím, že očekávají v různých elementech aplikace české fráze.

```
module.exports = {
  'init check': function (browser) {
    const devServer = process.env.VUE_DEV_SERVER_URL
    browser.url(devServer)
    browser.expect.element('#app').to.be.present.before(5000)
    browser.expect.element('#loginContainer').to.be.visible.before(1000)
    browser.expect.element('div.v-card__title.headline').text.to.equal('
        Přihlášení').before(1000)
  },
  'change locale on login page': function (browser) {
    browser.pause(100).click('#footChangeLocaleBtn').pause(50)
    browser.expect.element('div.v-card__title.headline').text.to.equal('
        Login').before(1000)
  },
  'log in and check content': function (browser) {
    browser
    .setValue('input[name=login]', '803260')
    .setValue('input[name=password]', 'admin')
    .click('#loginButton').pause(500)
    browser.expect.element('#billingStateHeadline').to.be.present.before
        (1000)
    browser.expect.element('#billingStateHeadline').text.to.equal('
        Billing state').before(1000)
    browser.expect.element('.showBillingButton').to.be.present.before
        (1000)
    browser.expect.element('.showBillingButton').text.to.equal('Show
        billing').before(1000)
    browser.pause(300).assert.elementCount('.serviceCard', 4)
  },
  'change locale on dashboard page': function (browser) {
    browser.moveToElement('div.v-toolbar__items > div.v-menu > div', 10,
         10).pause(500).useXpath()
    .expect.element('//a[@id="langSwitch"]/div').to.be.visible.before
        (1000)
    browser.click('//a[@id="langSwitch"]/div')
    .pause(300).useCss()
    browser.expect.element('#billingStateHeadline').to.be.present.before
        (1000)
    browser.expect.element('#billingStateHeadline').text.to.equal('Stav
        vyúčtování').before(1000)
    browser.expect.element('.showBillingButton').to.be.present.before
        (1000)
    browser.expect.element('.showBillingButton').text.to.equal('Vyúčtová
        ní').before(1000)
    browser.pause(500).end()
  }
}
```

Ukázka kódu 4.3: Příklad E2E testu pomocí nástroje *Nightwatch*

Kapitola 5

Nasazení aplikace

Po důkladném otestování aplikace popsaném v minulé kapitole proběhlo nasazení aplikace do provozu. Tato kapitola nejprve popisuje, jak je vyřešeno zálohování a monitoring aplikace. Dále popisuje, do jakého prostředí byla aplikace nasazena. Následující sekce diskutují také softwarové závislosti aplikace, síťovou konfiguraci zapojení nebo proces sestavení aplikace.

5.1 Zálohování a monitoring

Pro splnění nefunkčního požadavku **NP.5** je nutné zajistit monitoring a zálohování aplikace. Ze sekce 3.5 je známo, že všechny verze zdrojového kódu aplikace jsou zálohované na interním GitLabu. Uživatelská data uložená v databázi jsou každý den zálohovaná, přičemž historie záloh je z bezpečnostních důvodů uložena na jiném fyzickém umístění, než kde se nachází aplikace.

Aplikace je zároveň monitorovaná nástrojem Centreon, který kontroluje, zda webový server správně funguje.

5.2 Prostředí aplikace

Aplikace je nasazena do izolovaného virtuálního kontejneru, který je zprostředkovaný virtualizačním nástrojem OpenVZ, což je open-source řešení pro virtualizaci oddělených Linuxových kontejnerů na úrovni operačního systému fyzického serveru. Tato metoda virtualizace přináší výhody ve smyslu snížení administrace i managementu virtuálních prostředí v porovnání s ostatními typy virtualizačních technik (virtuální stroj nebo paravirtualizace) a zároveň zachovává porovnatelný výpočetní výkon[43].

Samotný kontejner aplikace má k dispozici následující výpočetní prostředky.

