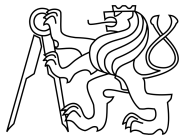
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Portlet pro filtraci a vyhledávání obchodních míst
Student:	Vojtěch Bakaj
Vedoucí:	Mgr. Jindřich Houska
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Cílem práce je návrh a vývoj konfigurovatelného portletu na platformě Liferay sloužícího pro vyhledávání a filtraci obchodních míst v bankovním sektoru.

- Sesbírejte, analyzujte a formalizujte požadavky na portlet.
- Návrhněte architekturu portletu tak, aby jej bylo možné sestavit z dílčích modulů a nakonfigurovat podle specifických požadavků. Využijte Google Maps API pro vykreslování mapy a operace nad ní. Klíčové je dodržení požadavků na rychlost operací nad mapou a samotné načítání stránky.
- Pro shlukování značek na mapě existuje nad Google Maps API více knihoven. Z pohledu funkcionality a rychlosti porovnejte tyto knihovny a vyberte knihovnu nejvhodnější pro použití při implementaci.
- Návrh implementujte, řádně zdokumentujte a otestujte. Součástí práce bude příručka pro nasazení a konfiguraci portletu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 20. února 2017

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

Bakalářská práce

Portlet pro filtraci a vyhledávání obchodních míst

Vojtěch Bakaj

Vedoucí práce: Mgr. Jindřich Houska

15. května 2017

Poděkování

Rád bych poděkoval Mgr. Jindřichu Houskovi za jeho ochotu, vstřícnost a trpělivé vedení mé práce. Dále bych chtěl poděkovat Bc. Jakubu Černohubému za cenné rady a připomínky. Děkuji také své přítelkyni Veronice Šotolové a rodičům za podporu během celé doby studia.

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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 .....

Odkaz na tuto práci

Bakaj, Vojtěch. *Portlet pro filtraci a vyhledávání obchodních míst.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem této práce je nahradit aktuální aplikaci, která slouží k vyhledávání a filtrování obchodních míst, novou aplikací, která ji bude rozšiřovat o nové funkce a zefektivňovat. Práce obsahuje analýzu současné aplikace, sběr požadavků, návrh a následnou implementaci nové aplikace.

Vytvořené řešení poskytuje plně konfigurovatelnou aplikaci rozdělenou do více modulů. Umožňuje tedy administrátorovi umístit aplikaci na svůj web a sestavit si ji podle svým představ a potřeb.

Hlavní přínos této práce je zejména pro koncové uživatele, kteří si lehce z jakéhokoliv zařízení můžou najít nejbližší bankomat, či jiná obchodní místa.

Klíčová slova Liferay, Portlet, Spring WebFlow, Java, Javascript, Banka

Abstract

The purpose of this thesis is to replace the current application that is used to search and filter business places by new application that will expand it with new features and that will make it more effective. The thesis includes analysis of the current application, collection of requirements, design and subsequent implementation of the new application.

My solution provides a fully configurable application divided into multiple modules. It allows the administrator to place the application on his website and build it according to his ideas and needs.

The main contribution of this thesis is especially for end users who can easily find from any device the nearest ATM or other business places.

Keywords Liferay, Portlet, Spring WebFlow, Java, Javascript, Bank

Obsah

Seznam obrázků

Úvod

V dnešní době má už téměř každý člověk u libovolné banky zřízený minimálně jeden bankovní účet, a to většinou ještě před branami dospělosti. K této skutečnosti se vztahuje několik služeb, které můžeme využívat. Mezi nejběžnější můžeme zařadit například výběr z bankomatů nebo poradenství na pobočce banky. Ne vždy ale víme, kde se daný bankomat či pobočka nachází, zda má otevřeno a jaké konkrétní služby nám jsou k dispozici.

Výsledek mé práce umožní přehledně a jednoduše informovat klienty banky o pozici, otevírací době a službách, které jednotlivá obchodní místa nabízí, formou interaktivní webové aplikace. Uživatelům chytrých telefonů bude navíc poskytovat geolokalizaci a následnou navigaci k nejbližším obchodním místům. Dále bude mít přínos také pro správce obsahu portálu, protože portlet bude rozdělen do modulů, které blíže rozeberu ve své bakalářské práci. Celá aplikace bude plně konfigurovatelná, takže správci si budou moci sestavit aplikaci podle svých představ a blíže ji tak přizpůsobit obsahu stránky.

Toto téma jsem si zvolil, neboť aktuální portlet konkrétní Banky nesplňuje všechny požadavky businessu, je jednoúčelový a špatně udržovatelný z hlediska budoucího vývoje.

V práci se nejprve zabývám analýzou současného stavu a poté přejdu ke sběru funkčních i nefunkčních požadavků od businessu a týmu, který zodpovídá za obsah na portálu. Dále jsi určím cíle, kterých bych rád dosáhnul a rozeberu možnosti konfigurace jednotlivých modulů. Následně vytvořím návrh řešení a prodiskutuji zvolené technologie. Poté celý portlet implementuji, porovnám rychlosti knihoven, které zajišťují shlukování nad mapou. Na závěr aplikaci otestuji jednotkovými a integračními testy.

Kapitola 1

Vymezení pojmů

1.1 Jednotlivé pojmy

Ještě před samotnou prací bych rád vymezil několik pojmů, které budu ve všech částech mé práce několikrát opakovat.

1.1.1 Portál

Liferay Portal je podnikový portál založený na jazyce Java. Jeho úkolem je spravovat aplikace, procesy a data z jednoho centrálního rozhraní. Celý portál se skládá z jednotlivých funkčních celků, kterým se říká portlety. [2]

1.1.2 Portlet

Portlety jsou výměnné webové komponenty, umožňující integraci webových aplikací a portálů. Instance jednotlivých portletů se vkládají do stránek vytvořené portálem a vytváří tím dynamický obsah stránek. Každy portlet se skládá z více módů, pro nás budou důležité dva. [3]

První z nich je editační mód. Ten uvidí pouze WCM a bude sloužit ke konfiguraci portletu, která je ke každé instanci portletu unikátní.

Druhý z módů je prezentační, pro naše potřeby to bude grafické rozhraní pro koncového uživatele.

1.1.3 WCM

Zkratka WCM(Web content management) je tým lidí, kteří se starají o obsah celého portálu. Vytvářejí textace, přidávají a nastavují jednotlivé portlety.

1.1.4 BUS

BUS, neboli business, je soubor lidí kteří vytváří zadání, které potom předají na IT analytiky.

1.1.5 Aplikační inženýr

Aplikační inženýr je člověk, který se stará o chod aplikací na portálu a zároveň je vlastníkem těchto aplikací. Dále komunikuje se zákazníkem a upravuje návrhy požadavků.

1.1.6 Obchodní místo

Též obchodní provozovna nebo prodejní jednotka. Jedná se o fyzické prostory přijímající platební karu k úhradě zboží nebo služeb, a to s obsluhou nebo samoobslužné. Mezi nejčastější obchodní místa můžeme zařadit například bankomaty, pobočku banky, poštu a nebo cashback.

1.1.7 Pin na mapě

V tomto kontextu se pinem myslí interaktivní značka umístěná na mapě, značící obchodní místo.

1.1.8 Frontend

Pojem frontend slouží k označení části webu, která je viditelná pro běžného uživatele. Nejběžnější technologie pro frontendový vývoj jsou HTML, CSS a JavaScript.

1.1.9 Middleware

Middleware je specializovaný software, který poskytuje aplikacím služby nad rámec služeb poskytovaných od operačního systému. Usnadňuje vývojářům vývoj komunikace a vstupů nebo výstupů. Tedy vývojáři se můžou soustředit na cíl své aplikace. Je to v podstatě software spojující dvě softwarové komponenty nebo aplikace. [4]

1.1.10 Backend

Opakem frontendu je backend, část aplikace sloužící k administraci. Zde se určuje a ovlivňuje obsah, které frontend zobrazuje.

1.1.11 MVC architektura

Softwarová architektura MVC(Model-view-controller) rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých vrstev. Tím se docílí, že úprava jedné vrstvy má minimální vliv na ostatní.

1.1.12 Framework

Framework je softwarová struktura, která slouží jako podpora při vývoji projektů. Obsahuje podpůrné programy a knihovny. Námi používaný Spring Web-Flow zajišťuje MVC architekturu portletů, zabezpečení a lze v něm přehledně zadefinovat přechody mezi jednotlivými obrazovkami.

1.1.13 AJAX

Technologie, která dokáže upravovat obsah stránky bez nutnosti znovunačtení celé stránky. Umožňuje tedy odesílání požadavků na sever a také jejich přijímání. Vše probíhá na pozadí stránky, po přijetí požadavku se akorát znovu načte určitý fragment na stránce. [5]

1.1.14 API

Application Programming Interface, zkráceně API, označuje rozhraní pro programování aplikací. Jedná se o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat.

1.1.15 URL

Celým názvem Uniform Resource Locator je řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací na internetu. Definuje doménovou adresu serveru, umístění zdroje na serveru a protokol, kterým je možné ke zdroji přistupovat. [6]

1.1.16 XML

Extensible Markup Language zkráceně XML, je obecný značkovací jazyk, který slouží k ukládání, serializaci a přenosu dat.

1.1.17 Json

JavaScriptový objektový zápis, slouží ke stejnému účelu jako XML, je to jeden z jeho konkurentů.

Kapitola 2

Analýza současného stavu

2.1 Současný portlet

Aktuální portlet obsahuje dva způsoby zobrazení. Prvním z nich je mapa, na které jsou obchodní místa vyznačená piny. Při určitém přiblížení mapy se třetina mapy skryje a nahradí se boční tabulkou. Po kliknutí na pin nebo na řádek v tabulce se zobrazí bublina v mapě, na které jsou vypsány všechny informace o daném obchodním místě, včetně odkazu na portlet detail.

Detail je další portlet, který načítá informace podle identifikátoru, který se nachází v přátelské url[1]. Tento portlet nebudu dále rozebírat, generuje pouze statický obsah a není součástí této práce.

