FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh systému pro reporting parkovacích systémů
Student:	Bc. Jakub Zahradníček
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je navrhnout systém pro vytváření reportů parkovacích systémů. Systém bude obsahovat několik serverů parkovacího systému a centrální server cloudu.

1. Prostudujte možnosti návrhu celého systému (aplikační server, komunikační protokoly, databáze, cloud).
2. Zanalyzujte hlavní požadavky a funkcionality na systém (např.: lokální vytváření reportů přes webovou aplikaci na parkovacím serveru, lokální správa šablon reportů na parkovacím serveru, automatické generování požadavků na naplánované reporty a odesílání výstupů mailem).
3. Navrhněte architekturu systému a jeho rozdělení do modulů a protokoly jejich vzájemné komunikace.
4. Navrhněte architekturu jednotlivých modulů.
5. Implementujte komunikační části všech modulů včetně testovacích funkcí.
6. Implementujte podstatnou část výkonných modulů (prototyp) a funkcionality otestujte.
7. Proces návrhu a implementace zhodnoťte z pohledu nákladů a benefitů a navrhněte další rozvoj.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 10. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Návrh systému pro reporting parkovacích systémů

Bc. Jakub Zahradníček

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

9. ledna 2020

Poděkování

Na tomto místě bych rád poděkovat Ing. Petře Pavlíčkové, Ph.D., že se ujala vedení práce a za její čas, který mi věnovala. Zvláštní poděkování patří mé rodině za vytrvalou podporu po celou dobu mého 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. 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 9. ledna 2020 .....

Odkaz na tuto práci

Zahradníček, Jakub. *Návrh systému pro reporting parkovacích systémů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Práce se zabývá návrhem a implementací systému pro získávání, zpracování a prezentaci dat parkovacího systému společnosti GREEN Center. Systém se skládá ze dvou samostatných webových aplikací vzájemně komunikujících pomocí REST API. Aplikace jsou implementovány v jazyce Java za použití frameworků Spring a Bootstrap. Mezi další technologie použité k implementaci a nasazení aplikací patří například aplikační server WildFly, databázový server SQL Anywhere a několik externích knihoven. Výsledkem je prototyp systému, který umožňuje provozovatelům parkovacího systému společnosti GREEN Center manuální i automatické generování reportů, souhrnů a statistik.

Klíčová slova parkovací systém, webová aplikace, Java, WildFly, SQL Anywhere, Spring, Bootstrap, REST

Abstract

This thesis deals with the design and implementation of a system for GREEN Center parking system data acquiring, processing and presentation. The system consists of two separate web applications using REST API to communicate with each other. Applications are implemented in Java programming

language using frameworks Spring and Bootstrap. Other technologies used to implement and deploy applications include WildFly application server, SQL Anywhere database server, and several external libraries. The result is a prototype of the system that allows GREEN Center parking system operators to manually and automatically generate reports, summaries and statistics.

Keywords parking system, web application, Java, WildFly, SQL Anywhere, Spring, Bootstrap, REST

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V České republice je aktuálně registrováno přibližně 6 milionů motorových vozidel, celosvětově se jejich počet odhaduje na 1,3 miliardy a každým rokem se zvyšuje. Narůstající podíl využití automobilů v osobní přepravě s sebou přináší mnohé obchodní příležitosti a jednou z nich je provoz placených parkovišť. Zejména v centrech velkých měst bývá totiž parkovacích míst nedostatek a kromě soukromých provozovatelů se ke zpoplatnění možnosti parkování čím dál tím častěji uchylují i samotná města nebo jednotlivé městské části. Většina zpoplatněných parkovišť využívá některý z automatických parkovacích systémů zajišťující řízení provozu i výběr parkovného. Pro provozovatele je přitom důležité mít přístup k informacím o jednotlivých aspektech využití parkoviště a nejrůznějším detailním statistikám.

Cílem práce je návrh a implementace prototypu reportovacího systému pro parkovací systém společnosti GREEN Center. Účelem reportovacího systému je tvorba souhrnů a statistik z provozu parkovacího systému. Návrhu a implementaci předchází rešerše vhodných technologií týkajících se tématu a provedení analýzy uživatelských požadavků. Závěrečnými cíly stanovenými zadáním práce je řádné otestování implementace, zhodnocení procesu návrhu a implementace z pohledu nákladů a benefitů pro společnost GREEN Center a navržení dalšího rozvoje.

Práce je logicky rozdělena do několika kapitol. První z nich obsahuje popis společnosti GREEN Center, její činnosti a produktů. Druhá kapitola se zabývá popisem nejdůležitějších technologií, které připadají v úvahu pro využití při návrhu a implementaci, a srovnáním některých jejich zástupců. Třetí kapitola se zabývá identifikací funkčních a nefunkčních požadavků kladených na výsledný systém. Součástí kapitoly je rovněž analýza současného fungování parkovacího systému s ohledem na budoucí další možnosti využití cloud computingu. Čtvrtá kapitola se zabývá návrhem systému reportů na základě provedené analýzy a návrhem využitelnosti cloud computingu v parkovacím systému obecně. Návrh systému reportů obsahuje popis uživatelských

rolí, případů užití, návrh architektury společně s modelem nasazení a návrh způsobů komunikace jednotlivých komponent. Pátá kapitola popisuje způsob implementace a při ní použité technologie. Závěrečné dvě kapitoly se pak zabývají testováním, zhodnocením odvedené práce a možnostmi dalšího rozvoje.

Kapitola 1

Společnost GREEN Center

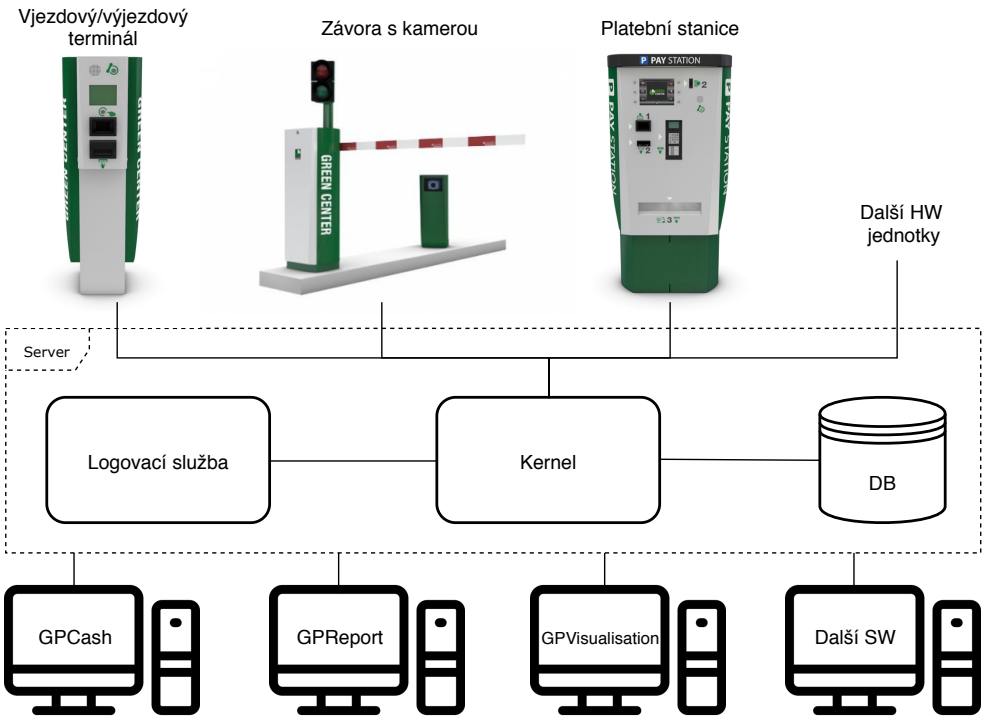
Kapitola obsahuje stručný popis činnosti společnosti GREEN Center, jejího parkovacího systému a nástrojů pro vytváření reportů. Cílem je uvést čtenáře do oblasti, ve které se společnost na trhu pohybuje a přiblížit mu alespoň základní principy fungování a architektury parkovacího systému a systému reportů, aby dokázal lépe pochopit obsah a cíle této práce.

1.1 Oblast činnosti

Společnost GREEN Center je specializovaným výrobcem, dodavatelem a prodejcem parkovacích a přístupových systémů. Kromě vývoje vlastního softwaru se zabývá i konstrukcí, výrobou a montáží hardwaru a dalších fyzických komponent parkovacího systému. Na českém i zahraničním trhu se společnost pohybuje již 27 let a za tuto dobu si dokázala vybudovat pozici nejvýznamějšího českého dodavatele parkovacích systémů, což dokazuje přibližně 55% zastoupení na tuzemském trhu a více než 30 zemí světa, ve kterých byly dosud systémy společnosti instalovány [1].

1.2 Parkovací systém

Parkovací systém, jak už název napovídá, slouží k zajištění a organizaci provozu parkoviště. Motivací provozovatele pro jeho instalaci je zajištění přístupu oprávněným a zamezení v přístupu neoprávněným osobám, usnadnění orientace návštěvníkům parkoviště, monitoring, a ve většině případů především zpoplatnění poskytovaných parkovacích služeb. Mezi typické zákazníky společnosti (provozovatele parkovišť) patří obchodní centra, hotely, nemocnice, zábavní parky, skiareály a podobně. Strandardně (v režimu krátkodobých karet) uživatel při vjezdu získá parkovací kartu, před odjezdem zaplatí u platební stanice příslušný poplatek, a výjezdový terminál se závorou mu následně umožní parkoviště opustit. Jedná se o nejjednodušší případ užití parkovacího



Obrázek 1.1: Architektura parkovacího systému

systému, který jinak poskytuje široké spektrum možností jeho využití – nabízí různé typy parkovacích karet a režimů využití parkovacích služeb, různé typy slevových karet, rozdělení parkovacích prostor do oddělených zón a řízení přístupů mezi nimi, takřka neomezené možnosti definování cenových tarifů a mnoho dalšího.

Parkovací systém společnosti GREEN Center je modulární a široce konfigurovatelný. Zjednodušená architektura je zobrazena na obrázku 1.1. Ve středu hvězdicového uspořádání se nachází server, na kterém běží software tvořící jádro celého systému (Kernel). Kernel je připojen k lokálnímu databázovému serveru a pomocí vlastního protokolu využívajícího TCP sockety na transportní vrstvě komunikuje s jednotkami obsaženými v konfiguraci dané instalace. Příkladem připojených jednotek mohou být vjezdové a výjezdové terminály, závory, kamery, platební automaty, dobíjecí stanice, komponenty naváděcího systému, informační tabule a další. Kromě hardwarových jednotek může dále Kernel komunikovat s několika aplikacemi poskytujících provozovatelům parkovacího systému určitý typ rozhraní pro ovládání systému, jeho kontrolu nebo získávání a zpracování dat o jeho využívání.

1.3 Reportovací systém

Data získaná z parkovacího systému představují pro jeho provozovatele důležité zdroje informací o využití daného parkoviště, jeho návštěvnících, nebo třeba zisku z porovozu a potenciálních možnostech jeho navýšení. Proto je důležité poskytnout provozovatelům systému nástroje k získání těchto dat a práci s nimi. Důležité je si zároveň uvědomit, že s těmito daty obvykle nepracují lidé na technicky zaměřených pozicích. Data v podobě, v jaké jsou uložena v databázi, pro tento účel rozhodně nejsou vyhovující. Je proto nezbytné je vhodným způsobem strukturovat a prezentovat.

1.3.1 Aktuální stav

V tuto chvíli existují dvě desktopové aplikace, které se týkají tvorby reportů z parkovacího systému.

První z nich je využívána technickou podporou a vývojáři společnosti GREEN Center. Poskytuje jejím uživatelům grafické rozhraní pro CRUD operace s položkami databáze souvisejícími s vytvářením samotných reportů (adresářové struktury, šablony reportů, překladové položky).

Druhá z aplikací je pak k dispozici provozovateli a obsluze parkovacího systému. Aplikace s grafickým rozhraním zobrazí uživateli v adresářové struktuře dostupné repory, které je možné vygenerovat. Po vybrání konkrétní šablony zadá uživatel vstupní parametry dané šablony, aplikace získá z databáze potřebná data a zobrazí výsledek uživateli. Následně umožňuje export dat do souboru v několika formátech a uložení na disk.

Obě aplikace využívají grafické knihovny Swing. Uživatelé si stěžují na zastaralý vzhled a rovněž nepřehledné a příliš složité grafické rozhraní. Problém desktopových aplikací představuje taktéž nutost jejich instalace a konfigurace na všech zařízeních, na kterých jsou využívány.

1.3.2 Vize společnosti

Záměrem je sloučit funkčnosti obou výše zmíněných aplikací a zpřístupnit je uživateli z webového prohlížeče. Zároveň by měl být backend aplikace (logika generování reportů) oddělen od frontendu, který by mohl být v budoucnu v případě potřeby nahrazen.

Kromě toho by měla být dále vytvořena webová aplikace přístupná z veřejné sítě, která umožní uživatelům vytvářet a plánovat pravidelně a automaticky spouštěné úlohy. Výsledkem těchto úloh by mělo být automatické vytvoření požadovaných reportů a jejich následné odeslání emailem.

Kapitola 2

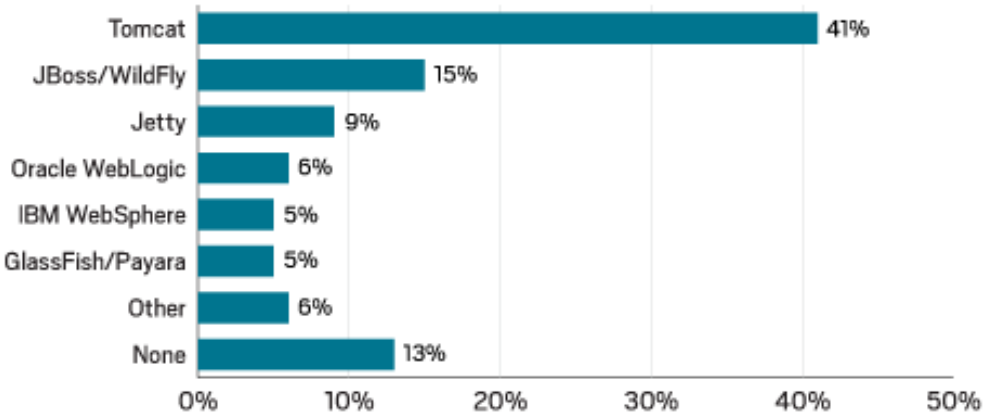
Teoretická část

Kapitola se věnuje popisu a porovnání některých zástupců nejdůležitějších technologií a softwarových produktů souvisejících s tématem práce. Konkrétní technologie skutečně použité v praktické části jsou shrnuty v sekci 5.1.

2.1 Aplikační server

Aplikační server je software, který vytváří prostředí pro spuštění aplikací, řídí jejich běh a tvoří mezivrstvu mezi operačním systémem a nasazenými aplikacemi. Kromě toho poskytuje aplikacím funkčnosti definované ve specifikaci Java Enterprise Edition, mezi které patří například specifikace pro vývoj webových aplikací (Java Servlet, JSP, JSF).

Pro účely nasazení aplikací připadají v úvahu především aplikační servery Apache Tomcat, Jetty, Glassfish a JBoss Application Server.



Obrázek 2.1: Statistika nejpoužívanějších Java aplikačních serverů [2]

Mezi nejpoužívanější aplikační servery (viz 2.1) patří, kromě výše zmíněných, ještě IBM WebSphere Application Server a Oracle WebLogic. Jedná se o komerční, robustní aplikační servery, jejichž výhod nedokáží menší aplikace plně využít, proto nebyly pro nasazení uvažovány a v následujícím srovnání nejsou zahrnuty.

2.1.1 Apache Tomcat

Zařazení Apache Tomcat mezi aplikační servery je častým tématem diskusí. Teoreticky se totiž o aplikační server nejedná. Problém je v definici aplikačního serveru jako implementaci standardu Java EE, který Apache Tomcat nesplňuje. Chybí například podpora EJB, JMS nebo distribuovaných transakcí. Namísto aplikačního serveru se tak v případě Apache Tomcat hovoří o HTTP webovém serveru nebo servlet kontejneru. Pro toto srovnání je tento fakt zanedbán a Tomcat, stejně jako Jetty je zařazen mezi plnohodnotné aplikační servery.

Apache Tomcat je podle průzkumů nejpoužívanějším aplikačním serverem pro nasazení Java webových aplikací. Jedná se o open source projekt, jehož velkou výhodou je právě rozšířenost a z toho vyplývající široká komunita uživatelů, kvalitní dokumentace a pravidelné aktualizace. Tomcat je zavedený, stabilní a ověřený mnoha lety používání. Díky chybějící podpoře některých Java EE standardů je navíc oproti dalším aplikačním serverům, jednodušší, snadněji konfigurovatelný i rychlejší například v případě spouštění nebo restartu.

Výhodou aplikačního serveru Tomcat je rovněž jeho flexibilita a rozšiřitelnost, díky níž mohou být na jeho základě postaveny aplikační servery doplňující Tomcat o některé chybějící funkce. Příkladem jsou TomEE, TomEE+ nebo i WildFly.

2.1.2 Jetty

V případě, že bychom trvali na formální definici aplikačního serveru, je i Jetty, stejně jako Apache Tomcat, pouze webový kontejner. Je vyvíjený společností Eclipse Foundation a rovněž se jedná o open source projekt, což není jejich jediná společná vlastnost.

Přestože, co se týče rozšíření, se vzájemně nedají příliš srovnávat, i tak je Jetty poměrně hojně používaný. Jeho hlavní výhody jsou podobné jako u Apache Tomcat a patří mezi ně kompaktnost, rychlost nebo malá velikost. Díky tomu je Jetty využitelný v případě omezených zdrojů, nebo jako součást jiného softwaru. Příkladem komplexnejšího softwaru, ve kterém je integrován aplikační server Jetty, může být Google App Engine nebo vývojové prostředí Eclipse.

2.1.3 WildFly

WildFly, dříve JBoss Aplication Server, je aplikační server vyvíjený společností Red Hat. Jedná se o celosvětově velmi úspěšnou a respektovanou společnost zabývající se vývojem softwaru. V roce 2019 bylo dokončeno její převzetí společností IBM za 34 miliard amerických dolarů [3].

Na rozdíl od dosud zmíněných serverů, v případě WildFly se již jedná o certifikovaný aplikační server plně podporující všechny specifikace Java EE. Jako takový je pochopitelně robustnější, pomalejší a náročnější na konfiguraci. Interně aplikační server Wildfly využívá servlet kontejner Tomcat, který doplňuje o jemu chybějící funkčnosti. Velkou výhodou je snadná migrace z Wildfly na JBoss Enterprise Application Platform s komerční podporou.

2.1.4 Glassfish

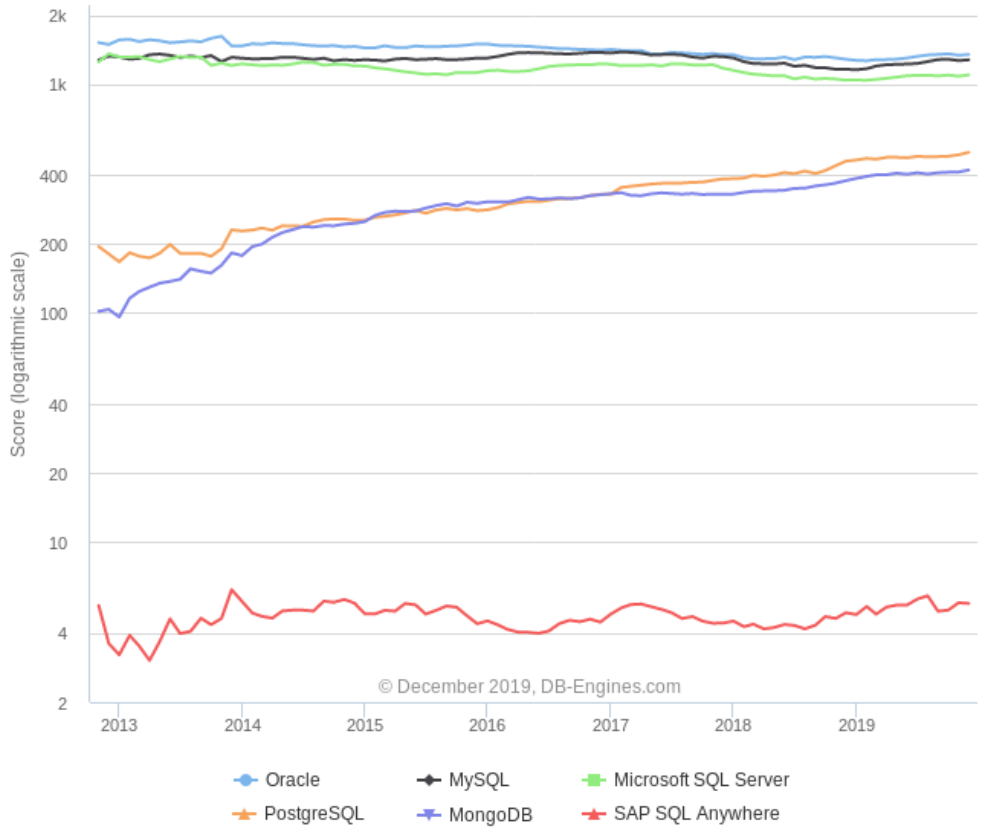
Glassfish je další plnohodnotný aplikační server od společnost Oracle, který na rozdíl od Wildfly využívá vlastní webový kontejner.