- Processor 2x Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
- Paměť 2GB RAM (1GB swap)

- Operační systém CentOS Linux 7.4.1708

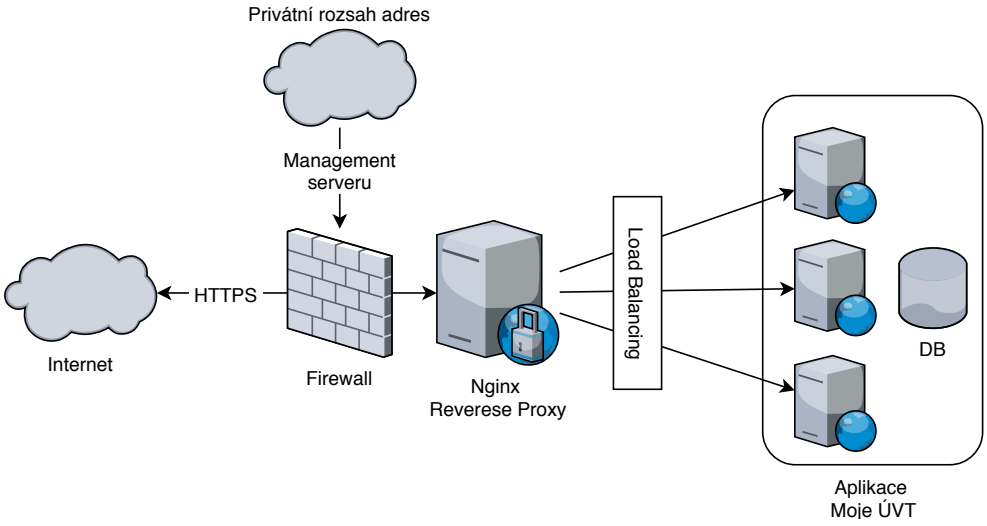
5.3 Softwarové závislosti aplikace

Kontejner je vybaven softwarem nutným pro chod aplikace. Požadované softwarové závislosti jsou následující:

- MongoDB v3.6.3
- Git 2.8.1
- NodeJS v9.9.0
- Nginx 1.13.12

5.4 Síťová konfigurace serveru aplikace

Virtuální server má nastavený striktní firewall a povoluje příchozí připojení pouze na specifikovaných portech. Veškeré porty pro management serveru jsou otevřeny pouze pro privátní rozsah adres zadavatele. Aplikačnímu HTTP Sails serveru je předřazen program Nginx. Nginx je populární webový server, který zde funguje jakožto reverzní proxy, která řeší SSL/TLS šifrování zabezpečeného HTTPS protokolu a dále umožňuje případný load–balancing nebo ochranu proti DDoS útokům[44]. Nginx dále umožňuje automatickou obnovu certifikátu certifikační autority Let's Encrypt, kterou aplikace využívá[45].



Obrázek 5.1: Schéma síťové konfigurace aplikace

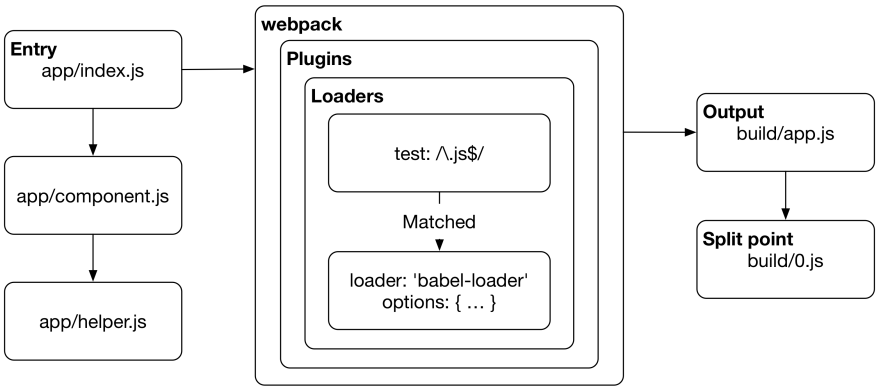
Na obrázku 5.1 je zobrazené síťové zapojení aplikace i s případným rozdělením zátěže na více aplikačních serverů, neboli load balancing. Tato technika

však pro aktuální zátěž aplikace není potřeba a proto je zapojen pouze jeden aplikační server.

5.5 Proces sestavení aplikace

Ze sekce 2.4.1 je známo, že aplikace je rozdělena na dvě hlavní části: backend server a frontend. Frontend je před nasazením nutné sestavit, a vytvořit tak build, který bude backend server poskytovat uživatelům.