Druhý z módů je tabulkové zobrazení se stránkováním. Na jedné stránce může být maximálně deset obchodních míst.

Nad oběma módy se nachází filtrovací panel, kde pomocí zaškrtávacích políček si můžeme vyfiltrovat místa, které právě hledáme. Navrchu celého portletu je textové pole, které slouží k fulltextovému vyhledávání. Dále také napovídá uživateli potencionální adresu. Po potvrzení své volby se mapa přiblíží na dané geografické místo. Pokud zadanému místu odpovídá více výsledků, zobrazí se tabulka, pomocí které může uživatel upřesnit svoje vyhledávání.

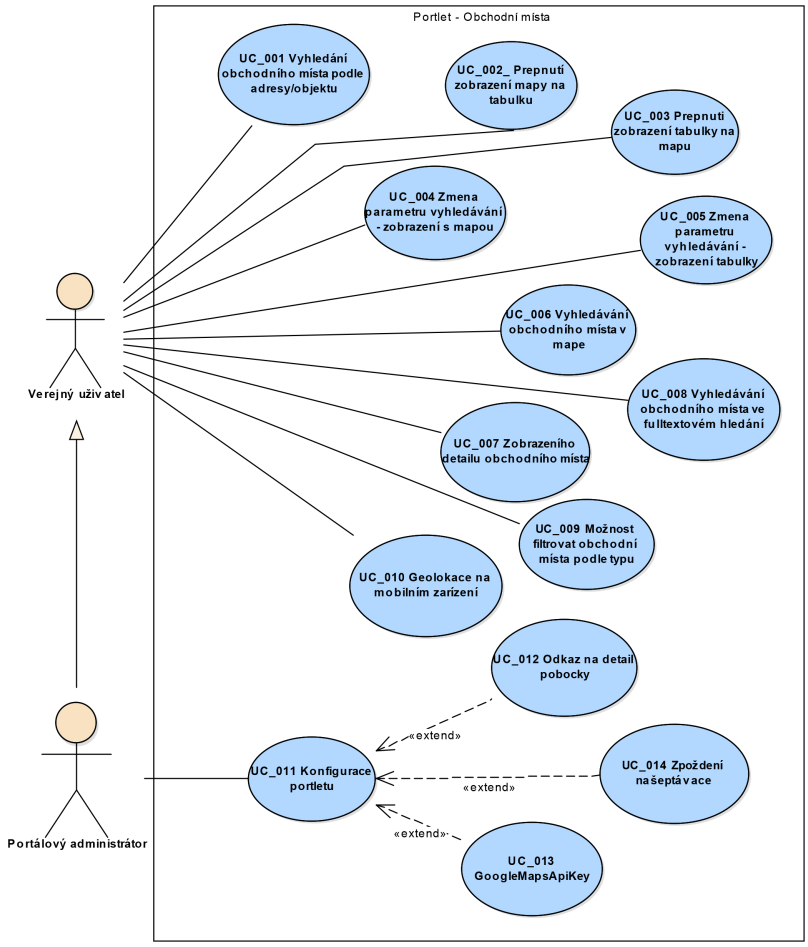
Všechny tyto parametry filtrování včetně pozice mapy, přiblížení a aktuálního vyhledávání se po každé změně v portletu serializují do url adresy, abychom se mohli vrátit kdykoliv o krok zpět, případně poslat výsledek hledání jinému uživateli.

2.2 Případy užití

Na základě funkčnosti portletu jsem sestavil diagram případů užití. Na základě uživatelů, kteří portlet používají, jsem stanovil dvě role. První z nich je

[1] Webová adresa, která lze jednoduše přečíst, často obsahuje slova, která vystihují obsah dané stránky.

veřejný uživatel, který reprezentuje klienty Banky. Druhá z rolí je portálový administrátor, což je člověk z WCM, který konfiguruje daný portlet.



Obrázek 2.1: UseCase diagram stávajícího portletu

U01 - Vyhledání obchodního místa podle adresy/objektu
Uživatel vstupuje na stránku, kde se nachází portlet, pomocí odkazu/položky v menu, kde jsou v url obsaženy parametry pro nastavení vyhledávání. Parametry jsou typy obchodního místa a adresa. Při inicializaci portlet vezme tyto parametry v potaz a zobrazí už výsledek v podobě mapy, či tabulky. Pokud je přiblížení mapy větší než hodnota, kterou udává konfigurace portletu, zobrazí se boční tabulka, která nahradí třetinu mapy a naplní se seznamem míst nacházející se ve výřezu mapy.

U02 - Přepnutí zobrazení mapy na tabulku
Uživatel přepne zobrazení vyhledávače z *mapy* do *tabulky*. V případě, že není zobrazena boční tabulka mapy se seznamem obchodních míst, zobrazí se varování, že tolik míst nezle zobrazit a je potřeba upřesnit vyhledávání. V opačném případě se zobrazí tabulkové zobrazení obchodních míst.

U03 - Přepnutí zobrazení tabulky na mapu
Zobrazí se výřez mapy odpovídající poslednímu stavu mapy, nebo případně novému stavu po vyhledávání, či filtrování v tabulce obchodních míst.

U04 - Změna parametru vyhledávání - zobrazení s mapou
Uživatel upraví typ vyhledaných obchodních míst zaškrtnutím tlačítka u názvu typu obchodního místa. Uživatel může vybrat libovolný počet kombinací typů. S každou změnou se změní výsledek vyhledávání stejně jako kdyby proběhlo nové hledání. Mezi zaškrtnutím a vyhledáním je prodleva daná pouze technologicky, kvůli dotažení dalších obchodní míst.

Při změnách nedochází k novému načtení stránky, ale pouze se mění url stránek, kvůli fungování tlačítka zpět v prohlížeči, či přeposlání výsledků hledání jinému uživateli.

U05 - Změna parametru vyhledávání - zobrazení tabulky
Stejné chování jako v případu užití U04 s tím že, všechny operace probíhají nad tabulkou.

U06 - Vyhledání obchodního místa v mapě
Uživatel změní přiblížení mapy nebo posune mapu ve výřezu. Proběhne nové vyhledání míst nejbližších ke středu zobrazovaného výřezu mapy. V mapě se zobrazí obchodní místa, která se nachází v nově zobrazeném výřezu. Pokud je mapa dostatečně přiblížena, zobrazí se výsledky hledání i v boční tabulce vedle mapy podle pravidel specifikovaných v U01.

U07 - Zobrazení detailu obchodního místa
Pro zobrazení podrobné informace o obchodním místu, klikne uživatel na ikonu v mapě, na řádek s daným obchodním místem v boční tabulce výsledků nebo na tlačítko v tabulce(viz. zobrazení seznamu v U02). V mapě se zobrazí u vybraného obchodního místa bublina s informacemi, které se liší podle typu obchodního místa.

Pokud dané obchodní místo je pobočka nebo finanční centrum, nachází se v bublině nebo tabulce daného obchodního místa odkaz na portlet detail. Tento odkaz se sestaví podle přátelské url, který je uložen v konfiguraci portletu a také se k odkazu přidá přátelský identifikátor. Ten nejčastěji obsahuje název, město nebo ulici daného obchodního místa.

U08 - Vyhledání obchodního místa ve fulltextovém vyhledávání
Uživatel vyhledává ve fulltextovém hledání v portálu. Pokud je výraz systémem vyhodnocen jako potencionální adresa, tak se na stránce s výsledky zobrazí tento portlet. Vložený portlet bude graficky menší a bude viditelná pouze mapa. Pod portletem bude odkaz *Zobrazit v kontaktech*, kterým uživatel přejde na standardní stránku s plnohodnotným portletem. Zde mu systém zobrazí výřez mapy a stejné typy obchodních míst, které viděl ve fulltextovém hledání.

U09 - Možnost filtrovat obchodní místa podle typu
Nad oběma zobrazeními jsou typy obchodních míst se zaškrtávacími tlačítky. Pokud uživatel klikne na dané tlačítko, změna se ihned projeví na mapě, či v tabulce. Poté se aktuální stav synchronizuje s parametry v url.

U10 - Geolokace na mobilním zařízení
Uživatelům se na mobilních zařízení zobrazí stránka s portletem obchodních míst ve stejném nastavení jako v U01. Při prvním přístupu se prohlížeč zeptá uživatele, jestli chce webu povolit zjišťování pozice.

Pokud ji uživatel povolil, tak po přístupu na stránku bez url parametrů je mapa vycentrována na zjištěnou pozici a úroveň přiblížení je nastavena na přilehlé jednotky kilometrů kolem ní.

U11 - Konfigurace portletu
Správce portálu se přihlásí na portál a přejde na stránku s portletem, odkud se dostane na konfiguraci portletu(*Edit view* v jazyce Liferay portálu). Při prvním zobrazení je konfigurace naplněna výchozími hodnotami. Správce případně upraví hodnoty a uloží konfiguraci portletu(*portlet preferences* v jazyce Liferay).

U12 - Odkaz na detail pobočky
Správce portálu má možnost nastavit absolutní url adresu na portlet detail. Lze nastavit adresu pro dvě lokalizace, českou a anglickou.

U13 - GoogleMapsApiKey
Portlet využívá placenou verzi GoogleMaps rozhraní. Identifikátor je potřeba zpřístupnit k používání těchto služeb.

U14 - Zpoždění našeptávače
Správce portálu může nastavit zpoždění našeptávače, který udává dobu v milisekundách, za kterou napoví uživateli potencionální adresy po zadání libovolného textu do vyhledávacího pole.

2.3 Technologie

Portlet je postaven na jazyku Java EE a frameworku Spring WebFlow. K vykreslení mapy se používá GoogleMaps JavaScriptApi, tedy velká část kódu je psaná právě v JavaScriptu. K prezentační vrstvě se používá JSP technologie. V následujících řádcích tyto technologie blíže specifikuji.