Výhodou aplikačního serveru Glassfish je, že se jedná o referenční implementaci standardu Java EE a používá se k prezentaci jejích funkčností. Tím je zaručeno, že funkčnosti Java EE jsou jako první implementovány a k dispozici právě v rámci aplikačního serveru Glassfish. Naproti tomu nevýhodou je absence komerční podpory, která byla společností Oracle ukončena v roce 2013 [4].

2.1.5 Srovnání

Z uvedených aplikačních serverů pochopitelně nelze vybrat jeden obecně nejlepší. Výběr by měl záležet na konkrétní situaci a individuálních potřebách daného projektu. V případě menších projektů, kde nejsou potřeba všechny funkčnosti Java EE je vhodný Apache Tomcat. V případě, že Tomcat některou z použitých technologií nepodporuje, je možné připojit odpovídající knihovnu a požadované funkčnosti doplnit. Pokud je předem známo, že Tomcat nepodporuje více funkčností než jsme ochotni tolerovat, nebo některá funkčnost není jako plugin k dispozici, je dobrou volbou aplikační server Wildfly.

V případě komerčních aplikačních serverů se jedná o robustní řešení využitelné v rozsáhlých projektech. Jejich výhodou oproti open source serverům je například propracovanější správa clusterů, vyvažování zátěže a eliminace výpadků. Jak již bylo ale zmíněno, těchto výhod ovbvykle nedokáží menší aplikace plně využít. Nevýhodou jsou pak pochopitelně náklady na provoz těchto systémů, které jsou pro většinu projektů nepřiměřeně vysoké.



Obrázek 2.2: Srovnání popularity databázových serverů [6]

2.2 Databázový server

V tomto kontextu je pojmem databázový server míněn systém řízení báze dat, nikoliv hardware serveru. Jedná se o software, který má na starosti zajištění práce s databází a poskytuje rozrhaní pro aplikace, které k databázi a jejím datům přistupují.

V současné době je v parkovacím systému využíván databázový server SQL Anywhere od společnosti SAP (dříve vyvíjeno společností Sybase), se kterým však s postupujícím časem nepanuje příliš velká spokojenost. Co se týče oblíbenosti a rozšíření, není na tom SQL Anywhere v porovnání s konkurencí příliš dobře (viz obrázek 2.2).

Přestože by se mohlo zdát, že popularita jednotlivých systémů se bude obtížně porovnávat, výsledné skóre databázových serverů na obrázku je odvozeno z několika jednoznačně kvantifikovatelných ukazatelů. Podrobný popis výpočtu je uveden na webu db-engines.com [7], základem hodnocení jsou následující ukazatele:

- počet zmínek na internetu měřený počtem výsledků vyhledávačů,

- počet zadaných dotazů ve vyhledávačích,
- počet souvisejících dotazů a zapojených uživatelů na Q&A webech jako je Stack Overflow,
- počet souvisejících pracovních nabídek,
- počet profilů na profesních sítích jako je LinkedIn, které server zmiňují,
- počet zmínek na sociální síti Twitter.

Do srovnání, vek kterém jsou zařazeny čtyři nejpopulárnějších databázové serverů z výše uvedeného pořadí, jsou zástupci jak proprietárního (Oracle database, Microsoft SQL), tak open source (MySQL, PostgreSQL) softwaru. Proprietární databázové systémy nabízí rovněž bezplatné Express edice s některými omezeními výkonu, které jsou zmíněny dále v textu. Záměrem těchto bezplatných edic může být oslovení společností v začátcích svých projektů, kdy omezení nijak nepocítí. Časem se může snadno stát, že bezplatná edice přestane požadavkům na výkon vyhovovat a zákazník je nucen přejít na placenou verzi, nebo projekt migrovat jiný open source systém, což může být náročné.

2.2.1 Oracle database

Společnost Oracle je jednou z předních společností zabývajících se vývojem relačních databází a nástrojů pro vývoj a správu databází. Produkty Oracle se těší velké oblíbenosti, což dokazuje i výše uvedená statistika, kde Oracle database obsadila první příčku. Jedná se o prověřený, robustní databázový systém, který vyniká výkonem, funkcionalitou nebo snadnou škálovatelností.

Kromě Enterprise edice poskytující plnou funkcionalitu nabící Oracle bezplatnou Express edici se základní funcionalitou a harwarovým omezením výkonu. Běh systému Oracle Database Express Edition je ve verzi 18c omezen na použití pouze dvou jader CPU, k dispozici je maximálně 2 GB operační paměti, velikost dat v databázi nesmí přesáhnout 12 GB a na databázovém serveru je možné spustit 3 samostané instance databáze [8].

Co se týče podpory operačních systémů, v minulosti byl systém Oracle database dostupný kromě platformy Windows jen na určitých distribucích Linuxu, například Oracle Enterprise Linux. V posledních letech došlo v tomto ohledu ke zlepšení a nejnovější verze systému již podporují i další platformy jako Mac OS, Solaris nebo HP-UX.

Oracle ke svým produktům poskytují službu My Oracle Support, v rámci které získá zákazník aktualizace softwaru a nepřetržitou technickou podporu. Tyto služby jsou pochopitelně dostupné pouze v případě placených edic. Uživatelé Express edice se musí spokojit s dokumentací v angličtině. K dispozici je rovněž podrobný instalační průvodce, tutorial pro začínající uživatele

nebo oficiální Oracle fórum. Nevýhodou Oracle database je v porovnání s konkurencí vyšší cena licencí i administrátorských kurzů [9, 10].

2.2.2 MySQL

MySQL je jeden z nejrozšířenějších open source relačních databázových systémů. Společně s operačním systémem Linux, webovým serverem Apache a programovacím jazykem PHP tvoří sadu open source technologií, známou pod zkratkou LAMP a používanou pro vývoj a nasazení webových aplikací.

Multiplatformní MySQL podporuje operační systémy Windows, Linux, Mac OS i Oracle Solaris. Mezi další výhody patří kvalitní online dostupná dokumentace v několika jazycích a oficiální diskuzní fórum na stránkách MySQL. Nevýhodou v porovnání s ostatními uváděnými databázovými systémy pak je neúplná implementace jazyka SQL, ve které chybí například konstrukce FULL JOIN.

Kromě bezplatné GPL licence je MySQL k dispozici také pod placenou komerční licencí. Komerční licence nabízí oproti bezplatné verzi nepřetržitě dostupnou podporu, přístup k nejnovějším aktualizacím nebo některé nadstandardní funkcionality a služby týkající se například monitorování, zálohování, bezpečnosti nebo správy clusteru.

2.2.3 Microsoft SQL Server

Microsoft SQL Server je proprietární databázový server vyvinutý společností Microsoft. Aktuálně nejnovější verze systému, Microsoft SQL Server 2019, je společností Microsoft nabízena v následujích edicích:

- SQL Server 2019 Express (1 CPU / 4 jádra, 1410 MB RAM, 10 GB dat),
- SQL Server 2019 Web (4 CPU / 16 jáder, 64 GB RAM, 524 PB dat),
- SQL Server 2019 Standard (4 CPU / 24 jáder, 128 GB RAM, 524 PB dat),
- SQL Server 2019 Enterprise (bez omezení s výjimkou množství dat – 524 PB),
- SQL Server 2019 Developer (pro vývojáře k nasazení mimo produkční prostředí, bez omezení stejné jako Enterprise) [11].

Microsoft SQL Server 2019 je možné provozovat na platformě Windows, Linux nebo image kontejneru na platformě Docker a Kubernetes. V tomto ohledu tak lehce zaostává za konkurenčními systémy, ještě ve verzi 2016 byl navíc k dispozici jen na platformě Windows.

I v případě Microsoftu je pochopitelně pro placené verze samozřejmostí poskytování technické podpory. Stejně jako pro Oracle database je k dispozici dokumentace pouze v anglickém jazyce, tutoriály, videa a diskuzní fórum.

2.2.4 PostgreSQL

PostgreSQL také známý jako Postgres (původní název) je stejně jako MySQL open source databázový systém vyvíjený globální komunitou. Na rozdíl od MySQL ale obsahuje některé koncepty z objektově orientované databáze jako například dědičnost, proto bývá PostgreSQL označován jako objektově-relační databázový systém.

Primárně je PostgreSQL vyvíjen pro operační systémy Linux, ale stejně jako MySQL podporuje i další významné operační systémy jako je Windows, MacOS nebo Solaris. Kvalita dokumentace, dostupné nejen v angličtině ale i řadě dalších jazyků, jakož i tutorialů a dalších materiálů je na vysoké úrovni, jako tomu bývá u open source projektů se širokou komunitou uživatelů. Drobným nedostatkem v tomto ohledu je pouze chybějící oficiální fórum.

Navzdory rostoucí oblibě v posledních letech, co se týče popularity, PostgreSQL z dlouhodobého hlediska zaostává za MySQL. Z toho plyne nižší počet existujících nástrojů a aplikací třetích stran pro usnadnění správy databáze nebo i méně databázových administrátorů se zkušenostmi s PostgreSQL. Dále oproti výše zmíněným systémům neumožňuje PostgreSQL uchovávání datových struktur pouze v operační paměti. Výhodou je naopak důraz kladený na dodržování standardů SQL, díky čemuž podporuje i některé konstrukce, které v MySQL chybí.

2.2.5 Srovnání

Jednotlivé databázové servery se samozřejmě v mnohém liší. Jsou to například rozdíly ve způsobu implementace správy diskových prostor, v implementaci transakcí, síťové konfiguraci, jazycích pro programování v databázi, nebo třeba rozdílných klientech a dalších nástrojích pro práci s databází [12].

Všechny zmíněné systémy jsou však vhodné pro použití v produkčním prostředí a v naprosté většině případů bude požadavkům projektu vyhovovat každý z nich. Při volbě databázového serveru pro nově zakládané projekty bych se proto zaměřil na zkušenosti a znalosti vývojářů a administrátorů, kteří budou se systémem pracovat. Současně bych preferoval, pokud by to bylo možné, open source systémy – proč také platit za licenci softwaru, ke kterému existují open source alternativy.

2.3 Komunikační protokoly

Komunikační protokol je soubor pravidel, která určují syntaxi a význam jednotlivých zpráv, a tím definuje způsob komunikace a přenosu dat mezi dvěma koncovými body. Komunikační protokoly pracují v rámci jedné vrstvy síťové komunikace. Příkladem mohou být známé protokoly HTTP (aplikační vrstva), TCP (transportní vrstva), IP (síťová vrstva).

V této sekci jsou porovnány dva přístupy k možné implementaci webového rozhraní, REST a SOAP, které připadají v úvahu pro použití v praktické části práce. Důležité je zmínit, že REST přitom není komunikační protokol, ale architektonický vzor pro návrh síťové komunikace. Jedná se však o dvě konkurenční technologie, které jsou v případě návrhu a implementace webového rozhraní nejčastěji používány a vzájemně bývají často porovnávány.

2.3.1 REST

Autorem konceptu REST je Roy Fielding, který popsal jeho principy v rámci své disertační práce Architectural Styles and the Design of Network-based Software Architectures [13]. Representational State Transfer je datově orientovaná architektura rozhraní pro práci se zdroji v distribuovaném prostředí. Komunikace je bezestavová, což znamená, že každý požadavek obsahuje veškeré informace potřebné k jeho zpracování a žádná data klienta tedy nejsou uchovávána na straně serveru. Zdroje, kterými bývají nejčastěji data, mají vlastní URI identifikátor a pro přístup k nim využívá REST základní metody HTTP protokolu:

- GET – získání zdroje, bezpečná metoda.
- POST – vytvoření zdroje.
- PUT – úprava zdroje, idempotentní metoda.
- DELETE – odstranění zdroje, idempotentní metoda.

Při definici vlastností jednotlivých metod jsou zmíněny dvě vlastnosti:

- bezpečná metoda – je určena k získání dat a nemění stav serveru,
- idempotentní metoda – zaručuje vždy stejný výsledek bez ohledu na počet odeslaných požadavků, Každá bezpečná metoda je logicky rovněž idempotentní.

Účel, způsob použití a vlastnosti jednotlivých metod nejsou určeny webovým serverem nebo protokolem, jedná se pouze o předepsaný standard. Technicky je samozřejmě možné definovat rozhraní tak, že například pro vytvoření nebo modifikaci zdroje bude použita metoda GET. Výsledkem ignorování standardů a doporučení je ale riziko, že dojde k nechtěným změnám způsobených uživateli počítajícími se standardním chováním rozhraní a jednotlivých metod. Rozhraní, které dodržuje předepsané standardy je označováno jako REST API nebo RESTful API.

2.3.2 SOAP

Simple Object Access Protocol je protokol pro výměnu správ v XML formátu. Každý SOAP požadavek obsahuje kořenový element Envelope a v něm elementy:

- Header – nepovinný element, který může obsahovat například informace o klíči využívaném pro zabezpečení komunikace.
- Body – povinný element obsahující volanou metodu a její parametry.

Odpověď na takový požadavek pak má podobný formát s tím rozdílem, že v elementu body jsou přenášena data, která jsou výsledkem volané procedury.

Výhodou protokolu SOAP je jednoznačná a strojově čitelná definice formátu zpráv, zatímco v případě HTTP metod používaných REST rozhraním nebývá obsah zpráv formálně definován. Pro popis rozhraní se používá jazyk WSDL vycházející z jazyku XML. Slouží k definování jednotlivých funkcí, které rozhraní nabízí, jejích vstupních parametrů a také výstupů volání dané funkce.

2.3.3 Srovnání

Základním rozdílem je, že zatímco REST je orientován datově, protokol SOAP je orientován procedurálně. To naznačuje, že pro přístup ke zdrojům bývá vhodnější použití rozhraní postavené na REST principech, zatímco pro volání procedur je použitelnější protokol SOAP. Neznamená to však, že by CRUD operace nad daty nebylo možné implementovat za použití protokolu SOAP a naopak volání procedur s použitím principů REST. V obou případech se jedná o poměrně obecné technologie, které lze přizpůsobit potřebnému účelu a potřebné rozhraní implementovat oběma způsoby.

Dalším rozdílem je formát přenášených dat, kde REST nabízí více alternativ, zatímco protokol SOAP umožňuje přenos dat pouze ve formátu XML. S tím souvisí i velikost přenášených dat, která je, při volbě vhodného formátu (JSON), nižší než v případě protokolu SOAP a jejího formátu XML. Další výhodou je rychlost REST rozhraní, k níž přispívá i možnost cachování zdrojů, ke kterým bývá často přistupováno. Naproti tomu výhodou protokolu SOAP jsou rozšířené možnosti zabezpečení díky podpoře WS-Security. Jedná se o protokol, který poskytuje mechanismy zabezpečení webových služeb a zajišťuje integritu přenášených zpráv [15].

Implementace rozhraní podle principů REST je v poslední době obecně využívanější variantou. Z mého pohledu k tomu přispívá, kromě již zmíněných výhod, vyšší flexibilita nebo snazší a rychlejší implementace. Naproti tomu pro určité případy může být protokol SOAP vhodnější – například pokud vyžadujeme vyšší úroveň zabezpečení, nebo udržování stavu v průběhu série několika samostatných požadavků.

Kapitola 3

Analýza

V průběhu návrhu artchitektury systému reportů, zejména jeho cloudové části, jsem se současně pokusil zamyslet nad možnostmi využití cloud computingu v dalších částech celého parkovacího systému. Pro získání podkladů pro tyto úvahy jsem v první části této kapitoly identifikoval výhody i nevýhody výplívající ze současné architektury. Následně jsem prověřil možnosti pro případný přesun parkovacího systému nebo některých jeho částí na cloudové servery a důvody, které naopak přesunu do cloudu brání.

Druhá část kapitoly se pak věnuje indentifikaci a popisu funkčních a nefunkčních požadavků kladených na systém reportů, jehož implementace je cílem této práce.

3.1 Parkovací systém

Cílem analýzi parkovacího systému je prověření využitelnosti cloud computingu v návrhu architektury a implementaci parkovacího systému, proto jsem se zaměřil na vlastnosti systému plynoucí ze současné architektury a především její uzavřenosti v rámci lokální sítě.

3.1.1 Výhody a nevýhody současného systému

Uzavřenost parkovacího systému v rámci lokální sítě s sebou nese řadu výhod. V následujících bodech jsem se pokusil shrnout ty nejzásadnější z nich.

- Nepřetržitá dostupnost – parkovací systém není závislý na dostupnosti služeb poskytovatele internetu a zaručuje nepřetržitý provoz bez ohledu na výpadky internetových služeb. Případné výpadky mohou ovlivnit pouze některé vedlejší funkčnosti, například platbu pomocí SMS, nebo integrovaná zařízení třetích stran, například kreditní terminál platební stanice.

- Kontrola nad daty – databáze i všechna data se nachází na serverech, které jsou fyzicky umístěny u provozovatele parkovacího systému. To u něj může zvyšuje pocit, že jeho data jsou v bezpečí a má nad nimi plnou kontrolu

- Jednoduchost – architektura celého systému je poměrně jednoduchá. Jednotlivé komponenty parkovacího systému nejsou vzájemně provázány, komunikují pouze s Kernelem a nedochází k výměně dat s žádnou komponentou mimo lokální síť. Pro netechnicky vzdělané zákazníky je pak systém snadněji pochopitelný, transparentnější a důvěryhodnější. Jednoduchost architektury rovněž snižuje pravděpodobnost výskytu závad systému.

Kromě zmíněných výhod s sebou současná podoba architektury přináší i některé nevýhody.

- Nutnost vzdálených přístupů – přístup k datům a rozhraním parkovacího systému je možný jen v rámci lokální sítě. V případě že je z instalace hlášen problém, je potřeba, aby technická podpora zajistila vzdálený přístup k systému. Obvykle je pro řešení hlášeného problému nutné získat příslušné logovací soubory a často i zálohu dat z databáze, která může dosahovat velikosti několika stovek MB. V případě, že se jedná o instalaci s pomalejším připojením k internetu, může být získávání dat poměrně zdlouhavé. Pokud by tato data byla uložena v cloudu, celý proces by se výrazně urychlil a zjednodušil.

- Instalace softwaru – při každém zavádění parkovacího systému je nutné instalovat a konfigurovat software na lokálním zařízení. Celkový čas potřebný pro uvedení nové instalace do provozu to však neprodlužuje, jelikož příprava prostor a montáž hardwarových jednotek je časově náročnější.

- Opravy a upgrade softwaru – v případě nalezení chyby v softwaru nebo nutnosti jeho upgradu je pro její odstranění potřeba přeinstalovat danou část systému na všech instalacích.

- Neexistence jednotného rozhraní – jednotlivé instalace parkovacího systému mohou v případě potřeby poskytovat určitý typ rozhraní pro komunikaci se softwarem třetích stran. Jednotné rozhraní, které by umožňovalo přistupovat k libovolné instalaci však k dispozici není. To sice přímo nesouvisí s fyzickým umístěním komponent systému, ale v případě vytvoření určité centrální komponenty, která by byla sdílená všemi instalacemi systému, by byla implementace jednotného rozhraní snazší.

3.1.2 Verzování

Během doby, po kterou je parkovací systém na trhu dochází pochopitelně k jeho průběžnému vývoji a uvolňování nových verzí softwaru. V průběhu vývoje systému bylo stanoveno několik milníků, které přinesly nové majoritní verze, jež nezaručují kompatibilitu všech komponent s předchozími verzemi systému. Aktuálně je v různých zemích světa v provozu několik různých majoritních verzí systému.

Současně dochází samozřejmě také k vývoji a modernizaci hardwarových komponent, které není možné aktualizovat stejným způsobem jako softwarové částí systému. Otázku vzájemné kompatibility je přitom nutné řešit i ve vztahu softwaru a hardwaru. To je také jeden z důvodů, proč systém neumožňuje některé instalace upgradovat na nejnovější verze bez nákladné výměny stávajících hardwarových jednotek. Z toho plyne, že není možné poskytovat veškerý software parkovacího systému jako cloudovou službu, neboť by vznikl problém s vývojem nových verzí a kompatibilitou se stávajícími instalacemi.

3.1.3 Zakázkové úpravy

Přestože systém nabízí široké možnosti konfigurace a přizpůsobení se specifickým požadavkům zákazníka, čas od času jsou pro některé instalace, nebo skupiny instalací, realizovány menší nebo zásadnější zakázkové úpravy softwaru.

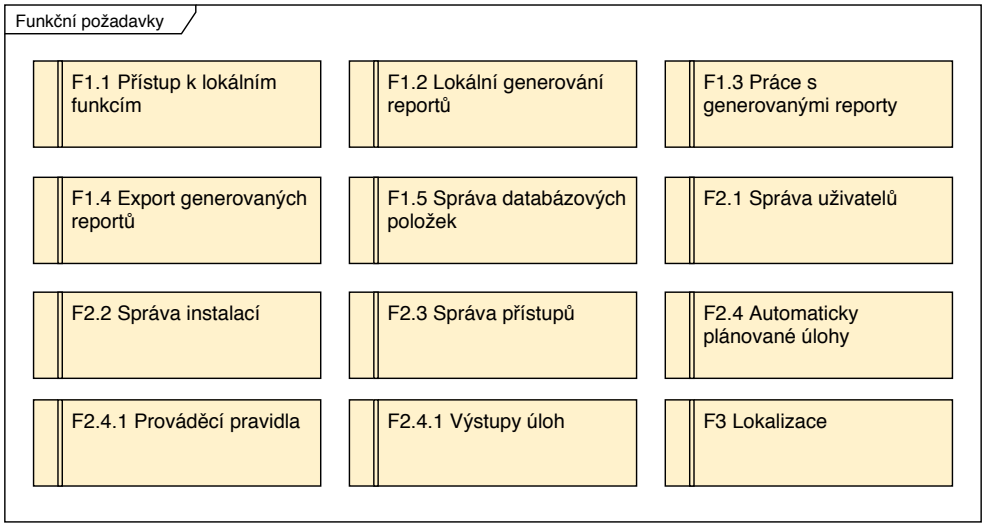
Zakázková úprava se může týkat jedné nebo několika softwarových částí systému, které jsou následně vydány ve verzích určených pro nasazení na konkrétních instalacích. Tyto verze přitom nemusí být kompatibilní s verzemi vydávanými v rámci hlavní vývojové větve a při záměru využití cloud computingu nastává podobný problém, jako v případě nekompatibility odlišných majoritních verzí systému.

3.1.4 Shrnutí

Vzhledem k tomu, že parkovací systém má svá specifika a kromě softwarové části sestává rovněž z hardworových jednotek, kompletní přesun systému do cloudu jednoduše není možný. Současně se zdá být vhodným řešením ponechat umístění hlavní části logiky systému na jednotlivých instalacích, což umožní zachování současného systému verzování a aktualizací systému. Zároveň by však přesunutí některých částí systému přineslo nezpochybnitelné výhody.

3.2 Reportovací systém

V části týkající se reportovacího systému jsou identifikovány hlavní požadavky kladené na systém.



Obrázek 3.1: Funkční požadavky

3.2.1 Funkční požadavky

Přehledný souhrn funkčních požadavků kladených na systém zobrazuje diagram požadavků (obr. 3.1). Lokální funkce (F1.x) musí být k dispozici i bez připojení k veřejné síti a zbylých částí systému umístěných mimo lokální síť instalace.

F1.1 Přístup k lokálním funkcím

Pro přístup k lokálním funkcím systému bude vyžadováno ověření pomocí uživatelského účtu a hesla. Ověření se bude provádět oproti přístupovým udajům do databáze.

F1.2 Lokální generování reportů

Systém bude umožňovat vytvoření reportů z libovolného zařízeních v rámci lokální sítě instalace parkovacího systému. Pro generování zvolí uživatel typ reportu z dostupných šablon a zadá její vstupní parametry. Po vygenerování systém zobrazí výsledná data v přehledné tabulce. Lokální vytváření reportů musí být funkční i bez připojení k veřejné síti a zbylých částí systému umístěných mimo lokální síť instalace.

F1.3 Práce s generovanými reporty

Po vygenerování reportu budou data přehledně zobrazena uživateli formou tabulky. Systém bude umožňovat vygenerovaná data filtrovat a vyhledávat v nich.

F1.4 Export generovaných reportů

Vygenerované reporty bude možné exportovat v několika formátech a uložit na disk. Systém bude podporovat minimálně formáty PDF, CSV, XML a XLS.

F1.5 Správa databázových položek

Lokální generování reportů vyžaduje umístění šablon reportů, adresářových struktur a překladových položek v lokální databázi. Systém bude umožňovat tyto položky v databázi vytvářet, prohlížet, editovat a odstraňovat.

F2.1 Správa uživatelů

Pro přístup do části systému společnou pro jednotlivé instalace parkovacího systému bude vyžadováno přihlášení k uživatelskému účtu, jež bude možné v sytému spravovat uživateli s administrátorským oprávněním. Systém bude umožňovat přidání nového účtu, úpravu a odstranění existujícího účtu. Běžným uživatelům bez administrátorského oprávnění bude systém umožňovat úpravu vlastního účtu včetně změny hesla.

F2.2 Správa instalací

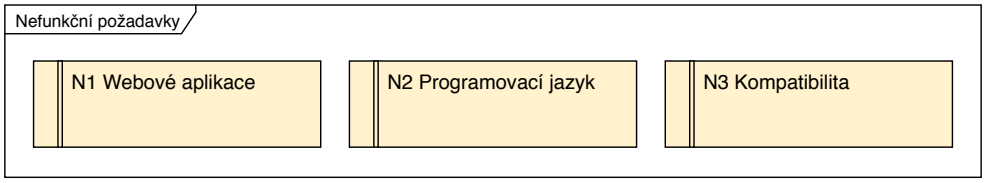
Systém bude obsahovat možnost správy jednotlivých instalací parkovacího systému. Systém bude umožňovat vytvoření nové instalace a nastavení údajů potřebných pro připojení k části systému umístěnému na serveru v lokální síti instalace. Kromě toho bude systém umožňovat i prohlížení, editaci a odstranění existujících instalací.

F2.3 Správa přístupů

Uživatelům bude možné administrátory přidělit přístup k jednotlivým instalacím obsahující přístupové údaje potřebné k autorizaci a úspěšnému připojení k části systému umístěnému na serveru v lokální síti instalace.

F2.4 Automaticky plánované úlohy

Uživatelé budou mít možnost vytvořit předpokonfigurované úlohy, jejichž úkolem bude spustit generování reportu na některé z instalací parkovacího systému, ke které má daný uživatel přiřazený přístup. Uživatel bude moci vybrat typ reportu z šablon dostupných na dané instalaci a nastavit její vstupní parametry. Úlohám bude možné přiřazovat formy výstupu a prováděcí pravidla.



Obrázek 3.2: Nefunkční požadavky

F2.4.1 Prováděcí pravidla

Každé úloze bude možné přiřadit libovolné množství pravidel aumotacikého provedení lišících se časem a frekvencí spuštění. Prováděcí pravidla bude možné vytvořit, upravit i odstranit.

F2.4.2 Výstupy úloh

Dále bude systém umožňovat odeslání výstupů emailem libovolnému počtu příjemců, včetně nastavení obsahu emailu a jeho příloh - reportu exportovaného do zvolených formátů. Vytvořené výstupy úloh bude rovněž možné upravit i odstranit.

F3 Lokalizace

Systém bude vícejazyčný s možností přidávat nové (a upravovat již existující) jazykové mutace bez nutnosti zásahu do zdrojových kóodů.

3.2.2 Nefunkční požadavky

Přehledný souhrn nefunkčních požadavků kladených na systém zobrazuje diagram požadavků (obr. 3.2).

N1 Webové aplikace

Systém bude přístupný prostřednictvím webového prohlížeče. Zaručena bude podporova minimálně nejpoužívanějších prohlížečů, kterými jsou Google Chrome (verze 42.0 a vyšší), Safari (verze 11 a vyšší), Internet Expolorer (verze 10 a vyšší)/Microsoft Edge a Mozilla Firefox (verze 50.0 a vyšší) [16].

N2 Programovací jazyk

Hlavním programovacím jazykem pro implementaci jednotlivých částí systému bude jazyk Java.

N3 Kompatibilita

Systém bude kompatibilní s parkovacím systémem ve stávající podobě a zejména pak s aktuální podobou databáze umístěné na lokálním serveru jednotlivých instalací.

Kapitola 4

Návrh

Kapitola se zabývá návrhem architektury parkovacího systému a návrhem systému reportů na základě provedené analýzi popsané v předchozí kapitole.

V první části jsou nastíněny základní myšlenky návrhu nové architektury parkovacího systému, ve které je část systému přesunuta mimo lokální síť jednotlivých instalací na cloudové servery. K návrhu nové architektury celého parkovacího systému jsem se rozhodl z důvodu, že návrh reportovacího systému se v průběhu vypracování práce ukázal být jednodušší než v momentě tvorby jejího zadání. Reportovací systém bude rovněž z části umístěn na cloudových serverech a považuji za užitečné zamyslet se nad možnostmi využití cloud computingu i v dalších částech systému.

Obsahem druhé části kapitoly je návrh architektury reportovacího systému, včetně modelu nasazení, a návrh komunikace jednotlivých jeho částí. Dále kapitola obsahuje popis uživatelských rolí, případů užití s jednotlivými scénáři a návrh databázových struktur.

4.1 Parkovací systém

Záměrem návrhu nové architektury parkovacího systému je eliminace některých nevýhod vyplývajících ze stávající architektury systému a současně v maximální možné míře zachování jeho aktuálních výhod. Výhody a nevýhody byly popsány v analýze současné podoby systému a v souladu s jejími závěry je při návrhu nové architektury hlavním cílem:

- zachování dostupnosti a spolehlivosti,
- zachování výkonu a rychlosti odezvy na požadavky uživatelů,
- usnadnění přístupu k datům a rozhraní.

4.1.1 Architektura systému

Jak již bylo zmíněno, transformace systému s využitím cloud computingu má v tomto případě svá omezení. Předně by nebylo vhodné, aby jednotlivé hardwarové jednotky komunikovaly přímo s komponentou systému umístěnou mimo lokální síť, což by mělo za následek nefunkčnost celého systému v případě výpadku internetových služeb. Z toho plyne nutnost zachování centrální komponenty v rámci lokální sítě. Ze stejného důvodu musí mít tako komponenta k dispozici minimálně část dat, která je nezbytná pro zajištění nepřetržitého provozu parkovacího systému z pohledu jeho koncových uživatelů. Současně by ale byla data lépe přístupná, pokud by byla umístěna na centrálním serveru mimo lokální síť. Řešením je rozdělení datového úložiště na dvě části:

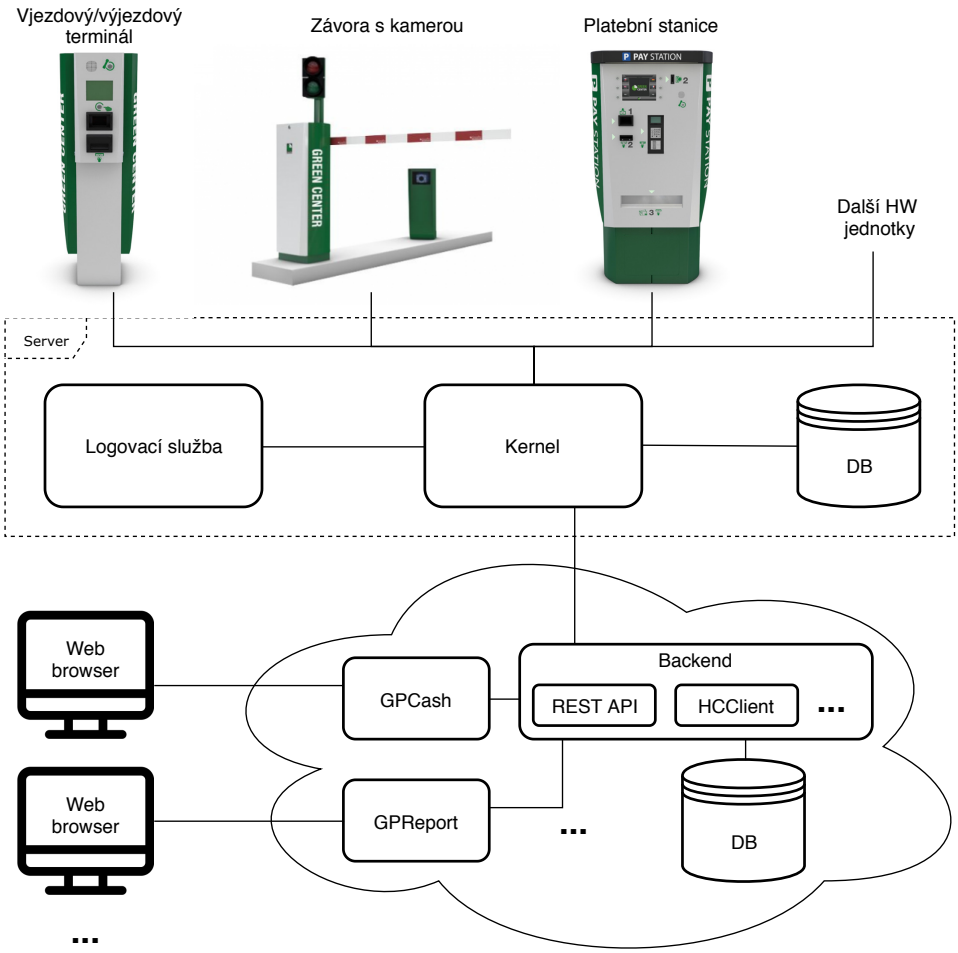
- lokální databáze – účelem je uložení dat nezbytných pro poskytnutí základních funkčností parkovacího systému,
- centrální databáze – bude obsahovat kompletní data, která jsou využívána nejen pro provoz systému.

Protože tím dojde k výraznému omezení ukládání dat do lokální databáze, bylo by možné udržovat tato data ve vnitřní paměti a tím dosáhnout zrychlení databázových operací. Vzhledem k tomu, že tyto operace jsou ale obvykle výsledkem požadavku některé z hardwarových jednotek, je s ohledem na nutnost přenosu požadavku a odpovědi mezi jednotkou a Kernelem časový rozdíl mezi prací s daty ve vnitřní paměti a na disku zanedbatelný.

Centrální databáze bude umístěna na cloudových serverech. K databázi bude přistupovat backend serverové části systému, jehož součástí bude definice rozhraní a jeho konkrétní implementace. Tato rozhraní budou sloužit jak pro komunikaci s dalšími softwarovými produkty společnosti GREEN Center, tak k možné integraci a komunikaci parkovacího systému s aplikacemi a systémy třetích stran.

Backend systému pak bude komunikovat s Kernelem na jednotlivých instalacích. Pro interní komunikaci je ve společnosti GREEN Center implementován vlastní protokol HCClient založený na výměně zpráv na transportní vrstvě pomocí protokolu TCP. Tento protokol by mohl být zachován i pro komunikaci Kernelu s backendem systému na cloudových serverech. Problémem je však jeho aktuální implementace pomocí standardních TCP socketů, které nejsou zabezpečeny. Pro použití v rámci veřejné sítě by bylo nutné implementaci lehce upravit a využít například třídy SSLSocket, která rozšiřuje standardní socket a zajišťuje jeho zabezpečení. Použitím protokolů jako je SSL zaručuje integritu zpráv, autentizaci a důvěrnost [17].

Kernel by v tomto návrhu zpracovával jen část příchozích požadavků od hardwarových jednotek. Po přijetí zpávy by na základě jejího typu rozhodl, zda je určena ke zpracování v lokální části systému. V takovém případě by tak



Obrázek 4.1: Návrh architektury parkovacího systému

učinil, v opačném případě by příchod zprávy pouze zalogoval a bez nutnosti jakékoliv modifikace by ji přeposlal ke zpracování cloudové části systému.

Desktopové aplikace, které v současné podobě systému komunikují s Kernelem nebo jsou připojeny přímo k databázi, budou nahrazeny webovými verzemi a poskytovány jako služba. K datům budou přistupovat prostřednictvím backendu na cloudovém serveru a některého z jeho rozhraní. Návrh architektury je zachycen na obrázku 4.1.

4.1.2 Datové modely

Obsahem lokální databáze by podle nového návrhu měla být pouze data nezbytně nutná pro základní funkčnosti systému. Proto je nutné identifikovat o které tabulky databáze se jedná. Momentálně obsahuje databáze 89 tabu-

lek a není možné je všechny uvádět. Pro zjednodušení je ale lze rozdělit do několika skupin podle společného účelu nebo některých společných vlastností.

- Konfigurace – jedná se o tabulky, které jsou read only a slouží k udržení informací o konfiguraci systému. Příkladem jsou například konkrétní hardwarové jednotky připojené v systému a jejich umístění v síti, typy používaných parkovacích a slevových karet, nastavení pro výpočet parkovného, měny, nastavení zón a podobně.
- Parkovací karty – záznamy parkovacích karet a dalších souvisejících entit jako například vlastníků karet nebo poznávacích značek.
- Slevové karty a vouchery – slevové karty a vouchery se používají při platbě k odečtění části parkovného.
- Obsazenost – za předpokladu, že daná instalace obsahuje kontrolu obsazenosti jednotlivých parkovacích míst, udržují tabulky tato data. Ta jsou následně využívána v naváděcích nebo informačních jednotkách nebo softwarem určeným k vizualizaci obsazenosti parkoviště.
- Platby – při každé platbě je v databázi vytvořen záznam účtenky, jakým způsobem byla platba realizována a další informace.
- Žurnály – tyto tabulky se mohou týkat například parkovacích karet nebo hardwarových jednotek a ukládají jejich záznamy o aktivitách. U parkovací karty to může být vydání, zobrazení informací o kartě, platba blokace a podobně.
- Statistiky hotovosti – do databáze jsou průběžně ukládány stavy hotovosti jednotlivých platebních stanic, informace o výběrech a doplnění hotovosti, konkrétní přijaté a vyplacené nominály v průběhu provedené platby a podobně.
- Reporty – tabulky týkající se reportů (viz sekce 4.3.4), využívány jsou pouze softwarem pro tvorbu reportů.

Jednotlivé skupiny tabulek se navzájem liší v tom, které části systému k nim přistupují a jakým způsobem (R – čtení, W – zápis, nebo RW – čtení i zápis). Souhrn těchto údajů zobrazuje tabulka 4.1.

Tabulky, které jsou čteny hardwarovými jednotkami, jsou pro běh systému nezbytné a musí být přítomny v lokální databázi, zbytek tabulek je pak možné udržovat pouze v centrální databázi.

4.1.3 Synchronizace databází

Rozdělení dat mezi lokální a centrální databázi s sebou přináší nutnost tyto databáze nějakým způsobem synchronizovat. To se týká tabulek,

Tabulka 4.1: Skupiny tabulek databáze parkovacího systému

Typ	Přístup HW jednotek	Přístup softwaru
Konfigurace	R	R
Parkovací karty	RW	RW
Slevové karty a vouchery	RW	RW
Obsazenost	RW	R
Platby	W	RW
Žurnály	W	RW
Statistiky hotovosti	W	R
Reporty	–	RW

které jsou součástí lokální databáze, ale současně mohou být modifikovány prostřednictvím rozhraní na cloudových serverech.

Typickým příkladem je situace, kdy uživatel zaplatí parkovné jinak než u automatické platební stanice. Služba umožňující platby například pomocí SMS bude přistupovat přímo k backendu na cloudových serverech a informace o proběhlé platbě je tak zanesena do centrální databáze. Současně je ale nutné uložit tuto informaci i do databáze lokální, aby systém umožnil uživateli parkoviště opustit. Podobné situace mohou nastat i v opačném směru.

V zásadě existují dva způsoby, jak synchronizaci databází realizovat.

- Na úrovni databází – databázové servery umožňují v distribuovaném prostředí jednotlivé databáze replikovat. Obvykle se používá replikace typu master-slave, kde jsou změny nadřízené databáze přenášeny do databáze podřízené. Existují však i řešení pro obousměrnou replikaci nazývanou master-master nebo multi-master. Tento typ je pochopitelně implementačně náročnější, zejména při předcházení a řešení konfliktu transakcí. To ale nemusí vadit vzhledem k tomu, že implementaci replikace řeší samotný databázový server.

- Na úrovni aplikací – v tomto případě by jednotlivé databáze navzájem nevěděly o své existenci. O zajištění uložení dat do obou databází by se staral Kernel a backend na cloudovém serveru jednoduchým přeposíláním příchozích zpráv. Problémem je však již zmíněné řešení konfliktních transakcí, jehož implementace na aplikační úrovni by byla velmi složitá. Z toho důvodu by měla být synchronizace řešena na úrovni databází.

4.1.4 Shrnutí

Výsledný návrh architektury parkovacího systému využívá cloudové servery pro umístění centrální databáze, což společně s jednotným rozrhaním

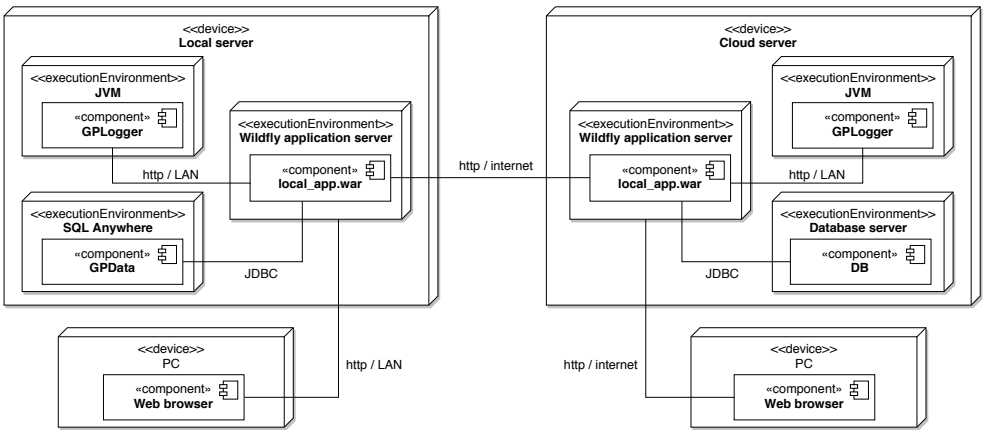
usnadňuje přístup k datům. Jedná se o velmi obecný návrh a nastínění základních myšlenek. Propracovanější návrh by vyžadoval, kromě jiného, vyřešení otázek týkajících se například bezpečnosti nebo zajištění dostatečného výkonu a dostupnosti systému. V tuto chvíli se o reálném využití návrhu, který by znamenal tak zásadní zásah do architektury systému, neuvažuje, proto bylo záměrem spíše ukázat směr, kterým by bylo možné se v budoucnu vydat a návrh rozvíjet.

4.2 Reportovací systém

Systém se skládá ze dvou samostatných webových aplikací. První z nich, dále v textu nazývaná lokální, bude nasazena na každé jednotlivé instalace parkovacího systému a použitelná v rámci lokální sítě. Z důvodu zpětné kompatibility a možnosti nasazení na instalace, které jsou již v provozu, bude komunikovat s databází v takové podobě, jakou parkovací systém využívá v současné době. Kromě databáze bude dále pomocí protokolu HTTP komunikovat s logovací službou (obvykle nasazenou na stejném zařízení). Pro potřeby druhé aplikace pak bude poskytovat REST API.

Druhou část systému tvoří aplikace, dále nazývána jako cloudová. Bude nasazena na serverech spravovaných společností GREEN Center nebo na pronajatých cloudových serverech. Cloudová aplikace bude přistupovat k vlastní nově navržené databázi a logovací službě. Pro přístup k datům na jednotlivých instalacích parkovacího systému bude využívat již zmíněné REST API lokální aplikace.

Rozložení jednotlivých komponent a způsoby jejich vzájemné komunikace popisuje model nasazení (viz obrázek 4.2).



Obrázek 4.2: Model nasazení

Pro případ změny architektury systému tak, jak je navrhována v předchozích částech této práce, mohou být do cloudové aplikace později

začleněny i funkčnosti lokální aplikace. Toho lze dosáhnout velmi snadno díky modulární architektuře podporované frameworkem Spring, zejména pak konceptu bean – veškerá logika aplikací bude implementována jako Spring service a danou část kódu pak lze využít v libovolné Spring aplikaci. Cloudová aplikace by tedy plnila úlohu obou aplikací současně. K datům z centrální databáze v nově navržené architektuře by pak přistupovala pomocí REST rozhraní podobným způsobem, jako v současné podobě systému přistupuje k datům z jednotlivých instalací prostřednictvím lokální aplikace a jejího rozhraní.

4.2.1 Uložení dat

Persistenci dat zajišťuje relační databáze. Obě aplikace budou konfigurovatelné pro připojení k databázi s libovolným umístěním, přestože se předpokládá, že, minimálně v případě lokální aplikace, bude ve většině případů databázový server umístěn na stejném zařízení jako aplikační server. Návrh organizace dat v databázi pro lokální a cloudovou aplikaci je uveden v sekci 4.3.4, respektive 4.4.3.

4.2.2 Logovací služba

Obě aplikace využívají již implementovanou logovací službu GPLogger. Komunikace s ní probíhá pomocí vlastního HTTP protokolu, jehož rozhraní importované do webových aplikací obsahuje základní metody:

- setupConnection(String applicationName, String logServerIP, int logServerPort) – slouží k navázání spojení s logovací službou podle zadaných parametrů IP adresy a portu. Logovací služba následně přijaté zprávy zapisuje do složky a logovacích souborů s názvem podle parametru applicationName.
- logInfo(String text) – zápis informační hlášky.
- logDebug(String text) – zápis debug hlášky.
- logError(String text) – zápis chybové hlášky.

Kromě uvedených základních metod obsahuje interface další méně používané metody, například přetížené metody zápisu události přijímající jako parametr objekt Java výjimky.

4.3 Lokální aplikace

Sekce se týká návrhu lokální aplikace. Obsahuje popis uživatelských rolí, případů užití a dále návrh webového rozhraní a uložení dat v databázi.

4.3.1 Uživatelské role

Aplikace rozlišuje dvě uživatelské role: administrátor a běžný uživatel. Nepřihlášený uživatel nemá přístup do žádné z částí aplikace a je přesměrován na úvodní stránku s možností přihlášení.

Administrátor

Administrátor má přístup do sekcí umožňujících zásahy do databáze. Má možnost zobrazit, přidat, upravit nebo smazat šablony reportů, jejich organizaci v adresářové struktuře, a překladové položky. Kromě toho má administrátor stejné pravomoce jako běžný uživatel. Pro zamezení vzniku škod v databázi by administrátorské účty měly být využívány pouze pracovníky IT Supportu společnosti GREEN Center.

Uživatel

Běžnému uživateli (obsluha parkoviště a ostatní zaměstnanci provozovatele) je umožněno prohlížet šablony, ke kterým má přístup (některé jsou k dispozici jen administátorovi). Z dostupných šablon pak může po zadání jejích parametrů generovat reporty. Aplikace umožňuje vygenerovaný report prohlížet a data dále filtrovat nebo v nich vyhledávat přímo ve webovém prohlížeči, nebo jej exportovat a uložit v několika různých formátech.

4.3.2 Případy užití

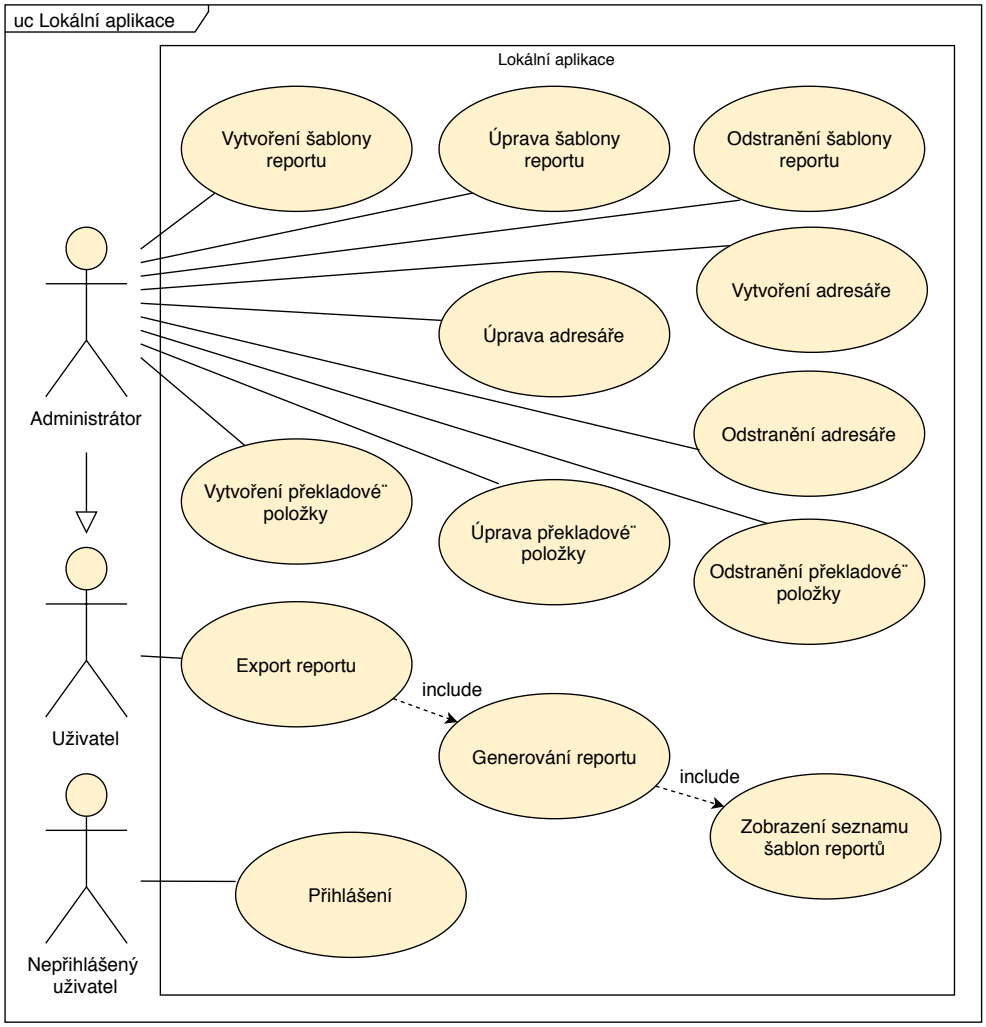
Model případu užití je zobrazen na obrázku 4.3. Ve slovním popisu jsou pro zjednodušení sloučeny téměř identické případy užití, jejichž výsledkem jsou databázové operace s šablonami, adresáři a překladovými položkami.

Přihlášení

Nepřihlášenému uživateli aplikace automaticky zobrazí přihlašovací formulář, do kterého uživatel zadá přihlašovací údaje. Aplikace přihlašovací údaje ověří a v případě jejich správnosti umožní uživateli přístup do aplikace a přesměruje ho na stránku se seznamem šablon reportů. Neproběhne-li přihlášení úspěšně, aplikace zamítne uživateli přístup a zobrazí znovu přihlašovací formulář společně s chybovou hláškou o neúspěšném přihlášení.

Odhlášení

Uživatel klikne na tlačítko „Odhlásit“. Aplikace uživatele odhlásí a přesměruje na úvodní stránku.



Obrázek 4.3: Use case diagram – lokální aplikace

Vytvoření databázové položky

Administrátor vybere v menu položku „Šablony“ případně „Překlady“ a následně klikne na tlačítko „Vytvořit šablonu“, „Vytvořit adresář“, případně „Vytvořit překladovou položku“. Aplikace zobrazí formulář pro přidání odpovídající databázové položky, administrátor vyplní potřebné údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu uloží položku do databáze. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Úprava databázové položky

Administrátor vybere v menu položku „Šablony“ případně „Překlady“ a následně klikne na tlačítko „Upravit“ u příslušné databázové položky. Aplikace zobrazí formulář pro úpravu odpovídající databázové položky s předvyplněnými daty, administrátor upraví potřebné údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu upraví položku v databázi. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Odstranění databázové položky

Administrátor vybere v menu položku „Šablony“ případně „Překlady“ a následně klikne na tlačítko „Odstranit“ u příslušné databázové položky. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní položku z databáze.

Zobrazení seznamu šablon reportů

Uživatel vybere v menu položku „Šablony“. Aplikace zobrazí seznam dostupných šablon reportů organizovaný v adresářové struktuře.

Generování reportu

Uživatel zobrazí seznam šablon reportů a následně klikne na tlačítko „Generovat“ u příslušné šablony. Aplikace zobrazí formulář pro zadání proměnných šablony, uživatel vyplní požadované údaje a formulář odešle. Aplikace provede validaci dat a v případě úspěchu vygeneruje ze šablony a zadaných proměnných požadovaný report a výsledná data zobrazí uživateli v přehledné tabulce s možností vyhledávání a filtrování. V případě neúspěšné validace vyzve aplikace uživatele chybovou hláškou k zadání validních hodnot proměnných.

Export vygenerovaného reportu

Uživatel vygeneruje požadovaný report a následně klikne na tlačítko „Export“ v požadovaném formátu. Aplikace exportuje data do souboru v příslušném formátu a soubor nabídne uživateli ke stažení.

4.3.3 REST API

Pro lokální a cloudovou aplikaci je třeba navrhnout vhodný způsob komunikace. Tato komunikace je jednostranná a jejím účelem je pouze odeslání vyžádaných dat z lokální aplikace do cloudové. Pro tento účel je navrženo REST API pouze na straně lokální aplikace, s jehož implementací výrazně pomáhá použití frameworku Spring (viz sekce 5.3.3 a 5.4.6).

Rozhraní se využívá při vytváření a spouštění automatických úloh v cloudové aplikaci. Při vytváření je nutné získat seznam dostupných šablon z příslušné instalace, jejich popis a následně i datové typy a význam proměnných, jejichž hodnoty je pro generování reportu z dané šablony nutné zadat. Při spouštění úlohy je pak potřeba odeslat informaci o tom, jaká šablona bude pro generování použita, hodnoty jejích proměnných a získat zpět vygenerovaná data.

Rozhraní pro popisované použití není nijak žvlášt složité a vystačí si s následujícími třemi metodami (v seznamu je uvedeno URL, HTTP metoda a stručný popis s uvedením parametrů a návratové hodnoty):

- /api/templates; GET – metoda bez parametrů vrací názvy a popis dostupných šablon organizované v adresářové struktuře.
- /api/template/{id}; GET – vrací datový objekt šablony podle zadaného ID v URL požadavku.
- /api/template/{id}/report; GET – návratovou hodnotou jsou vygenerovaná data reportu získaná z šablony podle zadaného ID v URL, a hodnot proměnných v těle požadavku.

4.3.4 Uložení dat

Sekce popisuje organizace dat v databázi a uvádí význam jednotlivých entit v aplikaci. Uvedené tabulky jsou součástí rozsáhlé databáze parkovacího systému v původní nezměněné podobě, v jaké byly navrženy pro desktopové aplikace. Důvodem je nutnost zpětné kompatibility webové aplikace s aktuálně používanou strukturou databáze.

Language

Obsahuje seznam jazyků, v jakých může být parkovací systém lokalizován.

- IDLanguage – primární klíč, integer, not null – jedinečné id jazyka,
- Name – varchar(128), not null – název jazyka.

RTProperty

V tabulce jsou organizovány překladové klíče a jejich hodnoty pro jednotlivé jazyky. Tabulka neobsahuje položku vlastního id jako primárního klíče, záznamy jsou namísto toho jednoznačně určeny dvojicí Key:Language, ke kterým je přiřazena příslušná hodnota. Překladové klíče jsou využívány například u názvů šablon a adresářových struktur a před vykreslením uživatelského rozhraní jsou nahrazeny jejich hodnotou podle aktuálně nastaveného jazyka. Tabulka RTProperty obsahuje datové položky:

- Key – primární klíč, varchar(1024), not null – název překladové položky,
- Language primární klíč, cizí klíč, integer, not null – jedinečné id jazyka,
- Value – text, not null – hodnota překladové položky pro daný klíč a jazyk.

RTDirectoryStructure

Obsahuje adresáře do kterých je možné organizovat šablony reportů podle jejich typu. Adresáře je možné vzájemně vnořovat. Tabulka RTDirectoryStructure obsahuje datové položky:

- IDDirStr – primární klíč, integer, not null – jedinečné id adresáře,
- RootID – cizí klíč, integer – jedinečné id rodičovského adresáře (může nabývat hodnoty NULL v případě umístění v kořenovém adresáři),
- DirectoryKey – varchar(1024), not null – překladový klíč názvu adresáře,
- OrderIndex – integer, not null – porovnáním s ostatními adresáři v rámci svého umístění definuje pořadí v jakém jsou jednotlivé adresáře seřazeny.

RTTemplate

Data v tabulce reprezentují šablony reportů, které je možné z dat parkovacího systému vytvářet. Tabulka RTTemplate obsahuje datové položky:

- IDTemplate – primární klíč, integer, not null – jedinečné id šablony,
- IDDirStr – cizí klíč, integer – jedinečné id rodičovského adresáře (může nabývat hodnoty NULL v případě umístění v kořenovém adresáři),
- NameKey – varchar(1024), not null – překladový klíč názvu šablony,
- DescriptionKey – varchar(1024), not null – překladový klíč popisu šablony,
- Level – integer, not null – určuje minimální level oprávnění uživatele, který může s danou šablonou pracovat,
- Valid – bit, not null – definuje platnost šablony (neplatné šablony nemohou běžní uživatelé používat),
- XMLBody – text, not null – vlastní tělo šablony definující výslednou podobu reportu ve formátu XML obsahující definice jednotlivých sloupců dat, vstupní proměnné šablony a jejich typy, SQL příkazy a podobně,
- OrderIndex – integer, not null – definuje pořadí v jakém jsou v rámci svého umístění jednotlivé šablony seřazeny.

4.4 Cloudová aplikace

Stejně jako v případě lokální aplikace, i pro cloudovou aplikaci jsou popsány uživatelské role, případy užití a návrh databázových struktur.

4.4.1 Uživatelské role

Opět se rozlišují dvě uživatelské role: administrátor a běžný uživatel. Nepřihlášený uživatel nemá přístup do žádné z částí aplikace a je přesměrován na úvodní stránku s možností přihlášení.

Administrátor

Administrátor má přístup do sekce správy uživatelů a instalací, a kromě toho i do dalších částí aplikace stejně jako běžný uživatel. Pro toto rozdělení oprávnění je důležité, aby role administrátora byla přidělena pouze lidem z oddělení IT Supportu společnosti GREEN Center, kteří budou systém spravovat a po získání nového zákazníka a zřízení nové instalace vytvoří jeho provozovateli přístupy do systému.

Uživatel

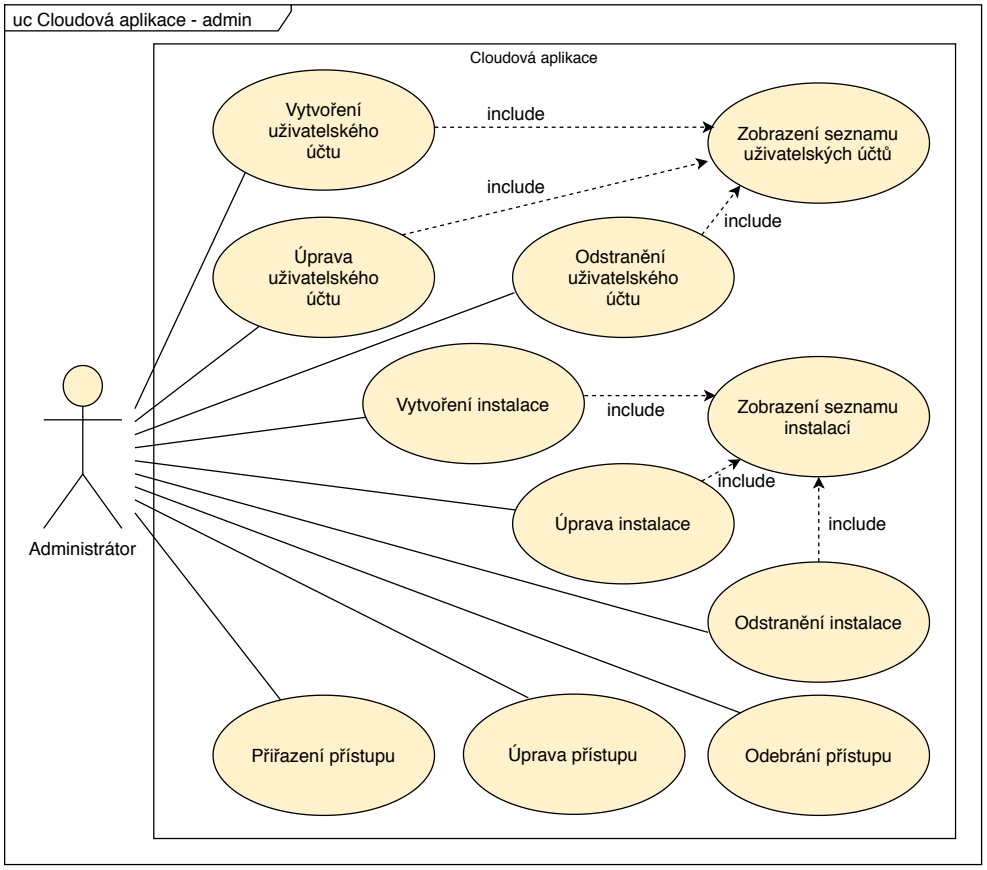
Běžný uživatel má přístup k detailům svého účtu a některé z nich může i upravovat. Hlavní funkčností aplikace je plánování spouštění automatických úloh. Uživatel může pro libovolnou instalaci, ke které má přístup, naplánovat úlohu, spouštějící se dle svého typu v pravidelných časových intervalech definovaných uživatelem. Automaticky spouštěným úlohám může uživatel nastavit, co bude jejich výsledkem a také směřování jejich výstupů, které je možné odesílat podle zadaných šablon v podobě emailové zprávy.

4.4.2 Případy užití

Případy užití vychází ze seznamu funkčních požadavků kladených na aplikaci a slouží k popisu funkcionalit aplikace z pohledu uživatele. Z důvodu velikosti byl model rozdělen (viz obrázky 4.4 a 4.5). Pro úplnost modelu zbývá doplnit, že administrátor dědí všechna oprávnění běžného uživatele.

Přihlášení

Nepřihlášenému uživateli aplikace automaticky zobrazí přihlašovací formulář, do kterého uživatel zadá přihlašovací údaje (email a heslo). Aplikace ověří přihlašovací údaje a v případě jejich správnosti umožní uživateli přístup do aplikace a přesměruje ho na stránku s detaily jeho uživatelského účtu. Neproběhne-li přihlášení úspěšně, aplikace zamítne uživateli přístup a zobrazí znovu přihlašovací formulář společně s chybovou hláškou o neúspěšném přihlášení.



Obrázek 4.4: Use case diagram – cloudová aplikace, administrátor (pozn.: administrátor dědí oprávnění běžného uživatele, viz obrázek 4.5)

Odhlášení

Uživatel klikne na tlačítko „Odhlásit“. Aplikace uživatele odhlásí a přesměruje na úvodní stránku.

Zobrazení seznamu uživatelských účtů

Administrátor vybere v menu položku „Uživatelé“. Aplikace zobrazí seznam uživatelů seřazený podle uživatelského jména.

Vytvoření uživatelského účtu

Administrátor zobrazí seznam uživatelských účtů a následně klikne na tlačítko „Přidat uživatele“. Aplikace zobrazí formulář pro přidání uživatelského účtu. Administrátor vyplní potřebné údaje a odešle formulář. Aplikace provede va-

lidaci dat a v případě úspěchu uloží uživatelský účet do databáze. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Úprava uživatelského účtů

Administrátor zobrazí seznam uživatelských účtů a následně klikne na tlačítko „Upravit“ u příslušného uživatelského účtu. Aplikace zobrazí předvyplněný formulář pro úpravu uživatelského účtu. Administrátor upraví požadované údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu uživatelský účet upraví. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Odstranění uživatelského účtu

Administrátor zobrazí seznam uživatelských účtů a následně klikne na tlačítko „Odstranit“ u příslušného uživatelského účtu. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní uživatelský účet z databáze.

Úprava vlastního uživatelského účtů

Uživatel vybere v menu položku „Můj profil“. Aplikace zobrazí předvyplněný formulář pro úpravu uživatelského účtu. Uživatel upraví požadované údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu uživatelský účet upraví. V opačném případě aplikace vyzve uživatele k zadání validních dat.

Změna hesla

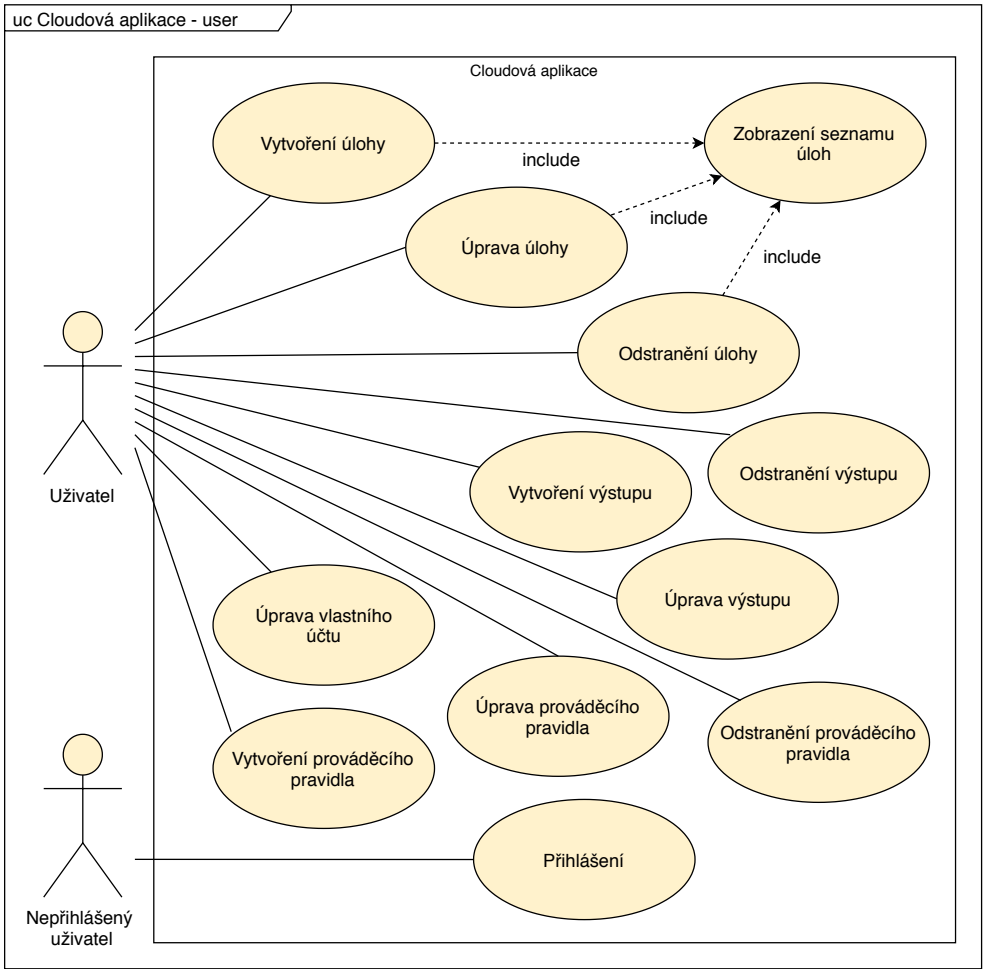
Změnu hesla provede uživatel jedním formulářem v rámci úpravy uživatelského účtu. V případě nevyplnění daných polí zůstane heslo nezměněno.

Zobrazení seznamu instalací

Administrátor vybere v menu položku „Instalace“. Aplikace zobrazí seznam instalací seřazený podle jejich názvu v abecedním pořadí.

Vytvoření instalace

Administrátor zobrazí seznam instalací a následně klikne na tlačítko „Vytvořit instalaci“. Aplikace zobrazí formulář pro vytvoření instalace. Uživatel vyplní potřebné údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu uloží záznam do databáze. V opačném případě aplikace vyzve uživatele k zadání validních dat.



Obrázek 4.5: Use case diagram – cloudová aplikace, běžný uživatel

Úprava instalace

Administrátor zobrazí seznam instalací a následně klikne na tlačítko „Upravit“ u příslušné instalace. Aplikace zobrazí formulář pro úpravu s předvyplněnými daty. Uživatel upraví požadované údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu záznam upraví. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Odstranění instalace

Administrátor zobrazí seznam instalací a následně klikne na tlačítko „Odstranit“ u příslušné instalace. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní záznam z databáze.

Přiřazení přístupu

Administrátor zobrazí formulář pro úpravu příslušného uživatelského účtu, jehož součástí je i seznam instalací, ke kterým má daný uživatel přístup. Administrátor klikne na tlačítko „Přidat instalaci“ a aplikace zobrazí formulář pro přiřazení přístupu. Administrátor zadá požadované údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu uloží záznam do databáze. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Úprava přístupu

Administrátor zobrazí formulář pro úpravu příslušného uživatelského účtu a následně klikne na tlačítko „Upravit“ u příslušné instalace. Aplikace zobrazí formulář pro úpravu přístupu s předvyplněnými daty. Administrátor upraví požadované údaje a odešle formulář. Aplikace provede validaci dat a v případě úspěchu záznam v databázi upraví. V opačném případě aplikace vyzve administrátora k zadání validních dat.

Odebrání přístupu

Administrátor zobrazí formulář pro úpravu příslušného uživatelského účtu a následně klikne na tlačítko „Odstranit“ u příslušné instalace. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní záznam z databáze.

Zobrazení seznamu úloh

Uživatel vybere v menu položku „Úlohy“. Aplikace zobrazí seznam uživatelem vytvořených úloh.

Vytvoření úlohy

Uživatel zobrazí seznam úloh, následně klikne na tlačítko „Vytvořit úlohu“ a aplikace zobrazí příslušný formulář pro vytvoření. Uživatel vyplní potřebné údaje, formulář odešle a aplikace uloží záznam do databáze.

Úprava úlohy

Uživatel zobrazí seznam úloh, následně klikne na tlačítko „Upravit“ a aplikace zobrazí příslušný formulář pro úpravu úlohy s předvyplněnými daty. Uživatel upraví požadované údaje, odešle formulář a aplikace záznam v databázi upraví.

Odstranění úlohy

Uživatel zobrazí seznam úloh a následně klikne na tlačítko „Odstranit“ u příslušné úlohy. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní záznam z databáze.

Vytvoření výstupu nebo prováděcího pravidla

Uživatel přejde na obrazovku pro úpravu úlohy a v seznamu výstupů nebo prováděcích pravidel klikne na tlačítko „Vytvořit“. Aplikace zobrazí formulář pro vytvoření šablony výstupu nebo prováděcího pravidla dané úlohy. Uživatel vyplní požadované údaje, odešle formulář a aplikace uloží záznam do databáze.

Úprava výstupu nebo prováděcího pravidla

Uživatel přejde na obrazovku pro úpravu úlohy a v seznamu výstupů nebo prováděcích pravidel klikne u příslušného záznamu na tlačítko „Upravit“. Aplikace zobrazí formulář pro úpravu šablony výstupu dané úlohy nebo prováděcího pravidla s předvyplněnými daty. Uživatel upraví požadované údaje, odešle formulář a aplikace upraví záznam v databázi.

Odstranění výstupu nebo prováděcího pravidla

Uživatel přejde na obrazovku pro úpravu úlohy a v seznamu výstupů nebo prováděcích pravidel klikne u příslušného záznamu na tlačítko „Odstranit“. Aplikace zobrazí potvrzovací dialog a v případě potvrzení odstraní záznam z databáze.

4.4.3 Uložení dat

Sekce popisuje organizaci dat v databázi a uvádí význam jednotlivých entit v aplikaci.

Operator

Tabulka oprávněných uživatelů aplikace sloužící k ověření přístupu při přihlášení. Tabulka uživatelů obsahuje datové položky:

- IDOperator – primární klíč, integer, not null – jedinečné id uživatele,
- Username – varchar(128), not null – uživatelské jméno (email) uživatele,
- Password – varchar(64), not null – hash otisk hesla uživatele využívaný při autorizaci,
- Admin – bit, not null – určuje, zda má daný uživatel administrátorská oprávnění.

Installation

Obsahuje seznam instalací parkovacího systému, ke kterým je možné přiřadit jednotlivým uživatelům přístup. Tabulka obsahuje datové položky:

- IDInstallation – primární klíč, integer, not null – jedinečné id instalace,

- Name – varchar(128), not null – název instalace,
- IP – varchar(64), not null – IP adresa používaná pro připojení k instalaci,
- Port – integer, not null – číslo portu používaného pro připojení k instalaci.

Access

Definuje instalace, ke kterým mají jednotliví uživatelé přístup. Tabulka neobsahuje položku vlastního id jako primárního klíče, záznamy jsou namísto toho jednoznačně určeny dvojicí IDOperator:IDInstallation. Pokud nemá dvojice v tabulce Connection záznam, nemá uživatel k dané instalaci přístup. Tabulka obsahuje datové položky:

- IDOperator – primární klíč, cizí klíč, integer, not null – jedinečné id uživatele,
- IDInstallation – primární klíč, cizí klíč, integer, not null – jedinečné id instalace,
- login – varchar(64), not null – login použitý pro přístup k lokální aplikaci dané instalace,
- password – varchar(64), not null – heslo použité pro přístup k lokální aplikaci dané instalace.

Contact

Záznamy kontaktů, na které je možné odesílat výstupy automaticky spouštěných úloh. Adresář kontaktů je pochopitelně veden pro jednotlivé uživatele zvlášť. Tabulka obsahuje datové položky:

- IDContact – primární klíč, integer, not null – jedinečné id kontaktu,
- IDOperator – cizí klíč, integer, not null – jedinečné id uživatele, který má kontakt v adresáři,
- Name – varchar(128), not null – jméno kontaktu,
- Email – varchar(128), not null – emailová adresa použitá pro odesílání emailů.

Task

Tabulka Task obsahuje záznamy automaticky spouštěných úloh. Tabulka obsahuje datové položky:

- IDTask – primární klíč, integer, not null – jedinečné id úlohy,
- IDConnection – cizí klíč, integer, not null – jedinečné id přístupu, který se použije při spuštění úlohy pro připojení k instalaci,
- IDTemplate – integer, not null – jedinečné id šablony v databázi lokální aplikace, která bude použita pro vytvoření reportu.

TaskParameter

Obsahuje hodnoty proměnných šablon přiřazených k automatickým úlohám. Podle jejich typu jsou hodnoty rozdílným způsobem parsovány. Tabulka obsahuje datové položky:

- IDTaskParameter – primární klíč, integer, not null – jedinečné id hodnoty parametru,
- IDTask – cizí klíč, integer, not null – jedinečné id úlohy, ke které se hodnota proměnné váže,
- Type – tinyint, not null – datový typ proměnné,
- Value – text, not null – hodnota převedená na textový řetězec.

MailTemplate

Obsahuje šablonu emailu, který je odesílán po dokončení odpovídající úlohy. Tabulka obsahuje datové položky:

- IDMailTemplate – primární klíč, integer, not null – jedinečné id šablony zprávy,
- IDTask – cizí klíč, integer, not null – jedinečné id úlohy, ke které daná šablona emailu přísluší,
- IDContact – cizí klíč, integer, not null – jedinečné id kontaktu, který je příjemcem zprávy,
- Title – varchar(128), not null – předmět odesílaného emailu,
- Text – text, not null – text odesílaného emailu.

TaskExecution

Tabulka TaskExecution uchovává prováděcí pravidla, která definují čas a frekvenci spouštění automatických úloh. Tabulka obsahuje datové položky:

- IDTaskExecution– primární klíč, integer, not null – jedinečné id prováděcího pravidla,
- Type – tinyint, not null – typ pravidla, podle kterého je vyhodnocováno, zda má dojít k provedení,
- Month – tinyint – měsíc provedení,
- DayOfWeek – tinyint – den v týdnu, kdy dojde k provedení,
- DayOfMonth – tinyint – den v měsíci, kdy dojde k provedení,
- Hour – tinyint – hodina provedení,
- Minute – tinyint – minuta provedení,
- LastRun – datetime – datum a čas posledního provedení.

Kapitola 5

Implementace

Kapitola věnující se implementaci obsahuje seznam a stručný popis použitých technologií a dále se zabývá konfigurací aplikace a frameworku Spring. Popis samotné implementace je pak pro přehlednost rozdělen podle účelů jednotlivých částí aplikací na prezentační, servisní a datovou vrstvu.

5.1 Použité technologie

Technologie jsou částečně stanoveny nefunkčními požadavky (viz sekce 3.2.2), které vycházejí ze zadání práce nebo požadavků společnosti GREEN Center.

5.1.1 Wildfly

Pro nasazení webových aplikací systému reportů by byl jistě dostačující webový kontejner Apache Tomcat. Zvolen byl však aplikační server Wildfly, a to především z důvodu jeho využití společností také v rámci dalších aktuálních i v minulosti realizovaných projektů.

5.1.2 Databázový server

Lokální aplikace využívá databázi v aktuální podobě, tedy databázový server SQL Anywhere. V případě cloudové aplikace navrhuji použití některého z open source serverů MySQL nebo PostgreSQL. Open source databázové servery se podle mého názoru vyrovnají těm komerčním a jejich výhodou je pochopitelně možnost použití bez nutnosti zakoupení licence. Kromě toho se jedná o menší projekt, který by mohl být ideální pro nasazení dosud nepoužívaných technologií a zjištění, zda by nebylo výhodné použít open source databázový server i v případě databáze parkovacího systému.

5.1.3 Java

Programovacím jazykem použitým pro implementaci webových aplikací v rámci této práce je v souladu se zadáním a s pokyny zadavatele jazyk Java, který je primárním programovacím jazykem používaným ve společnosti GREEN Center. Jazyk Java je interpretovaný, objektově orientovaný jazyk vyvinutý společností Sun Microsystems. Jeho velkou výhodou je nezávislost na architektuře. Zdrojový kód není překládán přímo do strojového kódu, ale pouze tzv. bajtkódu, díky čemuž je výsledná aplikace spustitelná v jakémkoliv prostředí, který má k dispozici interpret Javy – JVM.

5.1.4 JavaScript

JavaScript je multiplatformní skriptovací jazyk, jehož kód je spouštěn až na straně klienta webovým prohlížečem. Obvykle slouží k tvorbě interaktivních prvků uživatelského rozhraní. V implementovaných aplikacích je využit například k zobrazení dialogového okna pro přihlášení.

5.1.5 Spring

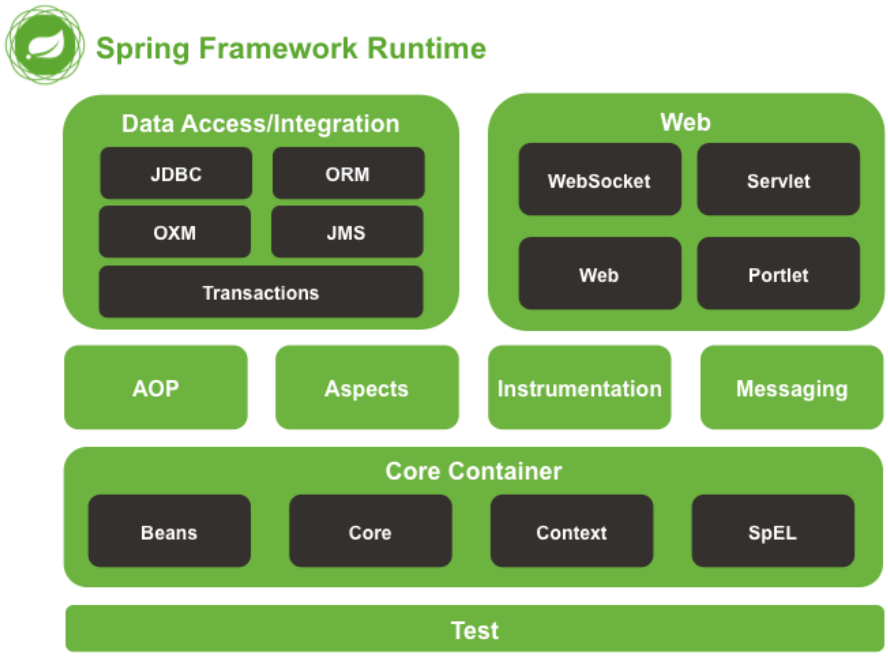
Aplikace využívají backend framework Spring. Framework poskytuje programátorovi sadu nástrojů a funkčností, které usnadňují a zrychlují vývoj aplikací. Základní vlastností frameworku Spring je jeho modulární architektura (viz obrázek 5.1). To umožňuje využití jen těch jeho částí, které jsou pro danou aplikaci potřeba.

V následujících sekcích jsou vysvětleny některé z nejdůležitých principů frameworku Spring.

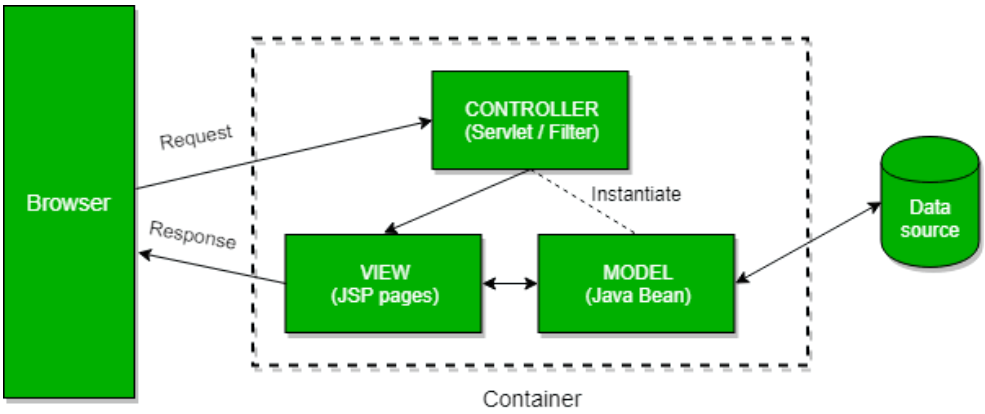
5.1.5.1 Navrhový vzor MVC

V aplikacích je použit frameworkem Spring podporovaný návrhový vzor MVC (Model-View-Controller). Jedná se o architekturu rozdělující zdrojové kódy aplikace do tří logických částí. Model obsahuje aplikační logiku a zajišťuje přístup k databázi a práci s daty. View (dále označován jako šablona) reprezentuje uživatelské rozhraní – zprostředkovává vstupy od uživatele a jemu naopak výsledné výstupy aplikace. Controller reaguje na uživatelské vstupy, zajišťuje volání požadované aplikační logiky a poskytuje šabloně data potřebná k vykreslení uživatelského rozhraní.

Spolupráce jednotlivých částí návrhového vzoru MVC ve frameworku Spring je zobrazeno na obrázku 5.2. V porovnání se standardním zobrazením návrhového vzoru MVC stojí za zmínku přerušovaná čára mezi komponentami kontroler a model. Ta v tomto případě, při správném použití frameworku Spring, představuje specifické provázání zmíněných částí aplikace, které je popsáno ve zbytku této sekce.



Obrázek 5.1: Spring framework – modulární architektura [18]



Obrázek 5.2: Spring framework – navrhový vzor MVC [19]

5.1.5.2 Bean

Jedním z nejdůležitějších pojmů v rámci frameworku Spring je pojem bean. V dokumentaci lze najít následující definici: "In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container"[20]. Volně přeloženo se dá říci, že bean jsou objekty tvořící základ aplikace, a jsou inicializovány a spravovány Spring IoC kontejnerem.

5.1.5.3 Inversion of Control a Dependency Injection

Kromě pojmu bean je rovněž důležité vysvětlit význam s ním úzce souvisejícího návrhového vzoru Inversion of Control a princip fungování IoC kontejneru. Inversion of Control je princip, ve kterém dochází k definování závislostí objektu, aniž by daný objekt své závislosti vytvořil. Vkládání závislosti (Dependency Injection) je delegováno právě na IoC kontejner, který spravuje instance tříd deklarované aplikací jako bean, a tyto instance vloží do příslušných proměnných. Deklarování a inicializace objektů spravovaných IoC kontejnerem je spolu s názornými příklady blíže popsána v části věnované konfiguračním souborům (viz sekce 5.2).

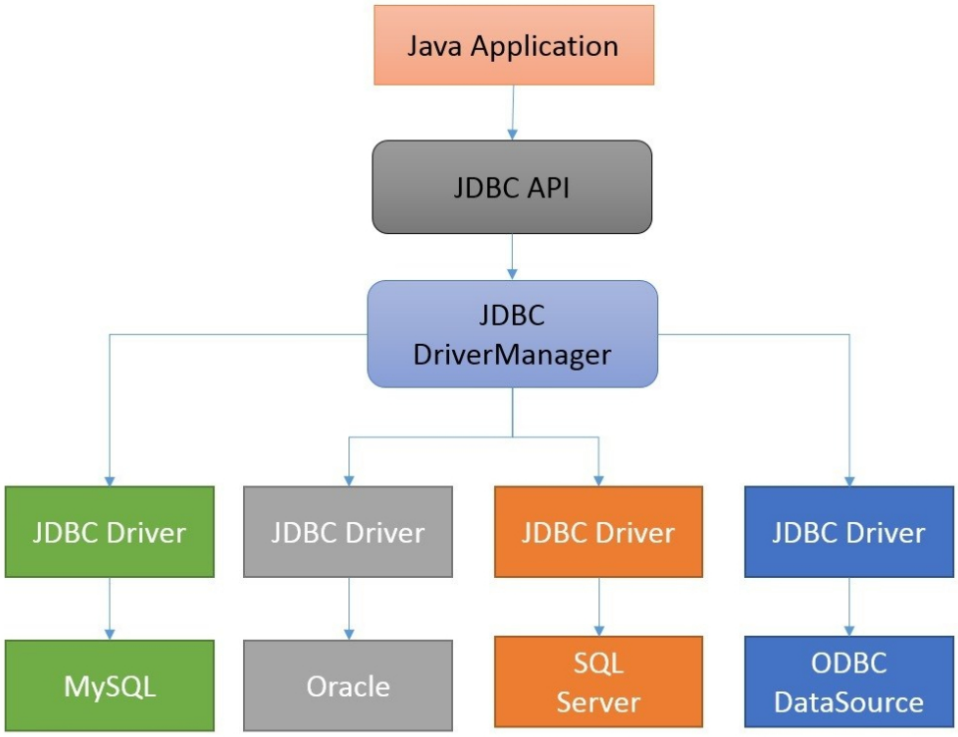
5.1.6 Bootstrap

Frontend framework usnadňuje programátorovi tvorbu uživatelského rozhraní a tím do značné míry zrychluje vývoj webových aplikací. V implementovaných aplikacích jsou za tímto účelem použity knihovny Bootstrap. Jedná se o open source CSS framework s předdefinovanými styly jednotlivých komponent uživatelského rozhraní. Framework Bootstrap je kompatibilní se všemi nejpoužívanějšími prohlížeči a umožňuje i snadnou tvorbu moderních webových stránek s responzivním designem.

5.1.7 JDBC

Java Database Connectivity je rozhraní programovacího jazyka Java pro přístup k relačním databázím a práci s daty v nich obsaženými. Mezi nejpoužívanější rozhraní JDBC patří:

- Driver, Connection – zajišťují spojení s databází, přihlášení a podobně.
- Statement – používá se ke spouštění statických SQL příkazů a získání dat, která jsou výsledkem daného dotazu.
- PreparedStatement – reprezentuje SQL příkaz obsahující proměnné. Proměnné lze nastavit pomocí setterů a příkaz tak spustit opakovaně s různými parametry.



Obrázek 5.3: Architektura JDBC [21]

- CallableStatement – rozhraní používané pro volání procedur uložených v databázi.
- ResultSet – reprezentuje data získaná spuštěním SQL příkazu.

Příklady zdrojových kódů s použitím zmíněných rozhraní jsou uvedeny v sekci popisující implementaci vrstvy persistence (viz sekce 5.5).

Pro přístup ke konkrétnímu databázovému serveru a práci s na něm nasazenou databází je potřeba příslušný JDBC driver, který je poskytovaný autory daného databázového serveru. Například v případě databázového serveru SQL Anywhere se jedná o knihovnu jConnect, která je tak součástí lokální aplikace. Správu JDBC driveru a interakci s ním zajišťuje DriverManager. Diagram komunikace jednotlivých komponent je zobrazen na obrázku 5.3.

5.1.8 JSP a JSTL

Java Server Pages je soubor technologií používaných pro vývoj dynamických webovových stránek s použitím jazyka Java. Do HTML kódu umožňuje vložení Java kódu, který je zpracován na straně serveru. Technologie JSP rovněž umožňuje práci s objekty uložených v proměnných, které jsou šablonám

zpřístupněny pomocí kontrolerů aplikace. JSP Standard Tag Library pak definuje soubor značek, a jejich syntaxi a sémantiku, které je možné v JSP používat.

5.1.9 REST

Architektura rozhraní REST již byla popsána v úvodní části práce (viz **??**). V implementaci je s její pomocí realizována vzájemná komunikace aplikací, přičemž cloudová aplikace je v této komunikaci v pozici klienta a lokální aplikace zastává pozici serveru.

5.1.10 Knihovny

Kromě frameworků usnadňuje programátorovi práci i využití externích knihoven obsahujících funkcionality, které by jinak musel sám zdlouhavě implementovat. Jednou z výhod programovacího jazyku Java je právě i dostupnost velkého množství open source knihoven. V této sekci je uveden seznam a stručný popis externích knihoven, které jsou použity v implementovaných aplikacích.

Apache POI

Apache POI je open source knihovna umožňující práci se soubory v různých formátech Microsoft Office. S pomocí knihovny Apache POI je možné soubory vytvářet, číst i do nich zapisovat. V implementovaných aplikacích je knihovna použita konkrétně při exportu vygenerovaného reportu do formátu XLS. Java sama o sobě totiž rozhraní pro práci se soubory v tomto formátu bohužel neposkytuje.

iText

Open source knihovna iText poskytuje rozhraní pro práci se soubory ve formátu PDF. V aplikacích je použita pro export vygenerovaného reportu do tohoto formátu.

jConnect

Jedná se o JDBC driver nutný k připojení k databázovému serveru SQL Anywhere. Autorem driveru je společnost vyvíjející daný databázový systém, v tomto případě tedy SAP.

5.2 Konfigurace

Konfigurace obou aplikací se zásadněji neliší a obsah konfiguračních souborů je tedy popsán obecně pro obě aplikace současně.

5.2.1 Konfigurační soubor web.xml

Konfigurační soubor */WEB-INF/web.xml* obsahuje veškeré informace o aplikaci, které potřebuje aplikační server k jejímu nasazení. Soubor může obsahovat deklarace servletů, filtrů, listenerů a dalších komponent.

Následující část XML deklaruje servlet s názvem *app-dispatcher*. Servlet je instancí třídy *DispatcherServlet* z balíku *org.springframework.web.servlet* inicializované s parametrem *contextConfigLocation*, který uvádí cestu ke konfiguračnímu XML souboru. Element *load-on-startup* zajišťuje načtení servletu při startu aplikace a jeho hodnota definuje pořadí inicializace v případě více servletů (hodnotou musí být celé kladné číslo, nižší hodnota pak zajišťuje dřívější inicializaci).

```
<servlet>
  <servlet-name>
    app-dispatcher
  </servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-mvc-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Po konfiguraci servletu je třeba deklarovat jeho mapování pomocí elementu *servlet-mapping*. S využitím dříve deklarovaného jména daného servletu a hodnoty elementu *url-pattern* se definují požadavky, které budou servletem zpracovávány.

```
<servlet-mapping>
  <servlet-name>app-dispatcher</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Aplikace bude lokalizována v množství jazyků od češtiny, přes arabštinu, až po čínštinu. Pro správné zobrazení všech znaků a podobu url požadavků je proto potřeba nastavit kódování. K tomu se používají filtry, jejichž účelem je různým způsobem zpracovávat požadavky před tím, než jsou zpracovány servletem, následně mohou upravovat i odpověď serveru před jejím odesláním klientovi. Následuje část konfigurace, která definuje filter s názvem *characterEncodingFilter*, který je instancí třídy *CharacterEncodingFilter* z balíku *org.springframework.web.filter* a prostřednictvím parametrů *encoding* a *forceEncoding* nastavuje kódování UTF-8.

```
<filter>
  <filter-name>characterEncodingFilter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
```

Stejně jako v případě servletu následuje deklarace mapování kódování pomocí elementu *filter-mapping*. Uvedená část konfigurace opět pomocí jména filtru a hodnoty elementu *url-pattern* definuje URL, které využívají dané kódování.

```
<filter-mapping>
  <filter-name>characterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

5.2.2 Konfigurační soubor spring-mvc-config.xml

Konfigurace Spring frameworku je v implementovaných aplikacích umístěna v souboru spring-mvc-config.xml. Přestože v nejnovějších verzích frameworku Spring se lze obejít zcela bez konfiguračních XML souborů a veškerou konfiguraci aplikace přesunout pomocí anotací do souborů se zdrojovými kódy, rozhodl jsem se pro tento zavedený způsob. Konfigurační soubor může obsahovat nastavení spojení s databází, zabezpečení aplikace, inicializace tříd v kontextu aplikace a mnoho dalšího.

Jak již bylo zmíněno, jedním z nejdůležitějších principů frameworku Spring je bean. Bean je možné deklarovat několika způsoby. Následující ukázka demonstruje způsob deklarace pomocí konfiguračního souboru. V deklaraci je uveden název, pomocí kterého může být následně získána instance v aplikaci, a třída, která bean implementuje.

```
<beans>
    <bean name="beanName" class="cz.green.beanClass"/>
</beans>
```

Další možností, která je v implmentovaných aplikacích používána častěji, jsou anotace. Aby však mohly být anotace používány musí být využití

anotačních konstrukcí povoleno. K tomu slouží následující část kódu v konfiguračním souboru.

```
<mvc:annotation-driven />
```

Další krokem před použitím anotací je definování balíků, ve kterých jsou anotavané třídy vyhledávány. V další ukázce části konfigurace je nastavení vyhledávání v kořenovém balíku zdrojových kódů, díky čemuž jsou nalezeny všechny třídy implementované jako bean napříč všemi soubory zdrojových kódů.

```
<context:component-scan base-package="cz.green" />
```

5.2.3 Anotace

Třídu lze označit jako bean několika anotacemi. Základní z nich je anotace *@Component* deklarující, že třída je komponentou framwerku Spring. Další uvedené anotace jsou rozšířením základní anotace a mohou takto deklarované třídě přidávat některé funkce.

- @Component – anotace používaná pro obecnou komponentu spravovanou frameworkem Spring.
- @Controller – anotace pro třídy v prezentační vrstvě. Povoluje mapování URL na jednotlivé metody kontroleru.
- @Service – anotace pro třídy v servisní vrstvě. Tyto třídy by měly obsahovat logiku aplikace a přistupovat k vrstvě persistence. Aktuálně se významněji neliší od obecné anotace @Component a jediným smyslem jejího použití je zatím označení účelu anotované třídy.
- @Repository – anotace pro třídy ve verstvě persistence. Funkcí stojící za zmínku je automatické zachytávání platformně specifických výjimek a jejich překlad na příslušného potomka třídy *DataAccessException*.

Pro umožnění automatického překladu výjimek je třeba deklarovat bean *PersistenceExceptionTranslationPostProcessor*.

```
<bean class="org.springframework.dao.annotation
  .PersistenceExceptionTranslationPostProcessor"/>
```

Výše uvedené anotace se používají pro celé třídy. Další možností jak deklarovat bean je anotace *@Bean*, která je umisťována před metodu. Následující část kódu je ekvivalentní s předchozí částí konfiguračního souboru.

```
@Bean
public PersistenceExceptionTranslationPostProcessor x() {
  return new PersistenceExceptionTranslationPostProcessor();
}
```

Pro jednotlivé objekty bean lze pomocí anotace *@Scope* rovněž nastavit jejich životní cyklus.

- @Scope("singleton") – dochází k vytvoření jediné instance v rámci celé aplikace. Tato možnost je defaultní v případě, že atribut není nastaven.
- @Scope("prototype") – při každém vyžádání je vytvořena nová instance.
- @Scope("request") – pro každý HTTP požadavek je vytvářena nová instance.
- @Scope("session") – instance je platná v rámci jedné HTTP session, po jejím zaniknutí zaniká rovněž daná instance.

Nyní se dostáváme zpět k principu IoC. Vytvoření instancí je delegováno na IoC kontenjer využívající deklarace v konfiguračním souboru nebo pomocí anotací. Následně je potřeba vytvořit požadované závislosti. K tomu je určena anotace *@Autowired*, jejíž použití zařídí inicializaci proměnné při vytvoření instance třídy. V případě, že není nalezen odpovídající bean, Spring vyhodí výjimku a build aplikace skončí neúspěchem. Následuje ukázka použití anotace *@Autowired* v jednom z controllerů.

```
@Autowired
private ReportService reportService;
```

5.2.4 Konfigurace aplikací

Samotné aplikace mají rovněž vlastní konfigurační soubor ve formátu XML. Jejich formát je definován v samostatném souboru pomocí jazyka DTD, který se používá pro popis struktury XML dokumentu. Následující řádek určuje soubor definující strukturu konfiguračního souboru a současně je v něm uveden název kořenového elementu.

```
<!DOCTYPE root SYSTEM "webreport_local.dtd">
```

Soubor DTD umoňuje předepsat elementy a jejich atributy, které je možné nebo nutné v konfiguraci uvést a jejich vzájemné uspořádání. Příkladem je předpis definující podobu konfigurace připojení k databázovému serveru.

```
<!ELEMENT dbConnection>
<!ATTLIST dbConnection
  driver CDATA #REQUIRED
  url CDATA #REQUIRED
  username CDATA #REQUIRED
  password CDATA #REQUIRED
>
```

CDATA je v DTD datový typ, kterému vyhovuje textový řetězec, #REQUIRED pak značí povinnost uvedení atributu. Část konfiguračního souboru odpovídající uvedené definici může vypadat následovně.

```
<dbConnection
  driver="com.sybase.jdbc3.jdbc.SybDriver"
  url="jdbc:sybase:Tds:127.0.0.1:2638/database"
  username="username"
  password="password"
/>
```

Kromě uvedeného spojení s databázovým serverem nabízí konfigurační soubory možnost definovat:

- IP adresu a port pro připojení k logovací službě,
- defaultní jazyk v případě lokální aplikace,
- název souboru a cestu k uložení generovaných reportů (k dispozici jsou proměnné, které jsou nahrazeny v době vytváření souboru – například jednotlivé části aktuálního data a času, nebo název reportu),
- v případě cloudové aplikace SMTP server a přihlašovací údaje pro odesílání emailů.

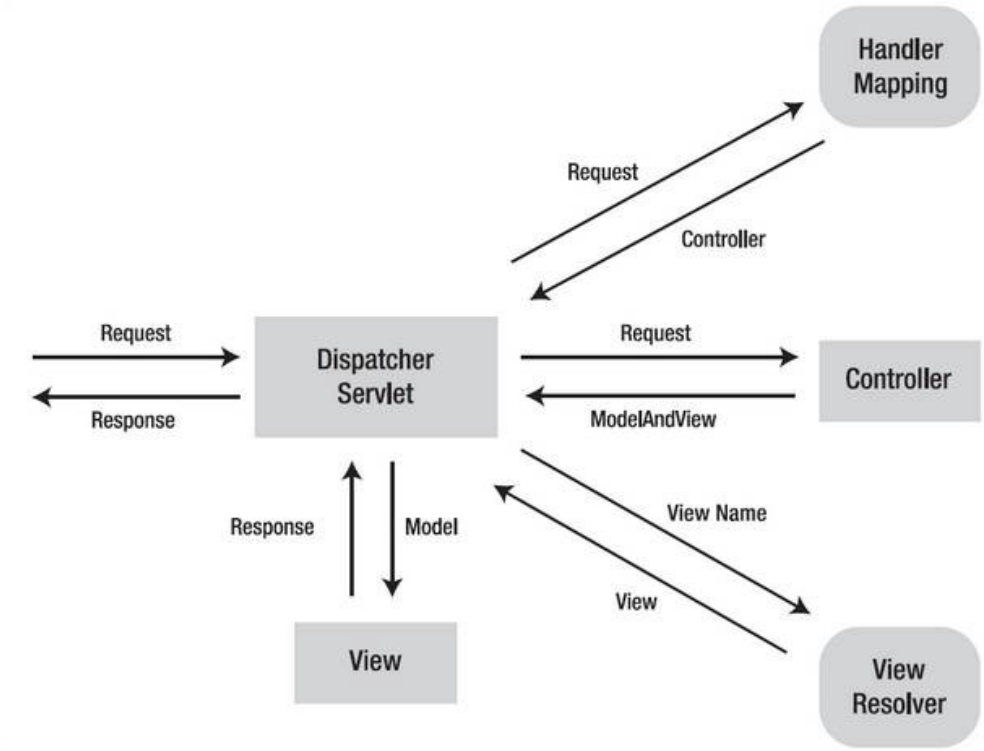
5.3 Prezentační vrstva

Úkolem přezentační vrstvy ve vícevrstvé architektuře je zobrazení uživatelského rozhraní a předání vstupů uživatele ke zpracování. Do prezentační vrstvy jsou v tomto případě řazeny šablony a kontrolery.

5.3.1 Dispatcher Servlet

O směřování požadavku na konkrétní kontroler a jeho odpovídající metodu se stará framework Spring, konkrétně komponenta Dispatcher Servlet. Jejím úkolem je koordinace jednotlivých komponent, které zajišťují dílčí úkony potřebné pro zpracování požadavku a následně odeslání odpovědi uživateli. Dispatcher Servlet je implementací návrhového vzoru front controller, komunikace s dalšími komponentami je znázorněna na obrázku 5.4.

Po obdržení požadavku je pomocí cílové adresy URL a rozhraní Handler Mapping přiřazena příslušná metoda kontroleru, který je již implementován konkrétní webovou aplikací. Návratovou hodnotou metody zpracovávající požadavek je objekt ModelAndView. Jedná se o třídu definovanou frameworkem Spring, jejíž funkcí je sdružení objektů Model a View, aby mohly být společně předány jako návratová hodnota. Obsahem modelu je mapa, ve které jsou pomocí klíče (textového řetězce) adresovány datové struktury. Šablona



Obrázek 5.4: Spring framework – Dispatcher Servlet [22]

(view), která bude použita k vykreslení dat modelu, může být určena řetězcem obsahujícím její název nebo přímo objektem šablony. V případě definování šablony jejím názvem je následně získán odpovídající objekt šablony pomocí komponenty View Resolver. Po získání objektu šablony je nakonec vygenerována a odeslána odpověď na přijatý požadavek.

5.3.2 Kontroler

Aby mohl být požadavek směřován na konkrétní metodu kontroleru, je potřeba jí namapovat na příslušnou URL. To je možné udělat několika způsoby, v aplikacích je použito mapování pomocí anotace *@RequestMapping*.

```
@RequestMapping(method =
  RequestMethod.GET, value = "/template/{id:.+}")
public ModelAndView detail(@PathVariable("id") int anId) {
  ModelAndView model = new ModelAndView("template/detail");
  Template template = templateDAO.get(anId);
  model.addObject("template", template);
  return model;
}
```

Aby bylo mapování úspěšné, je nutné sdělit frameworku Spring, v jakých třídách mapování hledat. Každý kontroler proto musí být označen anotací *@Controller*.

```
@Controller
public class TemplateController extends AbstractController {

  ...

}
```

Deklarovaný kontroler z ukázky (i všechny ostatní) je potomkem abstraktní třídy AbstractController. Jejím účelem je implementace tříd a metod, které jsou využívány všemi kontrolery. Příkladem jsou třídy reprezentující informační hlášky a metody pro jejich vytváření a následné vložení do modelu a vykreslení v šabloně.

```
public abstract class AbstractController {

  ...

  protected abstract class FlashMessage {

    protected String type;
    protected String message;

    public String getType() {
      return type;
    }

    public String getMessage() {
      return message;
    }
  }

  public class SuccessMessage extends FlashMessage {

    protected SuccessMessage(String aMessage) {
      type = "alert-success";
      message = aMessage;
    }
  }

  ...

}
```

Kromě typu success obsahuje implementace typy info, warning a danger. Jedná se o názvy CSS tříd definovaných knihovnami Bootstrap, které se již automaticky postarají o formátování vypisovaných sdělení.

5.3.3 REST API

Webové rozhraní je z hlediska implementace v podstatě jen speciálním případem kontroleru. Rozdílná je specifická anotace *@RestController* a návratová hodnota jednotlivých metod, která v tomto případě není instancí třídy ModelAndView. Na následujících řádcích kódu je ukázána implementace metody rozhraní poskytující objekt šablony reportu.

```
@RequestMapping(method = RequestMethod.GET,
  value = "/api/template/{id:.+}")
public TemplateREST template(@PathVariable("id") int anId) {
  Template template = templateDAO.getTemplate(anId);
  ArrayList<ParameterGroup> params = null;
  try {
    params = reportService.buildStructure(user, template);
  } catch (Exception e) {
    ...
  }
  return template.toRestTemplate(windows);
}
```

Na tomto místě je důležité zdůraznit, že implementace rozhraní není kompletní – účelem současné podoby implementace, jejímž zásadním nedostatkem je chybějící zabezpečení, je umožnění testování cloudové aplikace. Návratový typ metody a způsob přenosu dat je popsán společně s druhou stranou komunikace v sekci věnované servisní vrstvě (viz sekce 5.4.6).

5.3.4 Předzpracování požadavku

Před samotným zpracováním požadavku kontrolerem může být vhodné vykonat další akce. K tomu slouží ve frameworku Spring kompomponenta Interceptor, která poskytuje rozhraní pro práci s požadavkem a odpovědí v různých fázích zpracování. Za tímto účelem definuje metody:

- preHandle() – volána před zpracováním kontrolerem,
- postHandle() – volána po zpracování kontrolerem,
- afterCompletion() – volána po vyřízení požadavku.

V jednotlivých metodách mohou být prostřednictvím parametrů k dispozici objekty požadavku (HttpServletRequest), odpovědi (HttpServletResponse) nebo třeba návratové hodnoty metody kontroleru (ModelAndView).

Prostřednictvím rozhraní HandlerInterceptor je v aplikacích implementován vlastní způsob autorizace. Pomocí metody *preHandle()* dochází k ověření, zda je uživatel přihlášen a rovněž zda má přístup do požadované části aplikace. Při neúspěšném ověření je požadavek zamítnut a odeslán příslušný chybový kód.

```
public class AuthorizationInterceptor
  implements HandlerInterceptor {

  @Override
  public boolean preHandle(HttpServletRequest req,
                           HttpServletResponse res,
                           Object handler) throws Exception {
    User user = (User) req.getSession().getAttribute("user");
    if (user == null) {
      res.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
      res.sendRedirect(req.getContextPath());
      return false;
    }
    return true;
  }

  ...
}
```

Instance jsou pochopitelně opět spravovány IoC kontejnerem. Aby se tak dělo, je třeba použít příslušné anotace, nebo následujícím způsobem interceptor deklarovat v konfiguračních souborech.

```
</mvc:interceptors>
  <mvc:interceptor>
    <mvc:mapping path="/**"/>
    <mvc:exclude-mapping path="/"/>
    <mvc:exclude-mapping path="/login"/>
    <bean class="cz.green.interceptor.AuthorizationInterceptor"/>
  </mvc:interceptor>
</mvc:interceptors>
```

Součástí deklarace je definování požadavků pomocí cílových URL adres, při jejichž zpracovávání bude inteceptor použit. V popisovaném případě autorizace tedy vždy, s výjimkou domovské stránky a požadavku na přihlášení. Obdobným způsobem jsou chráněny i části aplikace, ke kterým mají přístup pouze uživatelé s administrátorským oprávněním. Pomocí interceptoru a metody *postHandle()* je rovněž do každého modelu před vykreslením šablony vložena instance služby TextContainer, poskytující přístup k souborům s lokalizačními texty.

5.3.5 Šablona

Po vykonání logiky aplikace a shromáždění potřebných dat zbývá vykreslit výslednou HTML stránku. K tomu ve frameworku Spring slouží technologie JSP a JSTL.

V ukázce z kontroleru je vidět, že požadovaná šablona, která má být použita, je definována pouze svým názvem. To umožňuje následující nastavení komponenty View Resolver, jehož účelem je usnadnění vyhledávání interních zdrojů. K názvu šablony je automaticky doplněna definovaná předpona (cesta k šablonám) a přípona (koncovka souboru .jsp).

```
<bean class="org.springframework.web.servlet.view
              .InternalResourceViewResolver" >
  <property name="prefix">
    <value>/WEB-INF/views/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>
```

Pozn.: umístění třídy v balících musí být na jednom řádku, zalomeno kvůli sířce stránky

Důvodem pro umístění šablon do adresáře WEB-INF je znemožnění veřejného přístupu k těmto souborům pomocí URL.

Šablony jsou tvořeny z velké části klasickým HTML kódem, do něhož jsou vkládány značky. Ve zbývající části sekce jsou uvedeny ukázky některých nejčastěji použitých značek.

Pro přístup k proměnným modelu slouží znak dolaru společně se složenými závorkami.

```
<tr>
  <th>Name</th>
  <td>${template.name}</td>
</tr>
```

Mezi nejdůležitější a nejčastěji používané značky patří podmínka. Zarážející je v JSTL absence konstrukce else, která tak v případě potřeby musí být nahrazena další značkou if s inverzní podmínkou.

```
<c:if test="${user.isAdmin()}">
  ...
</c:if>
```

Velmi užitečná je značka zavádějící for cyklus.

```
<c:forEach items="${users}" var="user">
   ...
  <td>${user.email}</td>
   ...
</c:forEach>
```

Jednotlivé JSP soubory je možné do sebe vkládat a vnořovat, čímž lze zabránit zbytečným duplicitám a opakování stejné části kódu v několika šablonách.

```
<jsp:include page="footer.jsp">
```

5.4 Servisní vrstva

Aplikační logiku, nebo alespoň rozhraní pro přístup k ní, je zvykem sdružovat do tříd service. Jedná se opět o singletony spravované IoC kontejnerem, proto je nutné service deklarovat v konfiguraci nebo, jako v implementovaných aplikacích, příslušnou anotací.

```
@Service
public class ReportService {
   ...
}
```

5.4.1 ReportService

Služba poskytuje rozhraní pro práci se šablonami reportů, a to po celý její životní cyklus – od parsování z XML, přes generování reportu, až po export dat do souboru. Nejdůležitější veřejné metody rozhraní jsou:

- ArrayList<ParameterGroup> buildStructure(User, Template) – zpracuje předaný objekt šablony, který byl načten z databáze. Z proměnné obsahující šablonu v XML vygeneruje objekt reportu a v návratové hodnotě předá seznam objektů reprezentujících proměnné šablony.

- void setVariables (Map<String, String>) – zajišťuje parsování hodnot proměnných šablony předaných v parametru do příslušných datových typů. Validitu proměnných zajišťuje frontend aplikace, i tak ale v případě neúspěšného parsování metoda signalizuje nemožnost pokračovat výjimkou.

- ReportModel generate() – metoda spouští proces generování dat reportu a vrací objekt, který jej reprezentuje.

5.4.2 Exporter

Implementovány jsou čtyři služby zajišťující export dat vygenerovaného reportu do souboru v daném formátu:

- ExporterPDF,
- ExporterCSV,
- ExporterXML,
- ExporterXLS.

Všechny z nich jsou součástí lokální i cloudové aplikace. Implementují stejné rozhraní ExporterInterface obsahující, kromě jediné metody pro zahájení procesu exportu, především gettery. Kromě toho mají ještě společného předka, abstraktní třídu AbstractExporter, od které dědí například metodu vytvářející název souboru.

5.4.3 MailSender

Imlementuje odesílání emailu včetně možností připojení souborů jako příloh. Služba je využívána třídou zajišťující automatické spouštění úloh v cloudové aplikaci. K implementaci jsou kromě standardních Java tříd využity i některá rozhraní frameworku Spring, především pak z balíku org.springframework.mail.javamail. Zdrojový kód je díky tomu velmi stručný.

```
public void sendEmail(String aReceiver, String aSubject,
                      String aText, List<File> anAttachments)
                      throws MessagingException {
  MimeMessage message = emailSender.createMimeMessage();
  MimeMessageHelper helper = new MimeMessageHelper(message, true);
  message.setFrom(Config.mailUsername);
  helper.setTo(aReceiver);
  helper.setSubject(aSubject);
  helper.setText(aText);
  for (File attachment : anAttachments) {
    FileSystemResource file = new FileSystemResource(attachment);
    helper.addAttachment(file.getFilename(), file);
  }
  emailSender.send(message);
}
```

Do proměnné emailSender je pomocí anotací a následující metody při inicializaci vložena instance rozhraní JavaMailSender. Nastavení SMTP serveru je načteno z konfiguračního souboru.

```
@Autowired
public JavaMailSender emailSender;

@Bean
public JavaMailSender getJavaMailSender() {
  JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
  mailSender.setHost(config.mailHost);
  mailSender.setPort(config.mailPort);
  mailSender.setUsername(config.mailUsername);
  mailSender.setPassword(config.mailPassword);

  Properties props = mailSender.getJavaMailProperties();
  props.put("mail.transport.protocol", "smtp");
  props.put("mail.smtp.auth", "true");
  props.put("mail.smtp.starttls.enable", "true");
  return mailSender;
}
```

5.4.4 TaskExecutor

Služba zajišťuje spouštění automatických úloh v cloudové aplikaci. Ty mohou být různého typu a spouštěny v odlišnou dobu a s různou frekvencí, proto je potřeba v pravidelných intervalech kontrolovat, zda má být úloha spuštěna. I pro tento účel poskytuje framework Spring užitečný nástroj v podobě anotace *@Scheduled*. Ta zajišťuje automatické volání anotované metody podle zadaných parametrů. Základ implementace je zachycen v následující ukázce.

```
@Configuration
@EnableScheduling
public class TaskExecutor {

  ...

  @Scheduled(fixedRate = 60000)
  private void checkTasks() {
    Calendar now = Calendar.getInstance();
    now.setTime(new Date());
    for (Task task : taskDAO.getAll()) {
      for (TaskExecution execution : task.getExecutions()) {
        if (execution.isToExecute(now)) {
            ...
  }

  ...
```

```
}
```

Vlastní vykonání úlohy je spouštěno v samostatném vlákně, protože může být časově náročnější a převážnou část doby zpracování tvoří čekání na data, která musí být vygenerována na instalaci, na níž se úloha dotazuje.

Pro umožnění automatického volání metody je potřeba příslušnou třídu označit anotací *@EnableScheduling*. Parametr *fixedRate* pak definuje frenkvenci v milisekundách, při tomto nastavení je tedy metoda volána každou minutu. Automaticky volaná metoda by neměla přijímat žádné parametry a měla by mít návratovou hodnotu void. Služba neobsahuje žádné veřejné metody a její využití je plně v režii frameworku Spring a IoC kontejneru.

5.4.5 TextContainer

Služba TextContainer slouží pro načítání lokalizačních textů. Při inicializaci jsou načteny všechny soubory s překlady textů a uloženy do mapy za použití zavedených locale identifikátorů (např. cs_CZ) jako klíče. Stejný identifikátor je rovněž použit pro rozlišení jednotlivých souborů. Šablony pak prostřednictvím rozhraní služby získávají texty příslušející aktuálně nastavenému jazyku.

```
@Service
public class TextContainer {

  private Map<String, ResourceBundle> resource;

  public TextContainer() {
    resources = new HashMap<String, ResourceBundle>();
    for (Language language : languageDAO.getAll()) {
      try {
        resource.put(language.getLocale(), ResourceBundle
          .getBundle("properties/Webreport_cloud_"
          + language.getLocale()));
      } catch (Exception e) {
        ...
      }
    }
  }

  public String getText(String aLocale, String aKey) {
    try {
      return resources.get(aLocale).getString(aKey);
    } catch (Exception e) {
      return aKey;
    }
```

```
    }
}
```

5.4.6 RESTService

Prostřednictvívím této služby je z cloudové aplikace přistupováno k REST API lokální aplikace na jednotlivých instalacích. Služba poskytuje rozhraní používané při vytváření automatických úloh a při jejich následném spouštění:

- DirectoryREST getTemplateStructure(Installation) – načtení adresářové struktury z instalace předané v parametru,
- TemplateREST getTemplate(Installation, int) – získání objektu šablony podle jejího id a instalace předaných v parametru,
- ReportDataREST getReport(Installation, int, Map<String, String>) – vygenerování reportu. V parametrech je předán objekt reprezentující instalaci, id šablony a mapa s názvy a příslušnými hodnotami jejích proměnných.

Datové typy použité jako návratové hodnoty jsou jednoduché třídy bez aplikační logiky obsahující pouze proměnné potřebné na straně cloudové aplikace. Instance těchto tříd pak lze pomocí frameworku Spring a jeho rozhraní RestTemplate mezi aplikacemi jednoduše přenést. Objekty jsou na straně příjemce požadavku převáděny do formátu JSON, a na straně odesílatele do objektu stejného datového typu parsovány. V následující části kódu je zobrazena ukázka implementace metody pro získání šablony.

```
public TemplateREST getTemplate(Installation anInstalation,
    int aTemplateId) {
  String ip = anInstallatin.getIp();
  int port = anInstallatin.getPort();
  RestTemplate restTemplate = new RestTemplate();
  TemplateREST template = null;
  try {
    template = restTemplate.getForObject("http://" + ip
    + ":" + port + "/localapp/api/template/"
    + aTemplateId, TemplateREST.class);
  } catch (Exception e) {
    ...
  }
  return template;
}
```

Strana implementující webové rozhraní je popsána v sekci zabývající se prezentační vrstvou (viz sekce 5.3.3).

5.5 Datová vrstva

Účelem datové vrstvy je perzistentní uchování dat a poskytnutí rozhraní pro přístup k těmto datům. O samotné uchování se stará databáze a databázový server, ke kterému je přistupováno pomocí rozhraní JDBC. Technicky je v zásadě možné provádět libovolné SQL dotazy napříč všemi částmi zdrojových kódů, je však více než vhodné komunikaci s databází oddělit od zbytku aplikace a vytvořit jednotné rozhraní. Pomocí něj mohou další komponenty aplikace k datům přistupovat, aniž by musely vědět, jak jsou databázové operace implementovány.

5.5.1 DAO

Data Access Object je návrhový nebo strukturální vzor pro odstínění aplikační logiky od zdroje dat a implementace práce s nimi. Základem je následující rozhraní definující základní CRUD operace.

```
public interface DAO<T> {

  T get(int id);
  List<T> getAll();
  void create(T entity);
  void update(T entity);
  void delete(T entity);

}
```

Jak je z ukázky vidět, rozhraní DAO využívá generického parametru, který je v jednotlivých implementacích nahrazen POJO třídami. POJO třídy jsou obyčejné objekty, které nejsou spravovány IoC kontejnerem, obvykle neobsahují ani žádnou logiku a slouží pouze k reprezentaci dat. Pomocí implementací rozrhaní DAO jsou tedy jednoduše mapována data z relační databáze na POJO objekty. Následující ukázka kódu je příkladem implementace DAO rozhraní pro entitu Property.

```
public class PropertyDAO implements DAO<Property> {

 private static final String SQL_GET_ALL =
   "SELECT * FROM DBA.RTProperty ORDER BY Language";

  @Override
  public List<Property> getAll() throws SQLException {
    ArrayList<Property> retval = new ArrayList<Property>();
    Connection con = connection.getConnection();
    synchronized (con) {
      PreparedStatement ps;
```

```
            ps = con.prepareStatement(SQL_GET_ALL);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
              String key = rs.getString("Key");
              int language = rs.getInt("Language");
              String value = rs.getString("Value");
              Property dir = new Property(language, key, value);
              retval.add(dir);
            }
            rs.close();
            ps.close();
            con.commit();
          }
          return retval;
        }

}
```

V ukázce jsou vynechány ostatní metody rozhraní jejichž implementace je podobná. Kromě implementace rozhraní DAO je zde rovněž vidět použití JDBC, v tomto případě konkrétně rozhraní Connection, PreparedStatement a ResultSet.

Jednotlivé implementace rozhraní DAO jsou označeny anotací *@Repository* a tedy spravovány IoC kontejnerem. IoC kontejner vytvoří při inicializaci právě jednu instanci (singleton) každé implementace a ta je následně vložena do příslušných proměnných v aplikaci. Výhodou takovéto implementace je, že jednotlivé části aplikace nemusí mít přístup ke všem datům, ale pouze k těm, které pro svou práci potřebují.

Kapitola 6

Testování

Kapitola se věnuje testování aplikací a zhodnocení výsledků testování. Samotné testování je rozděleno na tři nezávislé části:

- Testování logiky aplikací – logikou jsou myšleny funkčnosti, které aplikace nabízejí (správné generování dat a jejich export).
- Testování uživatelského rozhraní – testování frontendu aplikací.
- Automatické testy – unit testy.

6.1 Logika aplikací

Cílem těchto testů je ověření, zda probíhá korektně parsování XML šablon reportů, generování konkrétních datových struktur reportu a jejich export do požadovaných formátů. Pro tento účel je důležitá původní desktopová aplikace u níž předpokládáme, že dané funkčnosti implementuje bez chyby.

V této fázi testování jsou tedy použity obě aplikace na jedné databázi a pomocí stejných šablon se stejnými hodnotami proměnných jsou generovány různé reporty. Faktem usnadňujícím práci je, že výstupy ve formátech CSV a XML jsou velmi snadno porovnatelné strojově. Pro ověření správnosti generovaných dat byly tedy porovnány obsahy exportovaných souborů ve formátu CSV. Při vědomí, že strojové porovnání exportovaných CSV souborů skončilo úspěchem a generovaná data jsou tudíž správná, stačí u ostatních formátů exportovaných souborů zběžné porovnání zaměřené například na počty záznamů.

Pro účel porovnání dvou datových souborů ve formátu CSV existuje velké množství nástrojů a nebylo tudíž nutné vlastní porovnání implementovat. Při testování byl konkrétně využit online komparátor ze sady nástrojů ExtendsClass [23].

V průběhu testování byl vygenerován minimálně jeden report z každé z 59 existujících šablon za použití dat z reálné instalace parkovacího systému. Porovnání generovaných souborů neodhalilo žádné rozdíly mezi výstupy nově

implementované aplikace a dosud používané desktopové aplikace. Lze tedy říci, že v tomto směru fungují aplikace bezchybně.

6.2 Automatické testy

Testy na úrovni jednotlivých komponent aplikace je vhodné automatizovat. Automatické testy znamenají obvykle výraznou úsporu času při testování jednotlivých částí kódu a je možné spouštět je opakovaně, například po refactoringu. K automatickému testování je využit framework JUnit sloužící právě k testování menších částí zdrojového kódu.

Vzhledem k tomu, že testy jsou nezávislou částí zdrojových kódů, je v některých případech testování nutné inicializovat třídy, které jsou jinak spravovány IoC kontejnerem. K tomu slouží anotace *@RunWith*. Prostřednictvím této anotace je knihovně JUnit možné předat informaci, že testy implementované v anotované třídě vyžadují vložení závislostí. Následující část kódu implementuje test jedné ze služeb poskytující přístup k objektům v databázi, konkrétně služby PropertyDAO a její metody pro vytvoření překladové položky v databázi.

```
@RunWith(SpringJUnit4ClassRunner.class)
public class PropertyDAOTest {

  @Autowired
  PropertyDAO propertyDAO;

  @Test
  @Transactional
  @Rollback(true)
  public void testStoreProperty() {
    Property p1 = new Property(1, "key", "value");
    propertyDAO.create(p);

    List<Property> list = propertyDAO.getAll();
    Property p2 = list.get(list.size() - 1);
    assertEquals(p2.getLanguage(), p1.getLanguage());
    assertEquals(p2.getKey(), p1.getKey());
    assertEquals(p2.getValue(), p1.getValue());
  }

  ...
}
```

Implementace automatických testů není v tuto chvíli kompletní, záměrem je však pokrýt unit testy většinu dílčích funkčností aplikací.

6.3 Uživatelské rozhraní

Vzhledem k tomu, že vývoj aplikací nebyl v tuto chvíli kompletně dokončen a aplikace nejsou připraveny k nasazení do produkčního prostředí (viz kapitola 7), neproběhlo prozatím uživatelské testování. Pro tento účel jsou připraveny testovací scénáře pokrývající funkcionality aplikace. Scénáře není třeba popisovat, obecně vzato kopírují funkční požadavky (viz sekce 3.2.1) a případy užití (viz sekce 4.3.2 a 4.4.2).

Kapitola 7

Zhodnocení a další vývoj

V závěrečné kapitole je zhodnocena odvedená práce z ekonomického pohledu společnosti GREEN Center, shrnut aktuální stav implementace s uvedením částí aplikací, jejichž implementaci je potřeba dokončit před nasazením aplikací do produkčního prostředí, a navržen další postup týkající se výstupů této práce.

7.1 Parkovací systém

Část práce týkající se návrhu nové architektury celého parkovacího systému je brána jako úvodní zamyšlení nad možností využití cloud computingu v parkovacím systému spolešnosti GREEN Center. Záměrem nebylo vytvoření pevně dané architektury nebo přesné definování rozhraní, ale spíše zamyšlení se nad možnostmi a prozkoumání směrů, kterými by mohla vést cesta při zdokonalování systému. V tomto směru je, dle mého názoru, práce dobře použitelná, v budoucnu se na ní dá stavět a její výstupy případně dále rozvíjet.

7.2 Systém reportů

Hlavním výstupem diplomové práce, použitelným v krátkodobějším horizontu, je systém pro generování reportů. Jeho nasazení v současné podobě parkovacího systému je možné rozdělit do dvou fází.

1. Nasazení lokální aplikace – implementuje funkčnosti obou dosud existujících desktopových aplikací. Při nasazení bez druhé aplikace v cloudu nebude uživateli scházet žádná aktuálně dostupná funkcionalita.

2. Nasazení cloudové aplikace – implementuje zcela nové funkcionality.

Před nasazením lokální aplikace je třeba dokončit implementaci lokalizace do světových jazyků. Rozhraní v servisní vrstvě, které zajišťuje načtení

souborů s překladovými texty a uchovává a poskytuje dalším částem aplikace jednotlivé texty, je připraveno a úspěšně otestováno. Zbývá shromáždit texty zobrazované aplikací, přiřadit jim klíče a vytvořit tak výchozí lokalizační soubory v angličtině. Tyto soubory mohou být následně přeloženy do všech používaných jazyků. Kromě toho je potřeba upravit šablony, ve kterých bude místo textu použita návratová hodnota metod služby poskytující rozhraní pro překlad příslušného klíče. V následujícím seznamu jsou dále uvedeny funkcionality, které by, kromě lokalizace, bylo před prvním nasazením dobré implementovat, ale jejichž absence nasazení a použití v produkčním prostředí v zásadě nebrání.

- Filtrování, vyhledávání – součástí funkčních požadavků je umožnit uživateli data, která jsou výsledkem generování reportu, filtrovat a vyhledávat v nich. Jedná se o funkcionalitu novou, v desktopové aplikací neposkytovanou, proto by bylo možné nasadit aplikaci i bez těchto funkčností.

- Zabezpečení REST API – rozhraní bude využíváno výhradně druhou z aplikací vyvíjených v rámci této práce. Z tohoto důvodu není do jejího dokončení nutné rozhraní poskytovat. K tomu aby nebylo dostupné v aktuálním nezabezpečeném stavu, stačí dočasně upravit konfiguraci frameworku Spring a neinicializovat kontroler, který rozhraní implementuje. Před hromadným nasazením by však bylo vhodné rozhraní zabezpečit a vystavit i v době, kdy nebude využíváno. Odpadne tak nutnost pozdějšího upgradu na novější verzi aplikace.

Zatímco lokální aplikace je téměř připravena k nasazení, zdrojový kód cloudové aplikace je třeba ještě doplnit. Funkcionalita, která nutno říci není přiliš složitá, je implementována a otestována, ale některé části aplikace čekají na dokončení.

- Implementace DAO tříd – v současné chvíli aplikace pro účely testování využívá mock objekty, které udržují data namísto databáze.

- Zabezpečení REST API – situace již byla popsána o několik řádků výše. Zabezpečení rozhraní je před nasazením cloudové aplikace nutné implementovat na obou stranách komunikace. Přestože se nejedná o nijak zvlášť citlivá data, jejich ochrana, a od ní se odvíjející důvěryhodnost společnosti v očích zákazníků, je důležitá.

- Lokalizace – aktuální stav implementace je stejný jako v případě lokální aplikace, zbývá vytvořit soubory s lokalizovanými texty a ty v šablonách nahradit klíči a voláním služby pro jejich překlad.

Kromě bodů uvedených v seznamech zbývá vyřešit některé časově méně náročné nedodělky týkající se ve většině případů uživatelského rozhraní.

S ohledem na budoucí vývoj pak lze navrhnout implementaci dalších funkcí, které nejsou v původním zadání obsaženy.

- Import reportu – vzhledem k tomu, že aplikace obsahuje funkci pro filtrování a vyhledávání v generovaných datech, mohla by rovněž umožňovat exportované reporty ze souboru načíst a v aplikaci s nimi dále pracovat. Podporované formáty exportu XML a CSV jsou pro tento účel ideální.
- Rozšíření funkcí automatických úloh – úlohy by do budoucna nemusely sloužit jen ke generování reportů. Z jednotlivých instalací by bylo možné v daných intervalech automaticky stahovat například logovací soubory.

7.3 Náklady a přínosy pro společnost

Hodnocení nákladů a přínosů této práce z ekonomického hlediska není úplně jednoduché, jelikož v případě jejích výstupů se nejedná o systém nebo aplikace, které by přímým způsobem zvyšovaly obrat nebo generovaly zisk společnosti. Projekt webových aplikací pro tvorbu reportů je však součástí dlouhodobější vize a konkrétních plánů rozvoje společnosti. Díky jejich plnění, neustálým inovacím, modernizaci nabízených produktů a zkvalitňování poskytovaných služeb si společnost drží svoji pozici na trhu a dokáže oslovovat nové zákazníky.

Přínosem aplikací pro generování reportů je jednoznačně rozšíření a zkvalitnění poskytovaných služeb, což vede ke zvýšení spokojenosti stávajících zákazníků. Není přitom ojedinělé, že zákazníky jsou obchodní nebo sportovní centra, hotely a další zařízení s několika pobočkami nebo stejným majitelem či provozovatelem. Spokojenost stávajících zákazníků pak zvyšuje šanci na pokračování spolupráce a realizaci dalších zakázek. Stejně tak dochází samozřejmě i k budování dobrého jména, pozice na trhu a tím pádem i snazšímu získávání nových zákazníků.

Právě ve zvýšení komfortu a usnadnění práce se systémem spatřuji hlavní přínos této práce a implenentovaných aplikací pro společnost. V souladu s předchozím odstavcem může vést k uzavření nových zakázek, což už znamená reálný finanční přínos, který se však nedá spolehlivě vyčíslit.

Co se týče nákladů, samotnou implementací bylo dosud stráveno přibližně 160 člověkohodin. Kompletní dokončení implementace, testování, vytvoření dokumentace, uživatelských příruček, instalačních scriptů a podobně je odhadováno na dalších 120 až 160 člověkohodin. Jestli se tato konkrétní investice z ekonomického hlediska vyplatí je opět těžké posoudit, osobně se však domnívám, že ano. Implementace nového systému reportů totiž aktivně přispívá ke konkurenceschopnosti nabízených služeb. V případě funkce automatického generování reportů pak zřejmě vytváří i konkurenční výhodu, jelikož podle veřejně dostupných dokumentací tuto funkci žádný z kokurenčních systémů nenabízí.

Závěr

Cílem práce byl návrh a implementace prototypu systému pro vytváření reportů z jednotlivých instalací parkovacího systému společnosti GREEN Center. Vzhledem k tomu, že součástí práce byla rešerše technologií týkajících se cloud computingu a jedna z implementovaných aplikací bude nasazena na cloudových serverech, rozhodl jsem se nad rámec zadání práce zamyslet i nad možným využitím cloud computingu obecně v celém parkovacím systému. K tomuto rozhodnutí zároveň přispěl i fakt, že návrh systému reportů se nakonec ukázal být jednodušší, než se v době zadání práce mohlo zdát.

V úvodu jsem provedl analýzu současného stavu parkovacího i reportovacího systému. Identifikoval jsem výhody i nevýhody současné podoby architektury parkovacího systému v lokální síti a prověřil výhodnost případné změny architektury jak z pohledu společnosti GREEN Center, tak z pohledu jejích zákazníků, tedy poskytovatelů parkovacích služeb. Současně jsem provedl analýzu funkčních a nefunkčních požadavků kladených na systém reportů.

Na základe předchozí analýzy jsem navrhnul možnou změnu architektury parkovacího systému, popsal rozložení jednotlivých komponent a možné způsoby vzájemné komunikace. Součástí návrhu je i nová podoba reportovacího systému, který sestává z dvou webových aplikací, z nichž jedna bude poskytována jako cloudová služba a druhá instalována lokálně na jednotlivých instalacích parkovacího systému. Při případné změně architektury celého parkovacího systému by pak veškeré funkčnosti poskytovala pouze aplikace v cloudu.

Po provedení návrhu jsem implementoval funkční prototyp obou webových aplikací, který však prozatím není nasaditelný do produkčního prostředí. Současnou podobu implementace obou aplikací jsem řádně otestoval a části, které zbývá implementovat pro možnost nasazení aplikací do produkčního prostředí, jsem popsal v poslední kapitole práce.

Na závěr jsem se v souladu ze zadáním pokusil zhodnotit přínosy této práce pro spolešnost GREEN Center z pohledu nákladů a benefitů.

Závěrem lze tedy konstatovat, že zadaný cíl práce se podařilo splnit. Výstupem práce je prototyp systému reportů, který sestává z dvojice

webových aplikací, které budou po dokončení implementace používány v produkčním prostředí parkovacího systému GREEN Center. získávání reportů, souhrnů a statistik z provozu parkovacího systému společnosti GREEN Center. Vedlejším výstupem práce je pak návrh dalšího možného vuýužití cloud computingu v rámci parkovacího systému.

Literatura

[1] O společnosti GREEN Center [online]. [Citováno 5.1.2020]. Dostupné z: https://www.green.cz/clanek-o-nas-52-137

[2] The Largest Survey Ever of Java Developers [online]. 11 2018, [Citováno 17.12.2019]. Dostupné z: https://blogs.oracle.com/javamagazine/the-largest-survey-ever-of-java-developers

[3] IBM Closes Landmark Acquisition of Red Hat for $34 Billion [online]. 7 2019, [Citováno 17.12.2019]. Dostupné z: https://www.redhat.com/en/about/press-releases/ibm-closes-landmark-acquisition-red-hat-34-billion-defines-open-hybrid-cloud-future

[4] Oracle Ends Commercial Support For GlassFish [online]. 11 2013, [Citováno 17.12.2019]. Dostupné z: https://www.webpronews.com/oracle-ends-commercial-support-for-glassfish/

[5] Fiala, A.: Porovnání vybraných databázových systémů. 2016, [Citováno 17.12.2019]. Dostupné z: https://theses.cz/id/afm98n/

[6] DB-Engines Ranking – Trend Popularity [online]. [Citováno 17.12.2019]. Dostupné z: https://db-engines.com/en/ranking_trend

[7] Method of calculating the scores of the DB-Engines Ranking [online]. [Citováno 17.12.2019]. Dostupné z: https://db-engines.com/en/ranking_definition

[8] Oracle Database Express Edition [online]. [Citováno 17.12.2019]. Dostupné z: https://www.oracle.com/cz/database/technologies/appdev/xe.html

[9] ICT Pro – Školení Databáze [online]. [Citováno 17.12.2019]. Dostupné z: https://www.skoleni-ict.cz/kategorie/Databaze-db.aspx

[10] Počítačová škola Gopas – Kurzy databáze [online]. [Citováno 17.12.2019]. Dostupné z: https://www.gopas.cz/Kurzy/Katalog-kurzu/Databaze.aspx

[11] Editions and supported features of SQL Server 2019 – SQL Server [online]. [Citováno 17.12.2019]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15

[12] Srovnání nejrozšířenějších databázových severů pro výuku i praxi [online]. [Citováno 5.1.2020]. Dostupné z: https://profinit.eu/blog/srovnani-nejrozsirenejsich-databazovych-severu-pro-vyuku-i-praxi/

[13] Fielding, R.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, University of California, Irvine, 2000, [Citováno 5.1.2020]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[14] Gryc, V.: Srovnání komunikačních technologií a výběr efektivního protokolu pro přístup IoT zařízení k back-end serveru s využitím platformy Java. 2018, [Citováno 5.1.2020]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=173408

[15] SOAP vs. REST Comparison: Differences in Performance, APIs & More [online]. [Citováno 5.1.2020]. Dostupné z: https://stackify.com/soap-vs-rest/

[16] W3Counter: Global Web Stats – December 2019 [online]. [Citováno 17.12.2019]. Dostupné z: https://www.w3counter.com/globalstats.php?year=2019&month=12

[17] SSLSocket (Java Platform SE 8) [online]. [Citováno 8.1.2020]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/javax/net/ssl/SSLSocket.html

[18] Joshi, N.: Spring Modular Architecture [online]. 7 2018, [Citováno 5.1.2020]. Dostupné z: https://www.programmingmitra.com/2018/07/spring-modular-architecture.html

[19] Spring MVC Interview Questions [online]. 12 2018, [Citováno 5.1.2020]. Dostupné z: https://www.baeldung.com/spring-mvc-interview-questions

[20] Spring documentation – Core Technologies [online]. [Citováno 5.1.2020]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-introduction

[21] Gupta, S.: JDBC Architecture [online]. 7 2018, [Citováno 5.1.2020]. Dostupné z: https://facingissuesonit.com/tag/jdbc-architecture/

[22] Lasnitzki, G.: Using Spring in Web Applications - Spring Study Notes [online]. [Citováno 5.1.2020]. Dostupné z: http://springcert.sourceforge.net/2.5/9-study-web-app.html

[23] Extends Class – CSV compare [online]. [Přístup 20.3.2019]. Dostupné z: https://extendsclass.com/csv-diff.html

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CDI Context and Dependency Injection

CPU Central Processing Unit

CRUD Create, Read, Update, Delete

CSV Comma Separated Values

DAO Data Access Object

DBMS Database Management System

DTD Document Type Definition

EJB Enterprise Java Beans

GB Gigabyte

GPL General Public License

HP-UX Hewlett Packard UniX

HTTP Hypertext Transfer Protocol

IoC Inversiont of Control

IP Internet Protocol

Java EE Java Enterprise Edition

JBoss EAP JBoss Enterprise Application Platform

JDBC Java Database Connectivity

JMS Java Messagaging Service

JSF JavaServer Faces

JSON JavaScript Object Notation

JSP JavaServer Pages

JVM Java Virtual Machine

PB Petabyte

PDF Portable Document Format

POJO Plain Old Java Object

Q&A Questions and Answers

REST Representational State Transfer

SaaS Software as a Service

SMS Short Message Service

SOAP Simple Object Access Protocol

SSL Secure Sockets Layer

TCP Transmission Control Protocol

UTF-8 8-bit Unicode Transform Format

WSDL Web Services Description Language

WS-Serurity Web Service Security

XML Extensible Markup Language

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