Pro sestavení aplikace jsou využité moderní sestavovací nástroje, které jsou součástí balíčku *vue/cli*. Tento balíček umožňuje jednoduchým příkazem sestavit aplikaci pomocí nástroje Webpack, který je nejpopulárnějším sestavovacím nástrojem pro aplikace využívající technologii Node.js. Většina velmi rozsáhlého nastavení Webpacku je tak abstrahována a nastavena na konfiguraci optimalizovanou pro sestavení Vue.js projektu, přičemž je zde stále možnost tyto hodnoty nastavit ručně.



Obrázek 5.2: Proces sestavení aplikace pomocí nástroje Webpack [46]

Na obrázku 5.2 je zobrazen proces sestavení aplikace. Webpack projde závislosti od vstupního bodu aplikace *index.js*, vytvoří graf závislostí a provede optimalizované sestavení aplikace. Na výstupu je pak soubor *app.js* a případně další, pokud je sestavení nakonfigurované pro rozdělení kódu na více částí. Rozdělení na menší části může mít pozitivní efekt na rychlost přenosu a načtení aplikace. Rozdělení kódu se používá především u aplikací větších než 1MB, kde už začíná být tento efekt viditelný. Nová aplikace rozdělení kódu kvůli své velikosti nepotřebuje.

Po sestavení je k dispozici analýza buildu, která sleduje různé metriky. Nejdůležitější metrikou je velikost buildu, která přímo ovlivňuje rychlost odeslání frontendu k uživateli.

Po provedení procesů výše je build zkopírován do složky *assets* na backend serveru a je připraven ke spuštění.

5.6 Migrace dat původního řešení

Zákazníci, kteří využívali původního řešení aplikace, měli přiřazené přihlašovací údaje v podobě klientského čísla a hesla. Klientská čísla účtů byla zachována, nicméně bylo rozhodnuto, že zákazníci si budou muset vytvořit hesla nová. Hlavní důvod tohoto rozhodnutí byla bezpečnost databáze. Hesla v původní databázi totiž byla hashována algoritmem, který byl nevhodný pro hash hesel. Nové řešení používá algoritmus *bcrypt* a muselo by řešit zpětnou kompatibilitu s původním algoritmem. Vzhledem k nízkému počtu uživatelů původní aplikace bylo řešení tohoto problému kompatibility málo opodstatněné.

Migrace probíhala pomocí skriptu, kdy bylo postupně voláno API backendu nové aplikace pro tvorbu nových zákaznických účtů. Všem klientům s přístupem do nové aplikace byly rozeslány registrační emaily pro vytvoření hesla. Ty se velmi podobají emailu pro resetování zapomenutého hesla, který je zobrazen na obrázku 3.20.

Závěr

Cílem této práce bylo seznámit se se stávajícím řešením, analyzovat, navrhnout a implementovat webovou aplikaci pro zákazníky firmy ÚVT Internet s.r.o. podle požadavků zadavatele a zároveň aplikaci integrovat do jeho prostředí.

Splněné cíle

Cíle této práce byly splněné v plném rozsahu. Popis realizace jednotlivých požadavků zadání je uveden v následujících sekcích.

Zobrazení profilu zákazníka

Zobrazení profilu zákazníka je definováno přímo jako případ užití aplikace **UC.1**. Výsledek komponenty stránky, která tento případ užití realizuje, je vidět na obrázku 3.12.

Možnost odeslat požadavek na změnu osobních údajů

Entita *Požadavek* je diskutována v doménovém modelu aplikace v sekci 1.5.6. Případ užití **UC.2** vychází z tohoto bodu zadání a jedna z jeho realizací je vidět například na obrázku 3.13 v sekci profilu zákazníka 3.3.3.6, kde lze odeslat požadavek na změnu fakturačních údajů.

Zobrazení aktivních služeb zákazníka a informace o jejich stavu

Zobrazení aktivních služeb zákazníka je možné několika způsoby. Sekce *Přehled* a *Moje služby* z implementace v sekci 3.3.3 jsou jedněmi z nich. Další možný způsob realizace případu užití **UC.12** je při zobrazení detailu dané služby, jehož implementace je popsána v sekci 3.3.3.7.