2.3.1 Java SE

Java SE, neboli Java Platform, Standard Edition je široce používaná platforma pro vývoj portabilních aplikací, které se dají nasadit jak na desktop, tak na server. Obsahuje velké množství knihoven, API a frameworků. [7]

2.3.2 Java EE

Java Platform, Enterprise Edition, zkráceně Java EE, je nadstavba platformy Java SE určená pro vývoj a provoz podnikových aplikací a informačních systémů. Mezi součásti této platformy se řadí například JSP a komponenty zajišťující integraci webových aplikací a portálů, jinak řečeno portlety. [8]

2.3.3 JSP

JavaServlet Pages je technologie postavená na Java servletech[2] sloužící především pro ulehčení zápisu výstupu Java aplikace do HTML souborů. [9]

2.3.4 Spring Web Flow

Framework Spring Web Flow je založen na frameworku Spring MVC, který rozšiřuje o implementaci takzvané "flows", což je jazyk, kterým lze popsat krok po kroku průchod celým webem nebo portletem na větší úrovni abstrakce.

2.3.5 JavaScript

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Nejčastěji se používá k ovládání různých interaktivních prvků na webu a jsou pomocí něho tvořeny animace a efekty. V tomto portletu tvoří přibližně 70% celého kódů. Používá se ke komunikaci s GoogleMaps JavaScript Api a pomocí Ajaxu komunikuje i se servisní vrstvou portletu.

2.3.6 jQuery

JavaScriptová knihovna, která je rychlá, malá a podporuje ji většina prohlížečů. Usnadňuje práci s JavaScriptem a Ajaxem. V dnešní době je tato

[2]Program napsaný v jazyce Java, který je nástrojem pro tvorbu webových aplikací.

knihovna spíše standard, má podporu od velkých korporací, jako je například IBM, Adobe a Intel. [10]

2.3.7 jQuery UI

Tato knihovna je nadstavbou nad jQuery. Nabízí sadu rozhraní pro interakci s uživatelem. V tomto portletu se používá kvůli rozhraní "autocomplete", což je komponenta, které se předá zdroj informací. Poté, pokud se uživatel snaží vyhledat v našem případě nějaká obchodní místa, komponenta se mu snaží napovídat potencionální výsledky. [11]

2.3.8 GoogleMaps JavaScriptApi

Společnost Gooogle nabízí API k vykreslení mapy a operace nad ní v několika verzích. Stávající portlet používá placenou verzi, která není limitována přístupy a počtem požadavků odesílajících se na API. Mezi služby všech těchto verzí patří vytváření pinů na mapě, nastavení událostí na různé podněty uživatelů a také vytváření statistik na základě vyhledávání a přístupů k mapě.

2.4 Architektura

Tato sekce se bude zabývat pouze způsobem, jak se načítají data. Více podrobností o architektuře a způsobu komunikace mezi jednotlivými vrstvami rozebereme následně při návrhu nového portletu. Architektura je jak u stávajícího portletu, tak u nového stejná a mění se pouze způsob načítání dat.

Oba portlety jsou postaveny nad určitým backend frameworkem, který příjme požadavek a odešle požadovaná data, pro nás obchodní místa. Načítání míst tedy probíhá ve dvou fázích kvůli optimalizaci.

Při inicializaci portletu se volá služba, která vrací všechny obchodní místa, ale v omezené formě a to pouze *typ, pozice a identifikátor*. Tyto informace nám postačí k tomu, abychom je mohli vykreslit na dané pozici s příslušnou ikonou pinu a filtrovat je podle libosti.

Druhá fáze načítání dat nastane po zobrazení bočního panelu. V bočním panelu se zobrazuje adresa obchodního místa. Je tedy potřeba doplnit informace ke každému obchodnímu místu, které se nachází ve výřezu mapy. O tyto data se postará Ajax, který je získá od kontroléru. O něm také více v 3.1. Tento ajaxový požadavek se zavolá po každé změně výřezu mapy, tedy typicky po *drag&drop* a *vyhledávání*.

2.5 Problémy

Jak již bylo zmíněno v úvodu, portlet byl původně psán jednoúčelově pro čtyři typy obchodních míst. Každý další požadavek na změnu nebo drobná úprava

je tedy velmi komplikovaný, časově náročný a dělá portlet čím dál více neudržitelným.

Celkově má portlet problém s rychlostí načítání stránky a manipulováním s mapou. Načítaných obchodních míst dané Banky je několik tisíc. Při inicializaci mapy se tedy načítají pouze informace, které jsou právě potřeba. Od úrovně přiblížení mapy, kdy se zobrazí boční tabulka, se technologií AJAX dotáhne zbytek dat ke každému místu nacházejícímu se ve výřezu mapy.

Většina JavaScriptového kódu je napsána v jednom souboru, který má přes tři tisíce řádků, což ho činí nepřehledným a špatně udržovatelným.

Další z problémů je velké množství pinů na jednom místě, tato vlastnost dělá mapu nepřehlednou a současně to výrazně zpomaluje interakci a operace nad mapou.

Kapitola 3

Potřeby banky

S postupným vývojem Banky přišla potřeba portlet rozšířit o nové funkce a vlastnosti. Jeden z hlavních požadavků je upravit portlet tak, aby byl více flexibilní a nastavitelný, kvůli umístění na více stránek portálu. Myšlenka je taková, že WCM si portlet nastaví tak, aby co nejvíce vystihoval obsah na dané stránce, a to jak po stránce grafické, tak po stránce funkčnosti portletu. Například pokud mám stránku, která se zabývá bezbariérovým přístupem, mohla by zde být pouze mapa, které má už předfiltrované pobočky s bezbariérovým přístupem, a tak si lze jednoduše a přehledně vybrat vhodnou pobočku. Všechny bankovní požadavky rozeberu v sekci 3.1 níže.

Implementace veškerých těchto požadavků do stávajícího portletu by byla časově náročná a ve většině případů by vedla k přepsání velké části kódu, od základu až po frontendovou část. Proto vznikla tato práce, která se zabývá sběrem a analýzou požadavků, poté návrhem nového portletu a následnou implementací.

3.1 Požadavky

Sběr požadavků na nový portlet probíhal na několika schůzkách, především s WCM týmem a jedna schůzka s majitelem portálu. Na těchto sezení vzniklo několik funkčních i nefunkčních požadavků. Všechny tyto požadavky mají rozšiřovat aktuální portlet. Popíši Vám tedy jenom ty, které ho rozšiřují nebo mění. Stávající požadavky si můžete projít v sekci 2.2.

3.1.1 Funkční požadavky

Následující seznam představuje funkční požadavky na portlet. Ty se zaměřují především na popsání funkčních prvků aplikace, neboli toho, co by měla umět vykonat. Většinou popisují velké množství požadavků, které vzešly z předešlých zkušeností, uživatelských analýz a analýzy konkurenčních aplikací, založených na stejném principu.

F1 - Shlukování pinů na mapě
První z požadavků je shlukování pinů. Díky tomu se mapa zpřehlední a zrychlí. Samotné shlukování přináší jeden problém, a tím je situace, kdy je například bankomat hned vedle pobočky. Potom i přes maximální přiblížení mapy uživatel vidí pouze shluk a ne dva piny, jak by se dalo předpokládat. Tuto situaci je potřeba vyřešit úrovní přiblížení, kde se už piny neshlukují. Následně tuto hodnotu úrovně zpřístupnit v konfiguraci portletu.

F2 - Filtrace podle služeb
V portletu bude dostupná filtrace pro jednotlivé typy obchodních míst. Služby se zobrazí v roletce po najetí myší na dané obchodní místo.

F3 - Navigace k obchodnímu místu v chytrých telefonech
Pokud má uživatel chytrého telefonu zapnutou lokalizaci, tak při přístupu na portlet mu mapa zobrazí nejbližší obchodní místa v jeho okolí. Pokud ji zapnutou nemá, aplikace se ho zeptá, jestli by jí chtěl zapnout. Na mobilních zařízeních v bublině pinu, bude navíc tlačítko "Navigovat". Pokud na něj uživatel klikne, spustí se jeho navigační aplikace s nastavenou trasou ke zvolenému obchodnímu místu. V konfiguraci portletu se bude dát nastavit zobrazování tohoto tlačítka pro jednotlivé typy obchodních míst.

F4 - Rozšíření konfigurace filtrovacího panelu
Nastavení portletu bude obsahovat seznam všech obchodních míst a k nim jejich služby. U všech těchto položek budou dvě zaškrtávací políčka. Jedno bude určovat, zda obchodní místo či jeho služba je zobrazena ve filtračním panelu. Druhé slouží k informaci, která obchodní místa jsou již předfiltrována. Pořadí zobrazení jednotlivých obchodních míst a jejich služeb bude možné měnit operací drag&drop[3].

F5 - Možnost skrýt vyhledávací panel
V konfiguraci portletu bude možné nastavit, zda vyhledávací panel zobrazit či skrýt.

3.1.2 Nefunkční požadavky

Nefunkční požadavky na software jsou požadavky, které kladou omezení na design a provedení. Příklad takových požadavků je výkon aplikace, škálovatelnost, udržitelnost, spolehlivost a mnoho dalších. Všechny tyto požadavky by měly být měřitelné.

[3] Jinak řečeno *táhni a pusť* je operace, kdy uživatel uchopí pomoci myši virtuální objekt a přesune ho přetažením na jiné místo.

N1 - Optimalizace rychlosti načítání stránky
Rychlost načítání stránky by neměla přesáhnout tři sekundy, počínajíce příjetím dat z framweroku viz. 2.4, až po konec inicializace mapy.

N2 - Podpora prohlížečů
Portlet by měl podporovat všechny známé prohlížeče, včetně prohlížeče Internet Explorer a to od verze devět.

N3 - Responsivita aplikace
Aplikace by se měla přizpůsobit uživateli, přistupuje-li z počítače nebo mobilního zařízení, či tabletu. Musí být lehce a intuitivně ovladatelná.

3.2 Případy užití

Následující diagram případů užití jsem sestavil tak, aby popisoval fungování aplikace s ohledem na nové funkční požadavky sepsané v předchozí kapitole. Aktéři opět zůstávají pouze dva, jako v diagramu 2.2.

V následujících řádcích popíši pouze ty případy užití, které jsou nové nebo se změnily oproti stávajícímu portletu.

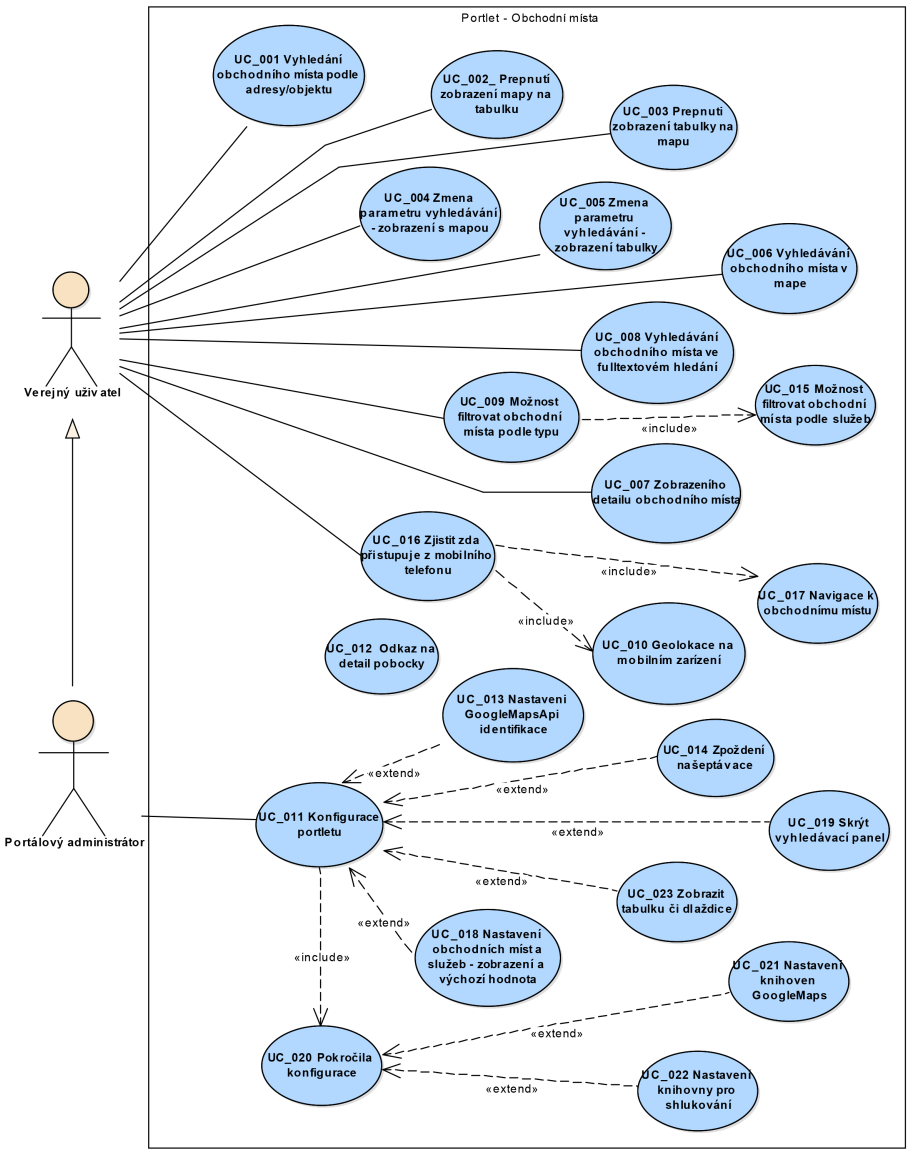
U15 - Možnost filtrovat obchodní místa podle služeb
Uživateli se po najetí na typ obchodního místa zobrazí roletka s výčtem služeb, které dané místo nabízí. Chování je následující: pokud uživatel zaškrtne službu u typu, který není zobrazen tedy zaškrtnutý, automaticky se daný typ zaškrtne sám. To samé platí analogicky pro opačnou situaci. Pokud je aktivní filtr na daný typ se službou, pak odškrtnutí typu se projeví na všech jeho službách.

Typy a služby, které jsou zobrazeny a jejich výchozí nastavení se dá nakonfigurovat viz. U18

U16 - Detekce mobilního zařízení
Pokud uživatel přijde na webovou stránku s portletem, aplikace dokáže rozlišit pokud uživatel přišel z mobilního zařízení, či tabletu. Pokud ano, proběhne geolokace viz. U10 a budou mu zobrazena navigační tlačítka U17.

U17 - Navigace k obchodnímu místu
Uživatel u některých typů obchodních míst může v bublině kliknout na odkaz, který otevře navigaci v mobilním telefonu s předvyplněným cílem trasy k danému obchodnímu místu. Tento případ užití je založen na U16.

U18 - Nastavení zobrazení a výchozí hodnota obchodních míst
Aktér (portálový administrátor) v konfiguraci portletu nalezne výpis typů obchodních míst, které dostane zavoláním služby od backednového frameworku, včetně všech služeb k jednotlivým typům. U každé položky se budou nacházet



Obrázek 3.1: UseCase diagram nového portletu

dvě zaškrtávací tlačítka. Jedno pro detekci zda má být typ, či služba zobrazena ve filtrovacím panelu. Druhé bude sloužit k výchozímu stavu zaškrtnutí dané služby nebo typu při počátečnímu zobrazení portletu.

U19 - Skrýt vyhledávací panel
Portálový administrátor může skrýt vyhledávací panel v konfiguraci portletu.

U20 - Pokročilá konfigurace
Pokročila konfigurace je speciální sekce obyčejné konfigurace, která ale vyžaduje nastudování a pochopení funkčnosti knihoven aplikace. Při zobrazení konfigurace portletu je skrytá, lze ji pomocí tlačítka rozbalit a upravovat. Momentálně se v ní nachází nastavení knihoven od společnosti Google, a.s. a knihovna, která zajišťuje shlukování pinů.

U21 - Nastavení knihoven GoogleMaps
Administrátor může nastavit v pokročilé konfiguraci všechny vlastnosti inicializačního objektu pro vytvoření google mapy. Tento objekt obsahuje například přiblížení, souřadnice středu, typ mapy a mnoho dalších vlastností, které budou k dispozici v konfiguraci portletu.

U22 - Nastavení knihovny pro shlukování
Další z položek pokročilé konfigurace obsahuje parametry jako velikost mřížky, maximální přiblížení, průměrný střed shluku oproti pinům a minimální velikost shlukovaných pinů.

U23 - Zobrazení tabulky
Nastavení zobrazení tabulky na kterou se můžeme přepnout viz. U02. Na výběr jsou dva typy zobrazení - tabulkové se stránkováním a dlaždice.

Kapitola 4

Cíle

Na základě sesbíraných požadavků jsem si vymezil několik cílů, na které jsem se zaměřil při návrhu a implementaci nového portletu. V následujících řádcích tyto jednotlivé cíle rozeberu.

4.1 Modularita

Jeden z nejhlavnějších cílů je právě zobecnění současného portletu. Rád bych ho rozdělil do samostatných konfigurovatelných modulů, které by si poté WCM mohl nastavit a poskládat dle své představy.

4.2 Konfigurovatelnost

Konfigurovatelnost aplikace je jedena z věcí, která pomůže k zobecnění portletu. Díky podrobnému nastavení jednotlivých částí lze ale také předejít budoucím požadavkům na změnu a tím pádem případnému zásahu do kódu aplikace.

4.3 Rychlost

Rychlost aplikace bude taky velice důležitý faktor, na který bych se rád zaměřil. Díky požadavku na filtraci obchodních míst podle služeb bude potřeba načítat daleko více informací už při samotném načítání stránky. Kámen úrazu stávajícího portletu je velké množství pinů vykreslených na mapě. Tento problém bych rád vyřešil výběrem vhodné knihovny, která slouží ke shlukování pinů.

4.4 Přehlednost kódu

Přehlednost kódu je jeden z cílů, na který bych se chtěl zaměřit. Konkrétně na JavaScriptový kód, který je v současném portletu psaný pomalu v jenom souboru. Bohužel Banka neumožňuje použití JavaScriptových knihoven, jako jsou například React, AngularJS a CoffeeScript, které jsou vhodnější na větší aplikace už jenom kvůli udržitelnosti. Jediná povolená knihovna je jQuery a její nadstavba jQuery UI. Budu se tedy snažit JavaScriptovou část co nejvíce zpřehlednit, rozdělit do více funkčních celků a zapouzdřit jednotlivé objekty.

4.5 Rozšířitelnost

Dalším z podstatných cílů této práce je snadná rozšiřitelnost portletu o další funkce. Tato rozšíření by měl být schopen provést každý programátor bez potřeby měnit architekturu aplikace, či detailnějšího studování kódu. Zkrátka, měl by přijít ke kódu, analyzovat modul, ve kterém je potřeba provést změnu a implementovat ji.

Kapitola 5

Návrh řešení

Před samotnou implementací aplikace, se objevilo několik otázek a možností, které bych rád zmínil. Než bych ale začal rozebírat tyto problémy, popsal bych Vás nejprve architekturu portálu a aplikací, s kterými komunikuje. Hned poté bych blíže rozebral životní cyklus a můj návrh portletu, protože se často na tyto souvislosti budu v textu odkazovat.

5.1 Architektura

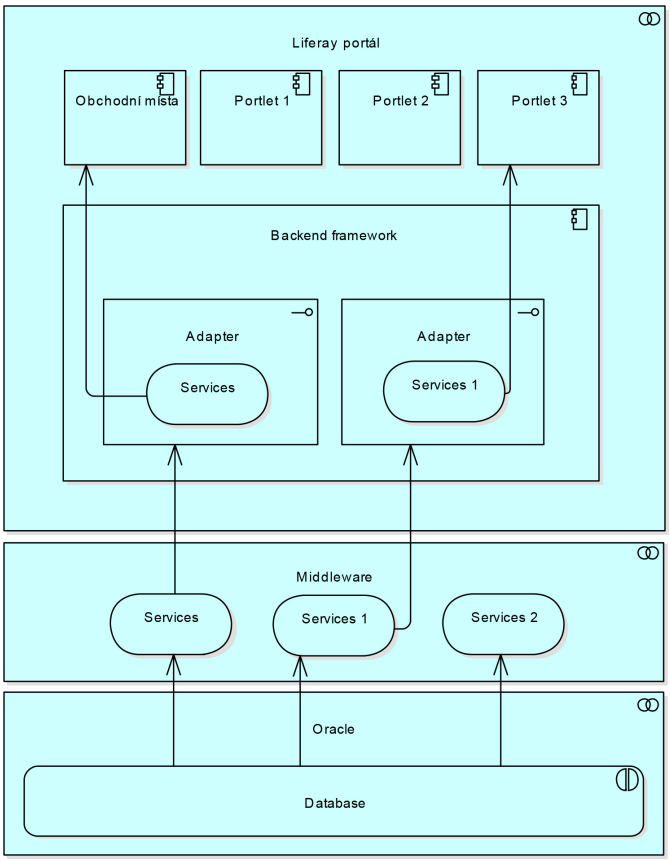
Architektura tohoto systému je rozdělena do tří hlavních celků, které se dále dělí a jsou daleko složitější. Jelikož se tato práce jimi nezabývá, tak je popíši zjednodušeně a povrchově. Pro nás bude nejdůležitější samotný portlet, který více rozeberu níže.

Začneme úplně spodní vrstvou a tou je relační databáze Oracle. Tato databáze obsahuje tabulky, kde jsou uložena veškerá data o obchodních místech.

Nad touto vrstvou je postaven middleware, vytvářející DAO *(Data access object)* což jsou objekty, které poskytují abstraktní rozhraní databáze vyšší vrstvě.

Poslední vrstva je už samotné portálové řešení. Pro nás to je konkrétně Liferay portal, který podporuje všechny známé aplikační servery, operační systémy a databáze. Pro vývoj jsem použil volně dostupný Apache Tomcat. O něm více v 6.1. Mezi základní funkce Liferay portálu můžeme zařadit: [12]

- tvorba jednotlivých stránek
- vícejazyčné uživatelské rozhraní
- zabezpečená SSO(Single Sign On) autentizace
- nástroje pro správu pracovních postupů
- personalizace webových stránek



Obrázek 5.1: Architektura portálu a dalších částí systému

- snadné uspořádání prvků/portletů pomocí *drag&drop*
- tagování a vyhledávání v obsahu portálu
- zobrazování obsahu v závislosti na definovaných rolích a oprávněních
- monitorování výkonu portálu
- vytváření článků a jejich publikace na portálu

Na Liferay portálu se nachází bankovní framework, který poskytuje rozhraní služeb jednotlivým portletům. Adaptér zajišťuje transparentnost služeb z middlewaru.

Poslední věc, kterou portál obsahuje, jsou právě zmiňované portlety. Tyto webové komponenty jsou vyvíjeny tak, aby veškerá backendová logika byla ve výše zmiňovaném frameworku. Tedy přístup k informacím, které jsou uložené v databázi, využívání služeb třetích stran, odesílání emailů a mnoho dalších.

Tento způsob má několik výhod, vývojář portletu je odstíněn od integračních a databázových problémů. Probíhá další validace informací, přenos dat je tedy bezpečnější.

Naopak nevýhody jsou vysoká režie, která vzniká kvůli marshallingu[4] objektů. Dále zdlouhavý proces malé změny v rozhraní frameworku.

5.2 Portlet

Dříve než se budu věnovat návrhu portletu, bych rád zmínil jeho životní cyklus od doby nahrání portletu na portál, až po dobu, kdy bude z portálu vymazán.

5.2.1 Životní cyklus

Životní cyklus portletu se dá rozdělit do šesti fází. Po celou dobu cyklu se o portlet stará a řídí ho portletový kontejner.

1. init
   První z fází je inicializační, metoda **init** se zavolá, pokud je portlet indikován kontejnerem. Tato metoda je zavolána pouze jednou a to při nasazení portletu na portál.

2. render
   Pokud uživatel přijde na stránku, kde se nachází portlet, zavolá se právě tato metoda. **Render** metoda je zodpovědná za generování HTML obsahu stránky, který je založen na aktuálním stavu portletu.

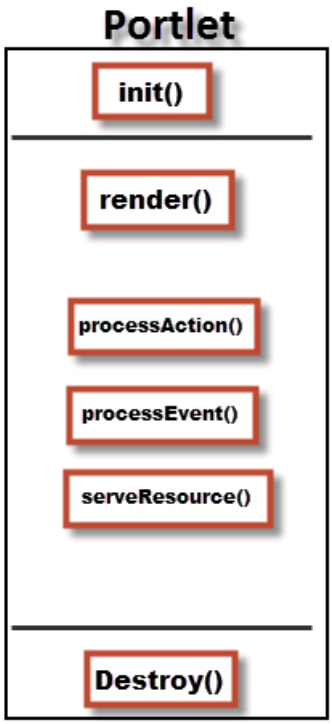
3. processAction
   Metoda **processAction** je volána při odpovědi na podnět od uživatele, nejčastěji kliknutím myší na odkaz nebo odesláním formuláře. Hned po dokončení této metody portletový kontejner zavolá metodu **render**.

4. processEvent
   Event metoda je zavolána na jakoukoliv událost, která v portletu nastane. Též hraje důležitou roli v událostech, založené na vnitřní komunikaci portletu. Stejně jako u metody **processAction**, po dokončení metody, portletový kontejner zavolá metodu **render**.

5. serveResource
   Tato metoda je také volána portletovým kontejnerem a stará se o generování obsahu, který není založen na HTML. Jde o formát výstupu typu JSON, XML atd. Pomocí této metody se implementují Ajaxové volání.

[4]Marshalling je proces transformace paměťově reprezentace objektů nebo dat, který je vhodnější pro uložení nebo přenos mezi aplikacemi.

6. destroy
   Metoda **destroy** je volána při odstranění portletu z portálu. Ve výchozí implementaci je prázdná, tedy nic nedělá. O zavolání této metody se stará portletový kontejner.



Obrázek 5.2: Životní cyklus portletu [1]

5.3 Návrh architektury portletu

Při návrhu architektury jsem portlet rozdělil na dvě vrstvy.

5.3.1 Servisní vrstva

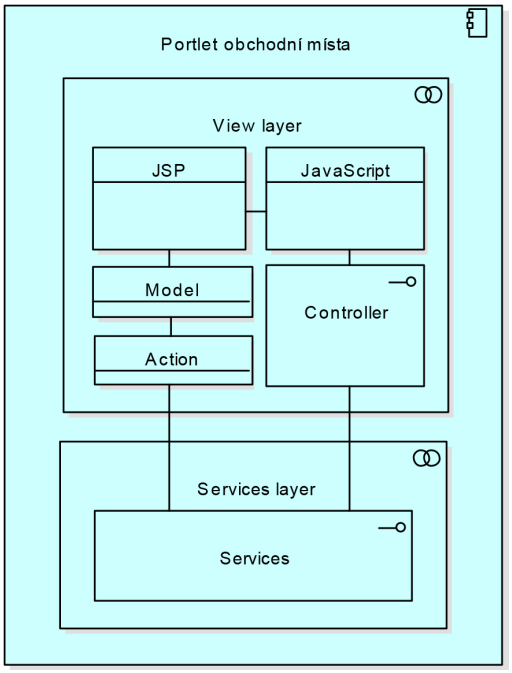
Servisní vrstva má na starost poskytování prezentační vrstvě rozhraní, které bude moci rovnou použít k vykreslování obchodních míst. Toto rozhraní má dvě různé implementace. První implementuje rozhraní, které poskytuje framework viz. 5.1. Druhá implementace je tvořena nástrojem *mock*.

5.3.2 Prezentační vrstva

Pří příchodu na stránku s portletem se vytvoří objekt *Action*. Tento objekt získá z Liferay databáze aktuální nastavení portletu. Pokud žádné neexistuje, vytvoří si vlastní s výchozími hodnotami. Poté na základě konfigurace pošle požadavek na servisní vrstvu, která mu vrátí požadovaná data. Tyto data dále zpracovává a plní model informacemi, které potom vykreslí do JSP stránky.

Některé informace jsou rovnou předány JavaScriptu ve formátu JSON- například konfigurace knihoven. Tyto objekty se předávají rovnou knihovnám při inicializaci mapy.

Další z funkcí prezentační vrstvy je poskytování rozhraní kontroléru, který obsahuje metody pro ajaxové volání z JavaScriptu. Veškeré informace, které kontrolér vrací JavaScriptu, čerpá ze servisní vrstvy zmíněné výše. Například pokud JavaScript potřebuje získat více informací o daném obchodním místě, pošle ajaxový požadavek kontroléru, ten ho zpracuje a vrátí výsledek.



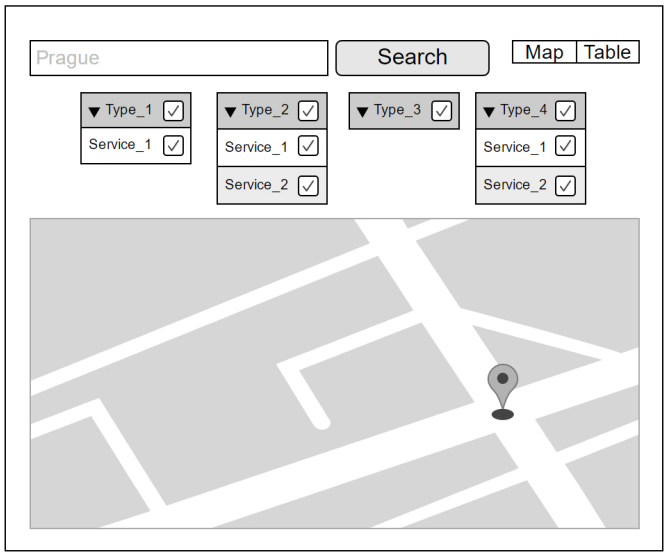
Obrázek 5.3: Architektura portletu

5.4 Návrh obrazovek

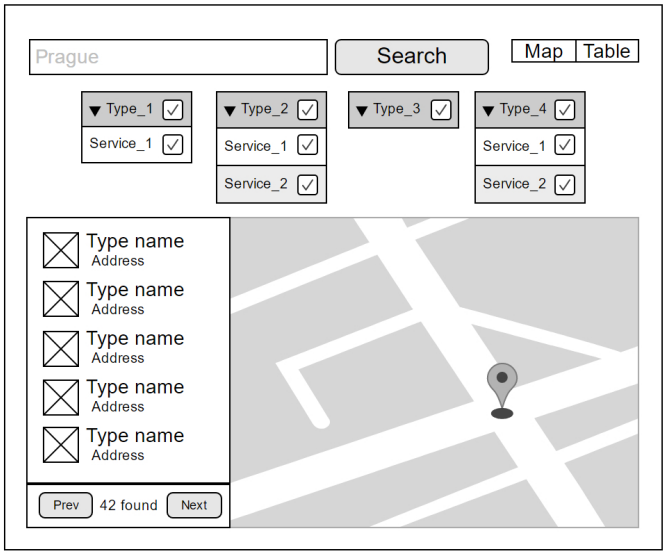
Při návrhu obrazovek jsem vycházel ze současného portletu. Každé zobrazení portletu obsahuje vyhledávací a filtrovací panel. V horním pravém rohu jsou přepínací tlačítka, pro přepnutí zobrazení.

5.4.1 Zobrazení mapy

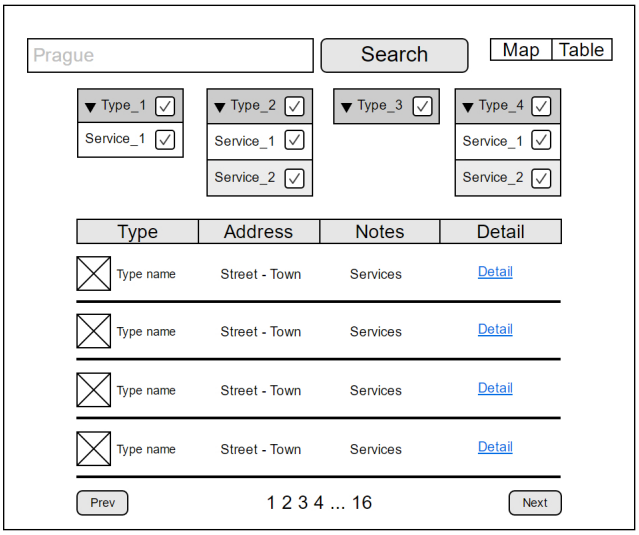
Při přiblížení na určitou úroveň mapy se zobrazí boční tabulka.



Obrázek 5.4: Hlavní obrazovka v zobrazení mapa



Obrázek 5.5: Zobrazení boční tabulky



Obrázek 5.6: Hlavní obrazovka v tabulkovém zobrazení

5.4.2 Tabulkové zobrazení

5.5 Načítání dat

Jeden z hlavních problémů byl, jakým způsobem načítat data obchodních míst. Oproti stávajícímu portletu potřebujeme v novém znát více informací už při vytváření mapy kvůli požadavku 3.1.1.

Provedl jsem tedy několik návrhů řešení, ty jsem poté otestoval a změřil. Při každém testu jsem načítal přesně 8000 obchodních míst.

Výsledné hodnoty testování se můžou zdát příliš vysoké. Je to způsobeno průchodem dat přes několik vrstev kvůli bezpečnosti. Také všechna prostředí jsou testovací, nejedná se tedy o produkční servery, které jsou mnohonásobně rychlejší, ani o produkční databázi.

5.5.1 První způsob

Prvním naivním způsobem je načítání všech informací najednou. Tedy požadujeme od backend frameworku kolekci osmi tisíc objektů, kde každý objekt obsahuje všechny informace ke konkrétnímu obchodnímu místu. Při této metodě mi odpověď od frameworku přisla v průměru po 24 603 milisekundách.

Toto řešení tedy určitě není vhodné, žádný uživatel nebude čekat 25 sekund, než se mu načte stránka s portletem.

5.5.2 Druhý způsob

Druhým způsobem je načítání pouze potřebných informací při přístupu na portlet. To jsou informace sloužící k filtrování a zobrazení pinů, tedy *typ, pozice, služby, parametry a identifikátor*. Doba odpovědi této služby trvá přibližně 2 443 milisekund.

Po určitém přiblížení, kdy se zobrazí boční tabulka u mapy, je potřeba načíst dvě nové informace - město a adresu obchodního místa. Z těchto dvou údajů se skládá název a je zobrazen v tabulce. Potřebujeme tedy zavolat ajaxově další službu, která nám vrátí informace ke každému obchodnímu místu, které se aktuálně nachází ve výřezu mapy. Tato služba zabere v průměru kolem 1 896 milisekund.

Tento způsob je o mnoho rychlejší oproti prvnímu způsobu, co se týče načítání stránky. Problém nastává až při situaci, kdy je zobrazena tabulka. Služba vracející obchodní místa ve výřezu mapy se volá po každém posunutí mapy, či dalšímu přiblížení. Díky vysoké režii by bylo vhodné volat backendové služby co nejméně.

5.5.3 Třetí způsob

Tento způsob je založen na myšlence druhého způsobu, tedy dotahovat informace, až když je opravdu potřebujeme. Rozšiřuje akorát původní volání o dvě nové proměnné, a to adresu a město. Tyto dvě nové informace samotný objekt o moc nezvětší oproti například službám a parametrům. Doba odpovědi je tedy podobná, jako u druhého způsobu a to 2 608 milisekund.

Tato úprava řeší i časté volání backendu. Problém nastává tehdy, pokud uživatel bude chtít kliknout na pin a zobrazit si bublinu obchodního místa, nebo přepnout portlet na tabulkové zobrazení. Tehdy bude potřeba zavolat backend. Při zobrazení bubliny bude stačit služba, která vrací detail obchodního místa na základe identifikátoru. Zavolání a odpověď této služby trvá přibližně 200 milisekund.

Při tabulkovém zobrazení obchodních míst se zavolá stejná služba, jako při druhém způsobu při přiblížení mapy.

Tento způsob na základě výsledků měření vyšel jako nejlepší, proto jsem ho implementoval.

5.6 GoogleMaps knihovny

Jedna ze zásadních otázek u nového portletu byla taková, zda bude potřeba placená verze GoogleMaps knihoven. Z hlediska funkcionality portlet nevyužívá žádné placené rozhraní, či funkce od společnosti Google, a.s.

Zbývá tedy limitní omezení počtu zobrazení mapy a počet požadavků na služby GoogleMaps. Tyto omezení jsou:

- Počet zobrazení mapy nesmí přesáhnout 25 000 načtení během 24 hodin.
- Požadavky na služby GoogleMaps jsou omezeny na 150 000 požadavků za jeden den.

Na základe statistik Google během posledních dvou let, se tyto limity ani jednou nepřesáhly s rezervou 40%. Je tedy možné použít tyto knihovny v bezplatné verzi. [13]

Kapitola 6

Implementace

V implementační části popíši postup vývoje v pořadí, v jakém jsem při implementaci aplikace postupoval. Nebudu zabíhat do přílišných detailů, pouze bych zmínil jednotlivé myšlenkové pochody a problémy, na které jsem během vývoje narazil.

6.1 Použité technologie

Následující kapitola obsahuje souhrn technologií, které jsem při vývoji používal. Ostatní technologie zůstávají stejné, jako v kapitole 2.3.

6.1.1 SVN

Jednou z nejdůležitějších věcí při vývoji jakékoliv aplikace, je použití nějakého nástroje pro verzování zdrojových kódů. Já jsem při své práci používal SVN. Tento nástroj umožňuje organizovat projekt a spravovat jeho verze. Projekt je fyzicky uložen na souborovém serveru, takže při výpadku disků na svém lokálním počítači, nepřijdu o všechny svá data.

6.1.2 Apache Tomcat

Jak už jsem zmiňoval na začátku, portlet bude umístěn na portálu Liferay. Je tedy potřeba samotný Liferay nainstalovat na nějaký aplikační server. Pro moje potřeby jsem zvolil server Apache Tomcat, který vyniká jednoduchostí, transparentností a nižší náročností na výpočetní výkon. Je tedy zpravidla méně robustnější, než jeho komerční konkurenti, například WebSphere od společnosti IBM, které jsou větší a bezpečnější.

6.1.3 Apache Maven

Tento nástroj slouží ke správě a automatizaci sestavení aplikací. Je založen na konceptu takzvaných pom*(Project Object Model)* souborů. Tyto soubory mají

strukturu XML, která definuje jednotlivé části projektu a jeho závislosti na externích knihovnách a nástrojích. Pokud je projekt složen z více dílčích modulů, každý z nich má pak svůj pom soubor, který dědí vlastnosti od nadřazeného souboru. Díky této struktuře je pak možné sestavit celý projekt jediným příkazem. [14]

6.2 Porovnání knihoven

Než bych se pustil do rozebírání samotné implementace, vrátil bych se k jednomu z požadavků, a to je požadavek na shlukování pinů na mapě. Pro tento požadavek jsem se rozhodl otestovat různé knihovny, které shlukování zajišťují. Porovnat je z pohledu funkčnosti, rychlosti a zvolit tu nejlepší pro naše potřeby.

Porovnával jsem všechny knihovny, které podporují aktuální verzi GoogleMaps JavaScript Api, umožňují změnit ikony shluku a nastavit minimální úroveň, od které se už daná místa nebudou nadále shlukovat.

Veškeré testování probíhalo v prohlížeči Google Chrome ve verzi 58 a počet shlukovaných pinů se rovnal 500.

MarkerManager v3
První z knihoven, kterou jsem vyzkoušel, byla MarkerManager. Práce s touto knihovno byla velmi jednoduchá. Dokázala přidávat a mazat piny, jak po jednom tak celé pole. Nevýhodou této knihovny byla strohá nastavitelnost chování shlukování a ikon shluků a také je tato knihovna velmi pomalá. Inicializace mapy trvala 1639 milisekund.

Clusterer2
Další z knihoven je Clusterer2. Tato knihovna exceluje v rychlosti načítaní pinů, a to v čase 29 milisekund. Je to zapříčiněno tím, že knihovna zobrazuje pouze ty piny, které jsou vidět ve výřezu mapy, tedy pokud mapu posuneme, proběhne načítání znovu. Tato vlastnost není moc uživatelsky přívětivá. Další nevýhodou této knihovny je, že pokud se má zobrazit najednou opravu velké množství pinů, knihovna umístí doprostřed mapy jeden shluk, který obsahuje všechny tyto piny.

MarkerClusterer
Poslední z knihoven, které jsem testoval, byla právě knihovna MarkerClusterer. Tato knihovna je velmi rozšířená mezi vývojáři, je k ní tedy hodně dokumentace a návodů. Nastavitelnost a funkčnost této knihovny je dostačující pro naše potřeby. Co se týče rychlosti načtení pinů, tak tato knihovna je dokáže načíst za 187 milisekund.

6.2.1 Shrnutí

Ke své implementaci jsem zvolil knihovnu **MarkerClusterer**, protože splňovala všechny požadavky, má velkou uživatelskou základnu, která ji neustále rozšiřuje.

6.3 Servisní vrstva

Jako první jsem začal implementovat servisní vrstvu viz. 5.3.2, kterou jsem realizoval jako samostatný modul v Mavenu.

6.3.1 Rozhraní služeb

Dále jsem pokračoval s vytvořením rozhraní, které budu využívat v prezenční vrstvě. Pro toto rozhraní jsem vytvořil dvě implementace. Jedna, která volá služby backendového frameworku a druhou, která je implementována pomocí mocku, tedy testovacích dat, které vytvářím přímo v implementaci. Tento koncept mi zajišťuje nezávislost na frameworku při vývoji portletu. Pokud například bude nedostupný server s databází, budu moci stále vyvíjet nad daty z mocku a nestarat se tak o problémy s integrací.

6.4 Prezentační vrstva

Jak už jsem zmiňoval na začátku své práce, portlet je postaven na frameworku Spring, konkrétně Spring Web Flow. Při implementaci prezentační vrstvy jsem vytvořil nový Maven modul, který má standardní MVC strukturu.

Na následujících řádcích Vám popíši jednotlivé balíčky(*angl. packages*), tak jak jsou v pořadí na obrázku struktury prezentační vrstvy 6.4.

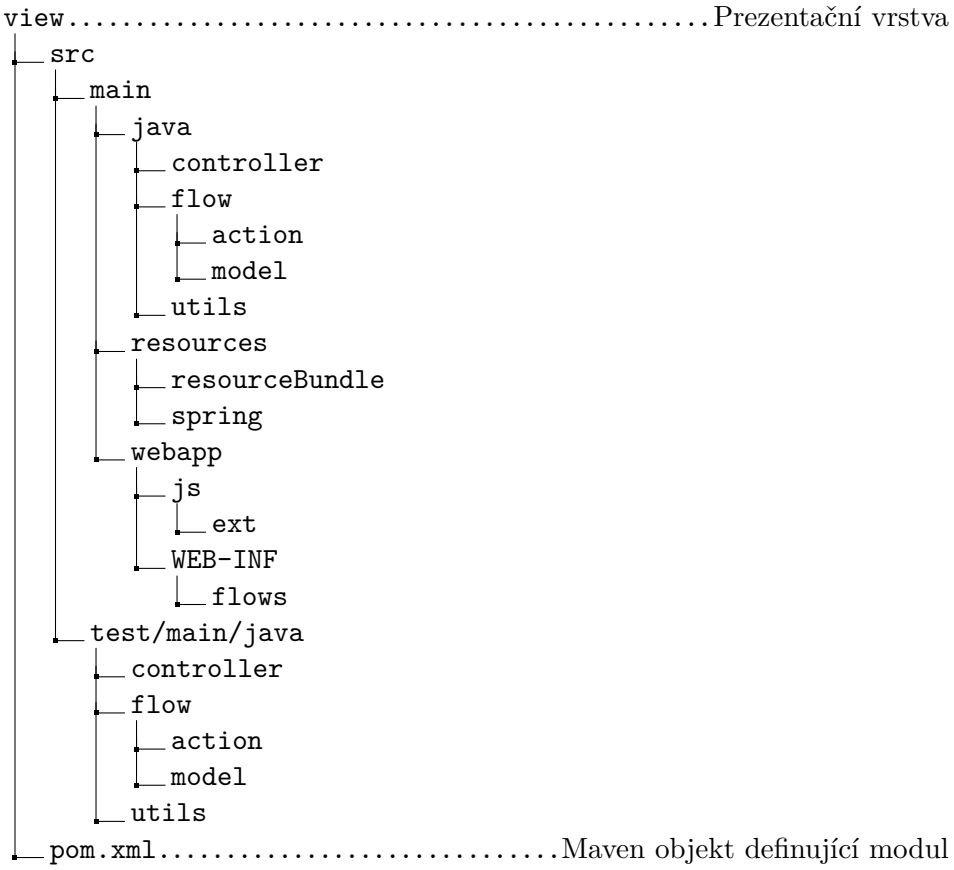
6.4.1 Controller

První balíček obsahuje kontroléry. Pro naše potřeby nám stačí pouze jeden kontrolér, který poskytuje dvě služby pro ajaxové volání.

První služba vrací veškeré informace o obchodním místě na základě parametru identifikátor.

Druhá ze služeb zohledňuje hned několik parametrů. Hlavním parametrem jsou souřadnice výřezu mapy resp. levý horní a pravý spodní roh, podle kterých se dopočítá plocha, kde se daná obchodní místa nachází. Návratovou hodnotou této služby jsou tedy všechna obchodní místa, která splňují všechny parametry a jejich pozice je v daném výřezu. Tato služba se používá pouze při tabulkovém zobrazení.

Všechny parametry, které tyto služby přijímají, jsou potřeba pečlivě validovat z důvodů bezpečnosti. Tyto služby totiž může zavolat každý uživatel webu a posílat jim nežádoucí data.



Obrázek 6.1: Struktura prezentační vrstvy

6.4.2 Flow

Následující balíček se dělí na dvě části: **action** a **model**. Začneme částí **model**. Tato část obsahuje hlavní a konfigurační model. Hlavní model obsahuje data, která se zobrazují v prezentační vrstvě MVC architektury. Konfigurační model nese informace o aktuálním nastavení portletu. Jeho stav se ukládá do databáze a o to se stará druhá část tohoto balíčku.

Část **action** obsahuje mimo jiné metody pro ukládání a načítání konfigurace. Metoda pro načítání používá unikátní klíč, podle kterého načítá data z databáze a vrací přímo konfigurační model. Ukládání probíhá analogicky. Jako parametr metody je samotný model, který se serializuje do json zápisu a zapíše se do databáze. Všechny tyto služby jako načítání a ukládání konfigurace, poskytuje Liferay a jeho objekt *PortletPreferences*. Tato část obsahuje také metodu pro přípravu a naplnění hlavního modelu daty, a metodu, která vrací všechny obchodní místa, která se mají zobrazit na základě nastaveného filtru v konfiguraci portletu.

6.4.3 Utils

Balíček *utils* obsahuje převážně statické třídy, které dělají jednu činnost a zpřehledňují tím ostatní kód. Nachází se zde třída, která na základě parametru lokalizace vrací překlad textu. Dále třída, která má na starosti transformování formátu otevíracích hodin obchodního místa z formátu vhodného pro ukládání do formátu čitelného pro běžného uživatele.

6.4.4 Resource bundle

V této sekci jsou veškeré textace a jejich překlady, které se objevují v portletu. Výhoda tohoto konceptu je mít veškeré textace na jednom místě a WCM si je může upravovat podle libosti.

6.4.5 Spring

Tento balíček obsahuje konfiguraci springu. Definují se zde jednotlivé vrstvy, aplikační kontext a cesty k XML souborům definující průchod portletem. Dále se zde nastavuje mapování na kontroléry a jakým způsobem se bude zobrazovat obsah portletu. V našem případě to je JSP technologie.

6.4.6 JavaScript

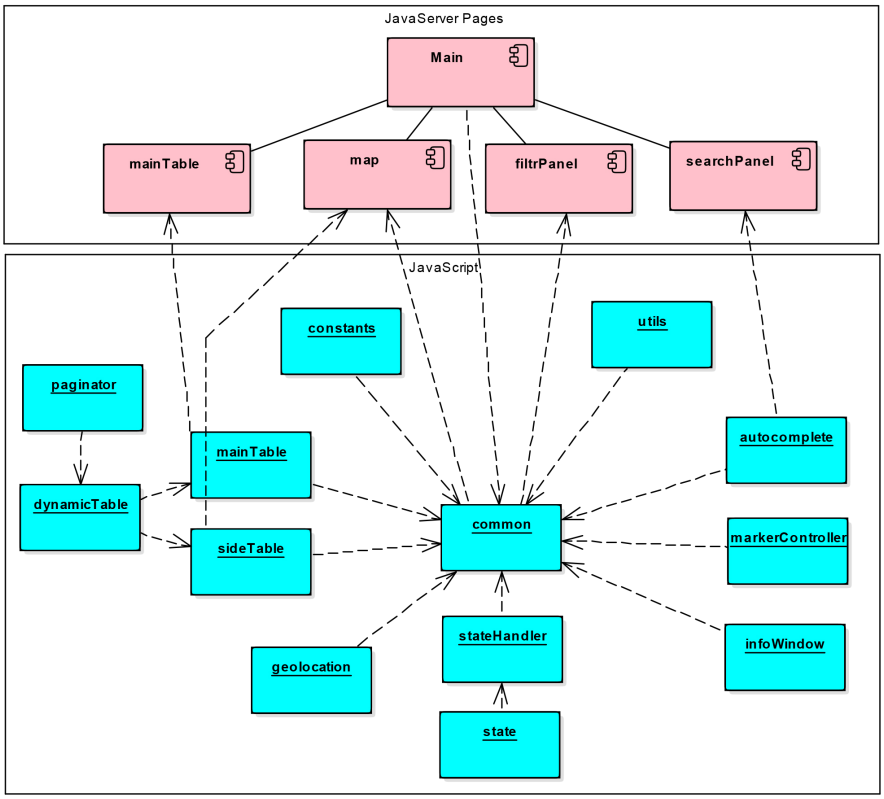
Balíčkem *js* začíná prezentační část MVC patternu. Je to taky jeden z největších balíčků teto práce, protože všechna logika je psaná pravě v JavaScriptu.

Jak už jsem zmiňoval v cílech této práce, snažil jsem se zaměřit na to, aby JavaScriptový kód byl co nejvíce přehledný a snadno se udržoval. Pokud by například přišel jiný programátor a chtěl by rozšířit URL o další parametr, aby byl pro něj intuitivní a hned věděl, kde co má hledat a nemusel nastudovat celý kód aplikace. Rozdělil jsem tedy jednotlivé komponenty do více souborů a zapouzdřil jsem jednotlivé objekty v JavaScriptu, jak můžete vidět na následujícím diagramu 6.4.6.

V následujících bodech popíši pouze důležité objekty, které mají vliv na celé fungování aplikace.

Common Objekt *common* je taková hlava celého portletu. Pří načítání stránky tento objekt vytvoří hlavní JSP soubor a předá mu aktuální konfiguraci a všechny obchodní místa. Na základě těchto informací vytvoří instance ostatních objektů a předá jim potřebná data. Poté začne s inicializací mapy a knihovny MarkerClusterer, které už předá vyfiltrovaná obchodní místa pomocí objektu *markerController*.

MarkerController Tento objekt při vytvoření instance převezme celý seznam obchodních míst. Poté na základě nastaveného aktuálního filtru vrací požadovaný seznam obchodních míst. Jelikož knihovna MarkerClusterer neposkytuje žádné služby, které by umožňovaly efektivně



Obrázek 6.2: Závislosti objektu v JavaScriptu

filtrovat piny, je potřeba všechny piny smazat a nahrát nové. Naštěstí tato operace je velmi rychlá, trvá v řádech desítek milisekund.

StateHandler a State Objekt *stateHandler* má dva úkoly. První z nich je uchovávat aktuální stav portletu se stavem v URL. K tomu slouží objekt *state* a jeho vlastnosti(angl. property), který se serializují do adresy prohlížeče. Další z úkolů *stateHandleru* je synchronizovat všechny procesy nad mapou, obsahuje tedy metody na přidání a odebrání zámku. Tyto metody zajišťují, že pokud uživatel bude rychle operovat nad mapou, tak nedojde k nekonzistenci stavu v URL a aktuálního stavu.

6.4.7 WEB-INF

Další z balíčku je *WEB-INF*. Tento balíček obsahuje nastavení portletu pro portál Liferay. Jsou zde XML soubory, ve kterých můžeme nastavit jméno portletu a jakým způsobem se bude portlet zobrazovat v administraci Liferaye. Je zde také soubor pro nastavení načítání JavaScriptových knihoven a friendly URL adres.

6.4.8 Flows

Balíček *flows* nacházející se v prezentační vrstvě MVC konceptu, obsahuje JSP soubory, které generují HTML obsah stránky. Lze do nich psát přímo Java kód. Dále obsahuje jeden z nejdůležitějších souborů portletu a to soubor XML definující přechody obrazovek v portletu. Tento soubor je jeden z vlastností frameworku Spring Web Flow.

6.4.9 Test

Poslední z balíčků je test, který kopíruje strukturu *main* a obsahuje unit testy na celý Java kód, který se zde nachází. K testování Java kódu jsem použil dva testovací frameworky.

TestNG
Tento testovací framwork je založen na JUnit a NUnit, které rozšiřuje o nové funkcionality, například o anotace, které tento testovací framework dělají snadnějším na používání. [15]

JMockit
JMockit je nejčastěji používán s JUnit nebo s TestNG frameworkem. Poskytuje API pro mokování a falšování služeb od integračních částí. Pokud tedy potřebujeme otestovat funkci, která využívá nějakou integrační službu, kterou nemáme v testu k dispozici, pomocí technologie JMockit ji můžeme přepsat tak, že pokud ji náš kód zavolá, daná služba vrátí námi přednastavená data. [16]

Závěr

V práci jsem se zabýval analýzou současného stavu portletu, sběrem požadavků na nový portlet, analýzou těchto požadavků, návrhem řešení a následnou implementací a otestování nového portletu.

Nově vytvořený portlet obsahuje všechny funkční i nefunkční požadavky, které jsem sesbíral. Rychlost tohoto portletu je srovnatelná se současným portletem, a to i přes nové požadavky, které vyžadují načítání mnohonásobně většího počtu dat už při samotném načítání stránky.

Co se týče konfigurovatelnosti, tu jsem rozdělil do dvou částí. První část slouží pro základní konfiguraci modulů, jako je zobrazení, výchozí hodnoty a obsah. Jsou zde také věci ohledně identifikace majitele portálu, kvůli statistikám od GoogleMaps. Druhá část konfigurace je pro pokročilé, lze v ní nastavit detaily shlukování, jako je průměrná odchylka při výpočtu pozice shluků a mnoho dalších věcí.

S konfigurovatelností souvisí i modularita a celková přehlednost kódu. Nový portlet jsem rozdělil do čtyř modulů: vyhledávací a filtrační panel, mapa a tabulka. Všechny tyto moduly jsou kódově zapouzdřeny a lze je konfigurovat.

V budoucnosti by bylo možné portlet rozšířit o nové funkce či moduly, které by například umožňovaly uživatelům sjednat si schůzku na vybraném obchodním místě, nebo například zobrazovat uživateli personalisované nabídky služeb.

Myslím si, že se mi podařilo dosáhnout všech cílů, kterých jsem chtěl v rámci této práce dosáhnout. Výsledek mé práce bude užívaný v praxi reálnými uživateli, prošel akceptací businessem a penetračními testy banky. Podařilo se mi tedy splnit hlavní cíl této práce, a to nahradit současný portlet novým.

Literatura

[1] Nilang: Introduction to Portlet Phases and Lifecycle Methods. December 16, 2014, [cit. 2017-05-14]. Dostupné z: http://www.opensource-techblog.com/wp-content/uploads/2014/12/lifecycle-2Bmethods.png?x24105

[2] Liferay, Inc.: *Portal Features* [online]. c2017, [cit. 2017-05-14]. Dostupné z: https://web.liferay.com/products/liferay-portal/features/portal/

[3] Hepper, S.: Introducing the Portlet Specification. *JavaWorld* [online], August 1, 2003, [cit. 2017-05-14]. Dostupné z: http://www.javaworld.com/article/2073645/soa/introducing-the-portlet-specification--part-1.html

[4] Defining Technology, I.: What is Middleware? *Middleware Resource Center* [online], 2008, [cit. 2017-05-14]. Dostupné z: http://web.archive.org/web/20120629211518/http://www.middleware.org/whatis.html

[5] Liferay, Inc.: *AJAX Introduction* [online]. c2017, [cit. 2017-05-14]. Dostupné z: https://www.w3schools.com/xml/ajax_intro.asp

[6] Berners-Lee, M. . M.: Uniform Resource Locators (URL). *Network Working Group* [online], December 1994, [cit. 2017-05-14]. Dostupné z: https://tools.ietf.org/html/rfc1738

[7] Oracle Corporation: *Java SE at a Glance* [online]. c2017, [cit. 2017-05-14]. Dostupné z: http://www.oracle.com/technetwork/java/javase/overview/index.html

[8] Oracle Corporation: *Java EE at a Glance* [online]. c2017, [cit. 2017-05-14]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[9] Horáček, P.: Java na webu IV. - JSP. *LinuxSoft* [online], June 16, 2013, [cit. 2017-05-14]. Dostupné z: `http://www.linuxsoft.cz/article.php?id_article=1978`

[10] The jQuery Foundation: *The jQuery Foundation Mission* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `https://jquery.org/`

[11] The jQuery Foundation: *What's New in jQuery UI 1.12?* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `https://jqueryui.com/`

[12] Liferay, Inc.: *Liferay DXP: Technical Specifications* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `https://www.liferay.com/product/tech-specs`

[13] Google a.s.: *Google Maps JavaScript API Usage Limits* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `https://developers.google.com/maps/documentation/javascript/usage`

[14] The Apache Software Foundation: *Introduction* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `http://maven.apache.org/what-is-maven.html`

[15] Beust, C.: TestNG - Welcome. *TestNG* [online], December 19th, 2015, [cit. 2017-05-14]. Dostupné z: `http://testng.org/doc/`

[16] jMockit.org: *About the JMockit Testing Toolkit* [online]. c2017, [cit. 2017-05-14]. Dostupné z: `http://jmockit.org/about.html`

Příloha A

Seznam použitých zkratek

WCM Web content management

BUS Business

HTML HyperText Markup Language

CSS Cascading Style Sheets

MVC Model-view-controller

AJAX Asynchronous JavaScript and XML

URL Uniform Resource Locator

XML Extensible markup language

API Application Programming Interface

JSON JavaScript Object Notation

JSP JavaServer Pages

SVN Subversion

Příloha **B**

Obsah přiloženého CD

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