Zobrazení vyúčtování/faktur jednotlivých služeb

Zobrazení vyúčtování a faktur za jednotlivé služby je realizováno případy užití **UC.8** a **UC.9**. V sekci 3.3.3.3 je popsána implementace tohoto požadavku a vyobrazena ukázka samotného vyúčtování.

Výpis telefonních hovorů (v případě, že zákazník využívá této služby)

Implementace výpisu telefonních hovorů je popsána v sekci 3.3.3.7. Možnost stažení výpisů je realizována komponentou *Detail hlasové služby*, která je zobrazena na obrázku 3.16.

Možnost aktivace dalších služeb pro zákazníka

Zadavatel u nové aplikace momentálně podporuje pouze aktivaci služby internetové televize *sledovani.tv*. Ukázka komponent detailu této služby a její aktivace je zobrazena na obrázku 3.15.

Možnost zobrazení QR kódu pro mobilní platbu nezaplacených faktur

Možnost zobrazení QR kódu je k dispozici pro všechna nezaplacená vyúčtování. Diskuze realizace tohoto požadavku je v sekci 3.3.3.3 spolu s ukázkou dialogu se zobrazeným QR kódem pro mobilní platbu na obrázku 3.9.

Zobrazení přehledu informativních a provozních zpráv zaslaných zákazníkovi

Zobrazení zpráv zákazníka je implementováno podobně jako komponenta pro zobrazení zákazníkových požadavků. Tato podobnost je popsána v sekci 3.3.3.5.

Multiplatformní přístup – řešení bude dostupné online jako webová aplikace

Výsledné řešení má formu webové aplikace. V sekci 3.1 probíhá diskuze a výběr technologií pro tvorbu webových aplikací.

Cizojazyčná lokalizace – aplikace bude podporovat lokalizace pro další jazyky

Řešení internacionalizace je implementováno za pomocí pluginu *VueTranslate*, jehož ukázka použití je v sekci 3.1.4.3.

Zabezpečení aplikace proti neoprávněnému vniknutí

Otázka zabezpečení je v této práci diskutovaná v kapitolách Návrh a Implementace. Konkrétní metody zabezpečení proti neoprávněnému vniknutí jsou uvedeny například v sekci 3.4.5 řešící způsob autorizace zákazníka nebo v sekci 5.4 zabývající se zabezpečením přístupu k managementu aplikace.

Zabezpečení aplikace proti úniku osobních údajů

Tento bod úzce souvisí s předešlým bodem o zabezpečení. Zabezpečení proti úniku osobních údajů je především dáno způsobem integrace se zadavatelovým interním systémem Opentis pomocí zabezpečeného API, jehož bezpečností se zabývá sekce 2.4.4.5. Lokální data aplikace, jako hesla zákazníků, jsou chráněna silným hashovacím algoritmem *bcrypt*, což je také popsané v sekci 3.4.5.

Integrace se softwarovou infrastrukturou zadavatele

Integrace se systémy zadavatele je řešena skrz všechny kapitoly tohoto textu. Sekce 1.1.1.1 popisuje původní integraci a sekce 2.1 navrhuje nový způsob integrace aplikace a systému Opentis. Diskuze implementace této integrace je uvedena v sekci 3.2. Sekce 4.3 pak přímo řeší integrační testování nové aplikace.

Nasazení aplikace do provozu

Nasazení nové aplikace již v požadavcích zadání této práce není, nicméně aplikace *Moje ÚVT* byla úspěšně nasazena do reálného provozu 27. 07. 2018 na webové adrese `https://moje.uvtnet.cz`. Aktuálně už ji použilo zhruba 30 % všech zákazníku zadavatele, kterým byl odeslán registrační email pro vytvoření hesla.

Literatura

[1] Richter, P.: Opentis (Open-source Enterprise Information System). [online], 2009-2010. Dostupné z: http://www.opentis.com

[2] The PHP Group: PHP: Introduction - Manual. [online], [cit. 2018-12-05]. Dostupné z: http://php.net/manual/en/intro.mysql.php

[3] Red Hat, Inc.: What are APIs? [online], [cit. 2018-12-05]. Dostupné z: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

[4] Dunkley, W.: Split Stack Development: A Model For Modern Applications. *Medium - Mentally Friendly*, červen 2014, [cit. 2018-10-02]. Dostupné z: https://medium.com/@MentallyFriendly/split-stack-development-a-model-for-modern-applications-d7b9abb47bd5

[5] Microsoft Corp.: The MVVM Pattern. únor 2012, [cit. 2018-09-26]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)

[6] Contributors, K.: Knockout : Observables. *Vue.js Documentation*, září 2018, [cit. 2018-10-02]. Dostupné z: https://knockoutjs.com/documentation/observables.html

[7] Jarrell, J.: Is Angular2 MVVM? [online], září 2017. Dostupné z: https://www.quora.com/Is-Angular2-MVVM

[8] Vue.js Contributors: Getting Started - vue.js. *Vue.js Guide*, červen 2015, [cit. 2018-10-02]. Dostupné z: https://012.vuejs.org/guide/

[9] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, červen 1999. Dostupné z: http://www.rfc-editor.org/rfc/rfc2616.txt

[10] Rescorla, E.: HTTP Over TLS. RFC 2818, RFC Editor, květen 2000. Dostupné z: http://www.rfc-editor.org/rfc/rfc2818.txt

[11] Morley, M.: JSON-RPC 2.0 Specification. [cit. 2018-10-15]. Dostupné z: https://www.jsonrpc.org/specification

[12] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, březen 2014. Dostupné z: http://www.rfc-editor.org/rfc/rfc7159.txt

[13] Node.js Foundation: About Node.js®. říjen 2017, [cit. 2018-10-17]. Dostupné z: https://nodejs.org/en/about/

[14] Node.js Foundation: Node.js. říjen 2017, [cit. 2018-10-17]. Dostupné z: https://nodejs.org/en/

[15] Google LLC: V8 JavaScript engine. [cit. 2018-10-17]. Dostupné z: https://v8.dev/

[16] npm, Inc.: npm. [cit. 2018-10-17]. Dostupné z: https://www.npmjs.com/

[17] MongoDB, Inc.: Introduction to MongoDB — MongoDB Manual. [cit. 2018-10-17]. Dostupné z: https://docs.mongodb.com/manual/introduction/

[18] MongoDB, Inc.: Documents — MongoDB Manual. [cit. 2018-10-17]. Dostupné z: https://docs.mongodb.com/manual/core/document/

[19] MongoDB, Inc.: BSON Types — MongoDB Manual. [cit. 2018-10-17]. Dostupné z: https://docs.mongodb.com/manual/reference/bson-types#objectid

[20] MongoDB, Inc.: Indexes — MongoDB Manual. [cit. 2018-10-17]. Dostupné z: https://docs.mongodb.com/manual/indexes#default-id-index

[21] Mardan, A.: Node.js frameworks. [online], červenec 2018. Dostupné z: http://nodeframework.com

[22] StrongLoop; IBM; other expressjs.com contributors: Express 4.x - API Reference. [online], prosinec 2018, [cit. 2018-12-05]. Dostupné z: https://expressjs.com/en/4x/api.html

[23] SPEC INDIA: React vs Angular vs Vue.js: A Complete Comparison Guide. [online], srpen 2018. Dostupné z: https://www.spec-india.com/blog/react-vs-angular-vs-vue-js-a-complete-comparison-guide/

[24] Merskin, A.: Should I learn React.js or Vue.js? Is it worth it if I learn Vue.js first and then React.js? [online], březen 2018. Dostupné z: https://www.quora.com/Should-I-learn-React-js-or-Vue-js-Is-it-worth-it-if-I-learn-Vue-js-first-and-then-React-js

[25] Vue.js Contributors: Introduction — Vue.js. *Vue.js Guide*, [cit. 2018-10-25]. Dostupné z: https://vuejs.org/v2/guide/

[26] Vue.js Contributors: Component tree. *Vue.js Guide*, [cit. 2018-10-25]. Dostupné z: https://vuejs.org/v2/guide/#Composing-with-Components

[27] Robertson, C.; Wiley, J.; Lee, J.; aj.: Google I/O 2014 - Material design principles. [online], červen 2014, [cit. 2018-10-29]. Dostupné z: https://www.youtube.com/watch?v=isYZXwaP3Q4

[28] Vuetify.js Contributors: Cards — Vuetify.js. [cit. 2018-10-29]. Dostupné z: https://vuetifyjs.com/en/components/cards

[29] Vuetify.js Contributors: Layout grid system — Vuetify.js. [cit. 2018-11-04]. Dostupné z: https://vuetifyjs.com/en/layout/grid

[30] Franks, J.; Hallam-Baker, P. M.; Hostetler, J. L.; aj.: HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, RFC Editor, červen 1999. Dostupné z: http://www.rfc-editor.org/rfc/rfc2617.txt

[31] Jones, M.; Bradley, J.; Sakimura, N.: JSON Web Token (JWT). RFC 7519, RFC Editor, květen 2015. Dostupné z: http://www.rfc-editor.org/rfc/rfc7519.txt

[32] Josefsson, S.: The Base16, Base32, and Base64 Data Encodings. RFC 4648, RFC Editor, říjen 2006. Dostupné z: http://www.rfc-editor.org/rfc/rfc4648.txt

[33] ISRG: Let's Encrypt - Free SSL/TLS Certificates. [online], [cit. 2018-12-05]. Dostupné z: https://letsencrypt.org/

[34] Provos, N.; Mazieres, D.: Bcrypt Algorithm. [online], 1999, [cit. 2018-12-05]. Dostupné z: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node5.html

[35] Aboukhadijeh, F.: JavaScript Standard Style. [cit. 2018-10-25]. Dostupné z: https://standardjs.com/

[36] JS Foundation; other contributors: Getting Started with ESLint - ESLint - Pluggable JavaScript linter. *User guide*, [cit. 2018-10-25]. Dostupné z: https://eslint.org/docs/user-guide/getting-started

[37] McFarlin, T.: The Beginner's Guide to Unit Testing: What Is Unit Testing? červen 2012, [cit. 2018-10-09]. Dostupné z: https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728

[38] visionmedia: SuperTest. *GitHub repository*, 2018. Dostupné z: https://github.com/visionmedia/supertest

[39] Instanbul Contributors: Istanbul, a JavaScript test coverage tool. [cit. 2018-10-09]. Dostupné z: https://istanbul.js.org/

[40] Mocha Contributors: Mocha - the fun, simple, flexible JavaScript test framework. [cit. 2018-10-09]. Dostupné z: https://mochajs.org/

[41] Nightwatchjs.org: Nightwatch.js | Node.js powered End-to-End testing framework. květen 2018, [cit. 2018-09-26]. Dostupné z: http://nightwatchjs.org/

[42] Selenium Contributors: Selenium - Web Browser Automation. srpen 2018, [cit. 2018-09-26]. Dostupné z: https://www.seleniumhq.org/

[43] Kolyshkin, K.: Introduction to virtualization. prosinec 2015, [cit. 2018-10-16]. Dostupné z: https://wiki.openvz.org/Introduction_to_virtualization

[44] proinity LLC: Setting up an Nginx Reverse Proxy. *KeyCDN*, říjen 2018, [cit. 2018-09-26]. Dostupné z: https://www.keycdn.com/support/nginx-reverse-proxy/

[45] Rawdat, A.: Update: Using Free Let's Encrypt SSL/TLS Certificates with NGINX. *Nginx Blog, section Tech*, říjen 2017, [cit. 2018-10-16]. Dostupné z: https://www.nginx.com/blog/using-free-ssltls-certificates-from-lets-encrypt-with-nginx/

[46] Koppers, T.: Webpack's execution process. [online], 2018. Dostupné z: https://survivejs.com/538c4af0d21e375d6d252d38cbb8a993.png

Příloha A

Seznam použitých zkratek

API Application Programming Interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

UI User Interface

E2E End-to-End

URL Uniform Resource Locator

CRUD Create Read Update Delete

RPC Remote Procedure Call

REST Representational state transfer

MVC Model–View–Controller

MVP Model–View–Presenter

MVVM Model–View–ViewModel

CLI Command Line Interface

DDoS Distributed Denial of Service

CPU Central Processing Unit

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF