EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/B/2017/ 36093656567262724

ANALÝZA, NÁVRH A IMPLEMENTÁCIA WEBOVEJ APLIKÁCIE NA REZERVÁCIU PARKOVACÍCH MIEST

Bakalárska práca

2017 **Bc. Jana Valentínyová**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Analýza, návrh a implementácia webovej aplikácie na rezerváciu parkovacích miest

Bakalárska práca

Študijný program: Hospodárska informatika

Študijný odbor: 9.2.10 Hospodárska informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: Ing. Ján Pittner, PhD.

Bratislava 2017 **Bc.Jana Valentínyová**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracovala samostatne a že som uviedla všetku použitú literatúru.

Dátum: .....

ABSTRAKT

VALENTÍNYOVÁ, Jana: Analýza, návrh a implementácia webovej aplikácie na rezerváciu parkovacích miest. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Ing. Ján Pittner, PhD – Bratislava: FHI, 2017, 44.
Cieľom záverečnej práce je vytvorenie webovej aplikácie v jazyku Java, ktorá bude slúžiť ako systém na rezerváciu parkovacích miest. Práca je rozdelená do 4 kapitol. Obsahuje 0 grafov, 0 tabuliek a 1 prílohu. Prvá kapitola je venovaná popisu súčasnej problematiky danej oblasti. Druhá kapitola je venovaná cieľu práce, špecifikácii funkčných a nefunkčných požiadaviek aplikácie. V tretej kapitole opíšeme architektúru systému. Záverečná kapitola sa zaoberá výsledkom práce a teda implementáciou webovej aplikácie. Výsledkom riešenia danej problematiky je zdieľaný parkovací systém vo forme webovej aplikácie.

Kľučové slová:
Webová aplikácia, Objektovo-orientované programovanie, Java EE, Parkovací systém

ABSTRACT

VALENTÍNYOVÁ, Jana: Analysis, design and implementation of web application as a parking reservation system. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. – Ing. Ján Pittner, PhD – Bratislava: FHI, 2017**,** 44.

The goal of the thesis is to create web application in Java language, which will be used as system for reservation of parking places. Thesis is divided into 4 chapters. Contains 0 graphs, 0 tables and 1 attachment. First chapter is dedicated to a description of an anctual problems in this area. Second chapter is about aim of the thesis, specification of functional and nonfunctional requirements of the application. In the third chapter we analyze the architecture of a system. Final chapter describes result of the thesis - implementation of the web application. The results of solution to the problem consist of shared parking system in form of a web application.

Keywords:

Web application, Object-oriented programming, Java EE, Parking system

OBSAH

Zoznam použitých skratiek

API Application Programming Interface

CRUD Create, Read, Update, Delete

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IOT Internet Of Things

Java EE Java Enterprise Edition

JPA Java Persistence API

JSON JavaScript Object Notation

MVC Model-View-Controller

ORM Object Relational Mapper

SQL Structured Query Language

URL Uniform Resource Locator

XML eXtensible Markup Language

XHTML Extensible Hypertext Markup Language

Zoznam obrázkov

ÚVOD

Preprava do firmy a parkovanie vie byť niekedy nočná mora. Problém vzniká, keď chcú napríklad osoby na neznámom mieste zaparkovať, ale počet miest nie je dostačujúci a tak nevýhodne parkujú vo veľkej vzdialenosti od ubytovania, či zamestnania. Okrem neznámeho prostredia problémom býva aj nutnosť parkovať svoj automobil v predražených parkovacích domoch. Takisto doprava osôb na dlhšie vzdialenosti, či už do dovolenkových destinícií ako napríklad hotelov a apartmánov alebo doprava osôb na služobné cesty do zahraničia, kde firma prenajíma ubytovacie zariadenie by malo byť zabezpečené nie len z hľadiska ubytovania ale aj parkovania a prepravy. Oba prípady predstavujú riziko komplikácií, stratu času a dodatočné náklady.

Nami vyvíjaná aplikácia sa preto pokúsi navrhnúť riešenie na komplikácie ktoré vznikajú, keď by si osoba rada zarezervovala aj ideálne parkovacie miesto a v prípade záujmu aj zviezla cestujúcich. Môžu to byť napríklad kolegovia, ktorí aplikáciu zdieľajú alebo návštevníci hotelového zariadenia a teda dôveryhodné osoby z hľadiska platieb. Ubytovanie je väčšinou zabezpečené jednoducho, ale parkovanie na niektorých miestach je obtiažne a niektoré hotely ho nemajú v ponuke alebo vôbec nezabezpečujú.

Cieľ mojej práce sa skladá z dvoch častí. Spočiatku sa oboznámiť s problematikou tvorby webových aplikácií v jazyku Java, preskúmať výhody a prípadné nevýhody danej tvorby a následne vytvoriť systém, ktorý by zefektívňoval parkovanie prostredníctvom využitia internetu. V dnešnej dobe sa parkovanie stáva čoraz komplikovanejším a efektívne rezervovanie miest vie priniesť časové, finančné a energetické úspory. Výhodou webovej aplikácie oproti desktopovej aplikácii je ten, že sa dá spustiť prakticky z akéhokoľvek zariadenia, či už je to mobil, tablet, laptop alebo stolový počítač a v reálnom čase pracovať s prenosnou aplikáciou. Výhodou, ktorú by sme radi využili, je aj možnosť doinštalovania nami vyvinutého softvéru do už existujúceho softvéru firmy. Takýto systém má priniesť inovatívny prístup k rezervácii miest a spĺňaní rôznych funkcionalít parkovania. Systém má byť jednoduchý, používateľsky ľahko prístupný a v budúcnosti ho bude možno jednoducho rozširovať a dopĺňať. Systém bude mať na starosti administátor, ktorý bude spravovať jeho činnosť.

1 Súčasný stav riešenej problematiky a úvodná analýza nášho riešenia

1.1 „Neinteligentné parkovanie – inteligentné autá“

Podľa nedávnych štúdií firmy Frost & Sullivan sa očakáva , že 100 miliardový svetový priemysel v oblasti parkovania pritiahne strategické investície vo výške 200 až 250 miliónov, v priebehu ďalších troch až piatich rokov, najmä na rozvoj inovácií a inteligentného parkovania. Hlavným dôvodom pre takýto prílev kapitálu, je odstránenie neefektívnosti v spôsobe akom parkujeme. Donedávna sa parkovanie a inovácie miešali ako olej a voda. Dôvodom pre takýto pomalý vývoj, je hlavne pomaly sa meniaci charakter dvoch hlavných obastí parkovania, parkovanie na ulici a parkovanie mimo ulice. Parkovanie priamo na ulici, ktoré reprezentuje tretinu zo všetkých príjmov z parkovania v Spojených štátoch, je zvyčajne kontrolované mestami a samosprávou. Zodpovedné osoby v tejto oblasti nie sú tí, ktorí by hýbali inováciami čo najrýchlejšie, keď sa jedná o technické zmeny a i keď tu prišlo k progresu v rámci platby automatmi, inovatívne parkovacie projekty, ktoré pomáhajú vodičom nájsť parkovacie miesta sú pomerne ojedinelé. Parkovanie mimo ulíc (napríklad v garážach), ktoré reprezentuje približne tretinu všetkých príjmov je hlavne vo vlastníctve súkromných firiem, a preto teoreticky by sa malo hýbať rýchlejšie vpred pokiaľ ide o inovácie. Ale aj tento segment parkovania bol pomerne pomaly sa meniaci.[11]

Automobilová technika sa naopak stala tak pokročilou, že auto sa stalo prakticky počítačom na kolesách. Ďalším zaujímavým vývojom prešlo auto, ktoré je kompletne pripojené na internet. Jedná sa inteligetný automobil, ktorý je označovaný ako prvé internetové vozidlo, čo je odkaz na internet vecí (IoT), kde je prepojené všetko vrátane auta, smartfónu a spotrebičov. Podstatné je, že v tomto aute sa nachádzajú zaujímavé inteligentné funkcie – umožní napríklad šoférovi objednať si parkovacie miesto cez vstavaný softvér a na toto miesto samostatne zaparkovať. [11]

Inovácie v oblasti parkovania a parkovacích systémov sú teda veľmi žiadané, ba priam očakávané a veľké množstvo investičného kapitálu je vyhradených priamo pre riešenia v tejto oblasti. V našej práci sa pokúsime o zlepšenie a zjednodušenie podmienok pre parkovanie typu off-street (mimo ulíc) najmä v súkromnej sfére, pomocou vyvinutia aplikácie slúžiacej ako integrovaný parkovací systém.

1.2 Zdieľaná ekonomika – fenomén súčasnosti

V našej práci sa pokúsime implementovať princíp zdieľanej ekonomiky v oblasti poskytovania a rezervovania spolujazdy ako doplnkovej funkcionality k riadeniu parkovacieho procesu, aby sme demonštrovali ako využívanie zdieľaných prostriedkov napomáha k efektívnejšiemu narábaniu so zdrojmi.

Zdieľaná ekonomika predstavuje v dnešnej dobe jav, kedy sú produkty na trhu prípadne služby ponúkané jednotlivcami napríklad domácnosťou a nie firmou. Výhody takéhoto novodobého javu sú významné. Nielen, že jednotlivci majú možnosť výberu z väčšieho množstva produktov a to prináša viac možností na trhu, ale aj ponúkajúci majú možnosť zárobku. Príkladom zdieľanej ekonomiky môže byť napríklad ak si chceme požičať nejaký nástroj, ktorý použijeme len jednorázovo a neoplatí sa nám ho zakúpiť. [15]

Prostredníctvom **informačných technológií** sa takýto systém obchodovania stal ľahký a vysoko efektívny. Čoraz viacej ľudí sa zapája do tohto systému s cieľom ušetriť náklady alebo získať finančný výnos. V budúcnosti sa odhaduje čoraz väčší nárast tohto sektora ako náprotivok tradičného sektora. [15]

1.3 Výhody riešenia danej problematiky

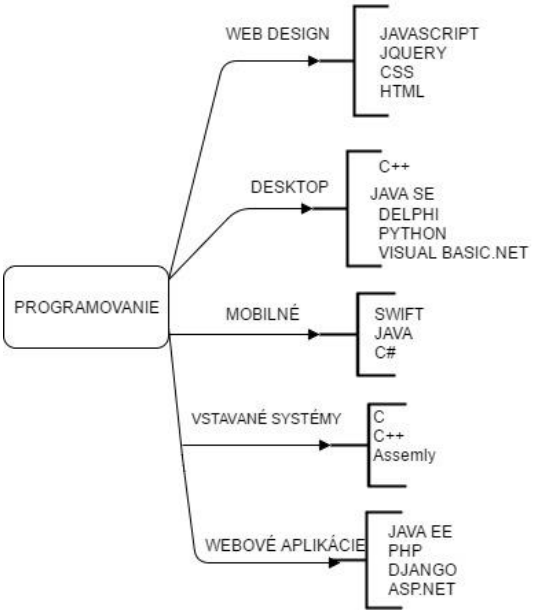
Aplikácia pre parkovanie automobilov môže priniesť výrazné zníženie nákladov ale i zníženie stresu. Takisto sa môže predísť konfliktom medzi zamestnancami, ktorí by vyjadrovali nesúhlas, prípadne „spamovali“ kolegov s prosbou o preparkovanie auta, ktoré na miesto nesprávne zastalo. Aplikácia by mohla priniesť prehľad o aktuálnych rezervovaných miestach pre všetkých zamestnancov, čo by najmä vo veľkých firmách odlahčilo manažérov o riešenie konfliktov, respektíve o zadeľovaní prázdnych miest. Ďalšia automatizácia by mohla prísť v oblasti prepravy, kde by sa kolegovia/iný užívatelia aplikácie mohli zviesť spolu a ušetriť si tým každodenné náklady. Dôležitým faktom ostáva, že do menej frekventovaných lokalít nie je možné tak ľahko nájsť dopravný prostriedok a nie každý má možnosť ísť vlastným dopravným prostriedkom. Firma prípadne hotel nie vždy dopravu zabezpečuje a naopak ísť sám autom je vysoko nákladné.

1.4 Výhody webovej aplikácie

Dôvod, pre ktorý sme si zvolili práve webovú aplikáciu na riešenie nášho bakalárskeho projektu spočíva v jej charaktere a výhodách, ktoré prináša. Webová aplikácia je aplikácia

sprístupňená prostretníctvom internetu (intranetu), ktorá beží na strane servra a je zobrazená vo webovom prehliadači používateľa (architekrúra typu klient-server). Webová aplikácia sa od webovej stránky odlišuje vo svojej zložitosti, i keď niekedy je zložité ba až nemožné odlíšiť ich. Webová aplikácia obvykle využíva databázu a obsahuje zložitejšiu business logiku na strane servra ako webová stránka. Webová aplikácia sa odlišuje od desktopovej aplikácie tým, že nie je nutnosť sťahovať si ju do počítača, je teda všadeprítomná z každého zariadenia, nutné je iba pripojenie na internet. Výhodou je teda dostupnosť a takisto aktualizácie prebiehajú na strane poskytovateľa oproti desktopovej, v ktorej užívateľ musí sám aktualizovať verziu. Pre nami vyvíjanú aplikáciu, ktorá predpokladá väčší počet užívateľov pristupujúcich k nej z rôznych zariadení predstavuje webová aplikácia pre vymenované dôvody i ako si ďalej ukážeme aj architektúru vhodné riešenie.

1.5 Zdôvodnenie vybranej technológie pre webovú aplikáciu



Obrázok 1 Druhy programovacích jazykov, podľa typu vývoja

Ako na obázku vidíme, programovacie jazyky vieme rozčleniť podľa oblastí typu vývoja, či už sú to mobilné aplikácie, webový dizajn, webový vývoj aplikácie a pod. Podobnou cestou sme uvažovali aj my, keď sme sa rozhodovali v akom jazyku budeme vyvíjať webovú aplikáciu, aby spĺňala uvedené funkčné aj nefunkčné požiadavky na vyvíjaný systém.

Webový dizajn aplikácie bude zabezpečený prostredníctvom všetkých spomenutých jazykov uvedených na obrázku, otázkou bola voľba najvhodnejšieho programovacieho jazyku na strane servra. V prvej verzii by mal systém fungovať v podobe webovej aplikácie, čím by

bol systém prístupný z každého zariadenia (mobilný telefón, počítač, tablet,..) ale v budúcnosti by sa mal rozšíriť o mobilnú verziu v operačnom systéme Android. Webová, desktopová i mobilná verzia využíva programovací jazyk Java, a nakoľko sme mali skúsenosti s týmto jazykom zvolili sme si vývoj aplikácie v jazyku **Java EE.**

Aplikácie v Androide sú písané v objektovo orientovanom programovacom jazyku Java a práve tento jazyk sme použili aj v backende našej aplikácie. Dôvodom bolo jednoduchšie implementovanie aplikácie pre mobilnú verziu.

Dôvodom pre ktorý sme pri výbere silne preferovali práve enterprise verziu Javy určenú na vývoj webových aplikácií je i ten, že predpokladáme potencionálne využitie systému veľkými firmami, prípadne hotelovými rezortmi a Java EE je svojimi riešeniami určená práve pre túto oblasť.

Java je zároveň jazyk, ktorý je najrozšírenejším programovacím jazykom na svete a z hľadiska uplatnenia na trhu práce je jedným z najvyhľadávanejších programovacích jazykov.

Dôležitá črta tohto jazyka je, že jeho programy sa nekompilujú do strojového jazyka predstavujúceho nuly a jednotky, ktorému rozumie počítač ale do medzistupňa, ktorý je spustiteľný na rôznych typoch počítačov a zariadení. Jazyk Java sa vyskytuje v rôznych verziách/ balíkoch, podľa toho, pre akú sféru daný program vyvíjame. Najznámejšia je Java SE (Standard Edition, typicky inštalovaná pre domácnosti a užívateľov), mobilná verzia Java ME (Micro Ediotion) a Java EE(Enterprise Ediotion) určená pre sektor firiem.

Naša práca bude teda vyvíjaná prostredníctvom Javy EE, ktorá obsahuje rozličné knižnice, frameworky, návrhové vzory a podobne. V tejto časti si objasníme pojem **framework**, ktorý je nutné vysvetliť, pretože je jeden z nosných technológií používaných v našom projekte. Framework nám má ulahčiť tvorbu projektov, pretože odstraňuje nutnosť písať kód, ktorý sa často v aplikáciách opakuje ako napríklad otázky bezpečnosti, autentifikácie a podobne a my sa potom môžeme sústrediť na programovanie našej aplikácie. Odstaňuje teda pomyselnú „byrokraciu“ v programovaní aplikácií a zefektívňuje a ulahčuje vývoj. V Jave EE frameworky zohrávajú dôležitú úlohu ako napríklad JSF, Spring a ďalšie, ktoré si neskôr opíšeme v našej práci podrobne.

2 Cieľ práce

Po úvodnej časti v ktorej sme stručne špecifikovali stav riešenej oblasti a zmysel vývoja systému, charakterizujeme aké funkcie by mala aplikácia spĺňať a aké požiadavky sú na systém kladené. Tento krok je je z hľadiska vývoja softvéru kľúčovým krokom, ktorý riadi celý smer vývoja aplikácie. Požiadavky delíme na dva druhy a to funkčné a nefunkčné.

2.1 Funkčné požiadavky

Funkčné požiadavky určia akú minimálnu funkcionalitu by mala aplikácia spĺňať. Vzhľadom na to, že berieme do úvahy už existujúce riešenia v tejto sfére, máme v zájme poskytnúť parkovací softvér dodržujúci princíp zdieľanej ekonomiky. Integrácia takýchto funkcií má za cieľ zjednodušenie a zefektívnenie parkovania, ako aj celkového prepravného procesu.

Systém by mal slúžiť na:

- rezerváciu parkovacích miest

- pridávanie nových parkovísk a parkovacích miest do aplikácie administrátorom

-prehľad obsadených parkovacích miest

-prehľad poskytovanej spolujazdy, ktorú je možnosť si na uvedenom kontakte rezervovať

- poskytnutie spolujazdy

- odstraňovanie spolujazdy administrátorom

- informovanie administrátora priamo v systéme o vyžiadaných zmenách v spolujazde

2.2 Nefunkčné požiadavky

Nefunkčné požiadavky vyplývajú jednak zo zadania práce, ktorá určuje vyvíjaný softvér ako webovú aplikáciu a taktiež špecifkujú vlastnosti systému:
- aplikácia sa musí dať zobraziť prostredníctvom internetu z akéhokoľvek zariadenia
- dáta musia byť uložené na strane servra v databáze namapovanej do objektov objektovo orientovaného jazyka
- autorizácia a autentifikácia užívateľov

Potom ako sme si vysvetlili význam vývoja takejto aplikácie a vymenovali aké funkčné a nefunkčné požiadavky by mala aplikácia spĺňať sa zameriame na analýzu trhu užívateľov a prevádzkovateľov aplikácie.

2.3 Cieľová skupina

Cieľová skupina poskytovateľov našej aplikácie by mohli byť hotely, väčšie alebo menšie firmy doma alebo v zahraničí, alebo aj firmy, ktoré poskytujú parkovacie služby a chcú zverejniť online parkovanie, ktorý doteraz tieto služby neposkytovali v softvérovej podobe. Užívatelia budú zamestnanci firiem alebo ľudia rezervujúci si miesto v hoteli.

Z toho vyplýva že aplikácia by mala byť viacpoužívateľská, prístupná kedykoľvek a z akéhokoľvek zariadenia.

3 Metodika práce

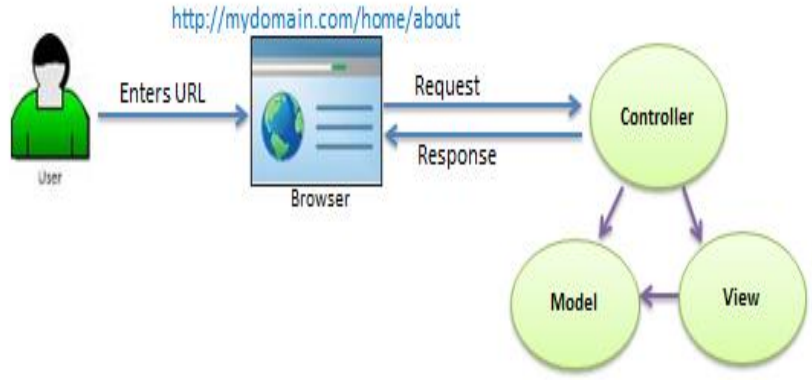
Pred samotným vývojom aplikácie sme sa museli podrobne zaoberať otázkou, ako bude vyzerať softvérová, dátová a technologická architektúra tohto systému. V tejto kapitole opíšeme akú architektúru bude mať vyvíjaná aplikácia a aké softvérové riešenie sme využívali pre čo najefektívnejšiu funkcionalitu aplikácie. Opíšeme aké konfigurácie sme vytvorili a akú stavbu majú jednotlivé časti aplikácie. V nasledujúcej kapitole zameranej na implementáciu si potom vysvetlíme už konkrétne implementované funkcie.

Vývoj webovej aplikácie písanej v Java EE sú vo všeobecnosti trojvrstvové aplikácie využívajúce architektonický vzor MVC.

3.1 MVC architekrúra a Java EE aplikácie

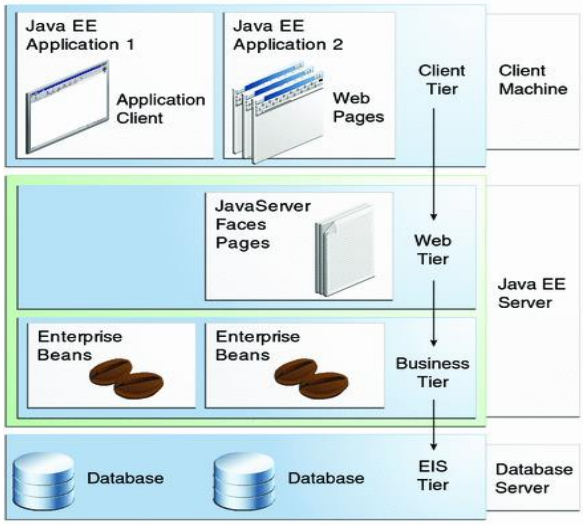
MVC architektúra rozdeľuje architetúru na 3 časti tak, aby bolo možné v prípade potreby každú časť upravovať samostatne a dopad zmien na ostatné časti bol čo najmenší. Tieto časti architektúry sa volajú Model, View, Controller. Model reprezentuje dáta uložené v aplikácii ale aj tú časť aplikácie, ktorá zastrešuje business logiku. Je dôležité si uvedomiť tento fakt, pretože väčšina ľudí nesprávne interpretuje model ako len dátový model databázy. View predstavuje užívateľské rozhranie a teda všetko čo užívateľ vidí a s čím prichádza pri manipulácii do kontaktu. Controller sa nachádza vo väčšine prípadov použitia ako vrstva medzi View a Model a nachádza sa tu aplikačná logika aplikácie a takpovediac má na starosti tok udalostí medzi prezentačnou a dátovou vrstvou. [12]

Nasledujúca ukážka zobrazuje aké vzťahy prebiehajú medzi jednotlivými vrstvami aplikácie využívajúcej architekrútu MVC:



Obrázok 2 MVC Architektúra ([12] - dňa 8.11.2016)

Viacvrstvové aplikácie v Jave EE ako trojvrstvové aplikácie sú teda distribuované cez 3 vrstvy: klientskú, Java EE server a databázu.



Obrázok 3 Viacvrstvová aplikácia v Jave EE ([4] - dňa 8.9.2016)

Java EE komponenty sú takisto písané v programovacom jazyku Java a kompilované sú tiež rovnako ako ostatné programy v tomto jazyku. Rozdiel medzi Javovskými komponentami a štandardnými triedami je ten, že komponenty sú zoskupené do aplikácie Java EE a sú uvedené do produkcie, kde bežia na Java EE servri. [4]

Java EE server je vlastne implementácia niekoľkých špecifikácií postavená na Jave SE.[7] Zoznam najpoužívanejších špecifikácií:

- *JPA – Java Persistence Api* – objektovo-relačné mapovanie
- *JTA – Java Transaction Api* – podpora transakcií serverom a perzistenými úložiskami
- *CDI – Context and Dependency Injection* – implementácia Dependency Injection vzoru
- *JAX-WS – Java Api for XML Web Services* –SOAP služby
- *JAX-RS – Java Api for RESTful Web Services* –REST sieťové služby
- *Servlety* – štandard na rôzne požiadavky prichádzajúce po sieti
- *JSP – JavaServer Pages* –statická web stránka
- *JSF – JavaServer Faces* – spolu so servletmi používaná ako súčasť MVC vzoru
- a iné [7]

3.1.1 Knižnice použité v projekte (Server, Java SE, Maven)

Na obrázku vidíme, s akými knižnicami sme pracovali v tejto práci:

Obrázok 4 Knižnice projektu

Okrem connectora, ktorý spája databázu a náš projekt, ktorého využitie bližšie opíšeme v časti databáza, sme v našej práci využili **Java knižnicu verziu 1.8** (JRE Sytem Library JavaSE-1.8). Ide o v súčasnosti nanovšiu verziu (v júli 2017 má výjsť dlhoočakávaná JavaSE-1.9). V našej práci sme pracovali ešte bez využitia lámbd, ktoré sú špecifické pre verziu Javy 8 ale pracovali sme s Javou 7, ktorá je na rozsah takejto práce efektívna a postačujúca.

Základný princíp využívania Java EE v samotnej podstate stojí na využívaní servra, na ktorom dokážeme spustiť našu aplikáciu zobrazenú v prehliadači. Tým dokáže obslúžiť veľa užívateľov naraz a slúži tak aj pre potreby veľkých firiem. Ako server sme si zvolili verziu 8 od Apache Tomcat.

Java Apache Tomcat, nazývaný Tomcat Server je open-source Java Servlet Container, ktorý implementuje viaceré Java EE špecifikácie vrátane Java Servlet, Java Server Pages a iné a poskytuje http web server prostredie v ktorom beží Java kód.

Nástroj, ktorý nám všetky predchádzajúce frameworky, knižnice a aj náš vlastný kód dokáže spojiť v jeden funkčný celok sa nazýva Maven.

Maven je nástroj, ktorým dokážeme spustiť(build) aplikáciu napísanú v jave a xhtml (html) a pridá k tomuto kódu aj všetky potrebné závislosti na knižniciach tretích strán a teda frameworkou ako hibernate a JSF a z utilitných knižníc a vytvorí jeden súbor WAR, ktorý vieme spustiť na lokálnom servri v našom prípade Apache Tomcat a následne sa nám vo webovom prehliadači zobrazí na lokálnej adrese naša aplikácia.

[13]

V našej práci sme využili aj nasledovné technológie, ktorých využitie si postupne predstavíme a oboznámime sa s ich úlohou v našej práci: JPA, Spring, Spring Security, Hibernate, MySQL, JSF, Eclipse, a iné.

3.2 Web tier - prezentačná vrstva aplikácie

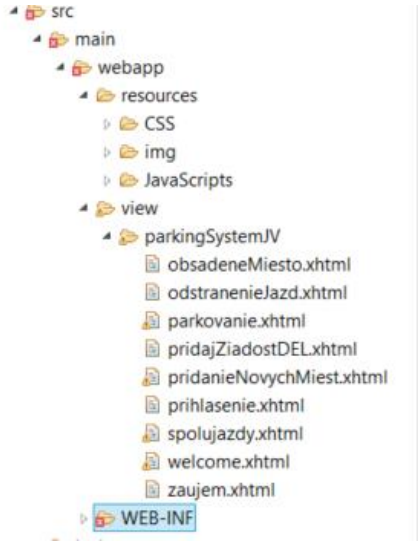
Pri navrhovaní prezentačnej vrstvy aplikácie, ktorá reprezentuje tú časť aplikácie, ktorá sa dá vidieť, sme sa v prvom rade zameriavali na jednoduchosť ovládania a prehľadnosť. Stránku sme navrhvovali tak, aby sa z hlavnej stránky užívateľ vedel navigovať do všetkých častí poskytujúcich funkcionalitu aplikácie. Finálnym zámerom je vytvoriť stránku zobrazenú v prehliadači, ktorá používa také jazyky ako **XHTML, CSS a javascript**, ktorým prehliadač rozumie.

V našej aplikácii použijeme XHTML formát, ktorý má korene v HTML ale je reformulovaný v striktné XML. To znamená, že XHTML dokument je XML s určitou schémou a má grafickú reprezentáciu v prehliadači. [6]

3.2.1 CSS a Javascript prostredníctvom Bootstrapu

Nakoľko sme si z časového hľadiska chceli ušetriť čas a zjednodušiť vývoj aplikácie využili sme framework Bootstrap, ktorý je voľne dostupný. Tento framework zahŕňa rôzne šablóny jazyka HTML, napríklad tlačítka, formuláre, obrázky a podobne, ktoré si jednoducho vieme dooimplementovať do našej aplikácie.

V aplikácii je CSS kód uložený v oddelených súboroch a tieto súbory sú importované na webovú stránku. Web dizajnér môže použiť skupinu odlišných CSS súborov a následne použiť alebo zmeniť stránku bez toho, aby sa obával o stratu a využitie CSS kódu. Na obrázku môžeme vidieť štruktúru priložených css, javascript zdrojov a obrázku a taktiež štruktúru xhtml súborov v našej aplikácii.



Obrázok 5 Opisovaná časť štruktúry práce

JavaScript je scriptovací jazyk používaný na strane klienta pri vývoji webových stránok a aplikácií. Aj napriek jeho menu, nemá žiadne spoločné znaky s Java programovacím jazykom. JavaScript je mocný nástroj na vytváranie dynamických webových aplikácií, písaním funcií ktoré interagujú priamo s DOM stránky.[6]

Pomocou bootstrapu môžeme tvoriť aj responzívny web design a vieme tak vytvárať webové stránky a webové aplikácie ktoré sa automaticky prispôsobia na akúkoľvek plochu obrazovky zariadenia, či už je to mobilný telefón alebo obrazovka počítača. Túto funkcionalitu využijeme pri zobrazení aplikácie v mobilných telefónoch používateľov, čím nám odpadá vývoj aplikácie pre určitý typ mobilného operačného systému.[6]

3.3 Business tier – business logika aplikácie

Pod web tier sa nachádza business logika aplikácie. Počas vývoja business vrstvy aplikácie sme využívali najmä **framework Spring a overenú prax** z tejto oblasti.

Spring framework uľahčuje vývoj JavaEE aplikácií. Dá sa o ňom uvažovať ako o frameworku frameworkou, pretože poskytuje podporu rozličným frameworkom ako Struts, Hibernate, Tapestry, EJB, JSF atď. Framework môže byť definovaný ako štruktúra, kde hľadáme riešenia na rozličné technické problémy. Spring poskytuje viacero modulov ako napríklad: IOC, AOP, DAO, Context, ORM, WEB MVC, a ďalšie. [14]

3.3.1 Application Context

V jednoduchosti by sa dalo povedať, že Application Context je kontext, ktorý načítava konfiguráciu (obyčajne z XML súboru) a po načítaní Spring začne manažovať beany:

- beany deklarované v balíkoch
- beany deklarované pomocou anotácií, ak sú nakonfigurované
- beany a ich automatické prepojenie a iné

Do súboru web.xml, ktorý sa považuje za konfiguračný súbor webových aplikácií Java EE, sme umiestnili argument, ktorý umožní, aby aplikácia pracovala so všetkými konfiguračnými súbormi začínajúcimi na applicationContext.

```
-       <context-param>
-             <param-name>contextConfigLocation</param-name>
-             <param-value>
-             /WEB-INF/applicationContext*.xml
-             </param-value>
-       </context-param>
```

Osvedčená prax (best practicies) pri vývoji, je vytvoriť viacero konfiguračných súborov, z ktorých každý sa zameriava na určitú oblasť. My sme preto vytvorili application context nazvaný: dataSource, hibernate, security, transaction a main. ApplicationContext-security rozoberieme neskôr v časti autentifikácia, kde sa dopodrobna zaoberáme prihlasovaním a autorizovaním užívateľov.

Ako už názov napovedá, konfigurácia dataSource upravuje prácu s datázovým zdrojom. My sme využili databázu MySQL, ktorú sme si pomenovali parking_system a prístupové práva ponechali root.

```
	<bean id="dataSource"

	class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="com.mysql.jdbc.Driver"
/>
		<property name="url"
value="jdbc:mysql://localhost:3306/parking_system" />
		<property name="username" value="root" />
		<property name="password" value="root" />

	</bean>
```

Ďalším súborom je applicationContext-hibernate.xml, kde sme deklarovali, ktoré entity má hibernate čítať ako oanotované triedy a spojiť ich s databázou.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<property name="dataSource">
<ref bean="dataSource" />
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
<prop key="hibernate.hbm2ddl.auto">update</prop>
<prop key="hibernate.show_sql">true</prop>
</props>
</property><property name="packagesToScan"
value="com.parkingSystemJV.entity"></property>
</bean>
```

Snažili sme sa o čo najväčšie ulahčenie „administratívnej práce“ s konfiguráciou a preto sme si zvolili **Spring Auto skenovanie komponentov.**

Obyčajne deklarujeme všetky beany a componenty v XML bean konfiguračnom súbore, tak, aby Spring kontajner mohol nájsť a registrovať beanu alebo komponentu. Podstatné ale je, že Spring je schopný automaticky naskenovať, detekovať a inštanciovať beany z preddefinovaného projektového balíku, takže nie je už viacej nutné beany deklarovať v XML súbore.

Typy anotácií určených pre automatické autoskenovanie. V Springu 2.5, sú 4 typy anotácií, ktoré sú automaticky naskenované:

- @Component – indikuje auto scan komponent.
- @Repository – indikuje DAO komponent v persistence layer.
- @Service – indikuje Service komponent v business layer.
- @Controller – indikuje kontroler komponent v presentation layer. [9]

Následne je nutné dať výraz: "context:component" do bean konfiguračného súboru, čo znamená, že umožníme auto skenovaciu vlastnosť Springu.

Používanie anotácií v našej aplikácii umožníme pomocou annotation-driven v Spring konfiguračnom XML súbore. [9]

V našej aplikácií sme použili vrstvy repozitáru a servisnú. Pre každú z týchto vrstiev bolo vytvorené aj rozhranie, v ktorej sú predpisy jednotlivých metód. Tieto metódy sa potom v triedach, ktoré toto rozhranie implementujú, musia implementovať.

Repozitár

Na vrstve repozitáru je trieda ParkovanieRepository, ktorý je uložený v package com.parkingSystemJV.repository. V repozitáry sú metódy na prístup do databázy. Tieto metódy sa volajú zo servisnej vrstvy.

Service

Na servisnej vrstve je trieda ParkovanieService, ktorá je uložená v package com.parkingSystemJV.service. V tejto vrstve je implementovaná business logika aplikácie. Z tejto vrstvy sa vracajú vypočítané údaje na frontend aplikácie.

3.4 Perzistenčná vrstva – vrstva uchovávania dát a jej namapovanie na aplikáciu

Aplikácie sú tvorené business logikou vysvetlenou vyššie, interakciou s inými systémami, užívateľským rozhraním a dátami. Veľká väčšina dát v našich aplikáciách sú uložené v databázach, sú získavané a analyzované. **Princípom *objektovo-relačného mapovania(ORM)* je spojiť svet databáz a objektov**. To obsahuje delegovanie prístupu relačných databáz externým nástrojom alebo frameworkom, ktoré dávajú relačným dátam

objektový vzhľad. Mapovacie nástroje sú napríklad Hibernate, TopLink a Java Data Objects, ale Java Persistence API(JPA) je preferovaná technológia a je súčasťou Java EE 7. [4]

V našej aplikácii sme použili na vytvorenie perzistnečnej vrstvy **JPA**, ORM framework **Hibernate** a databázu **MySQL**. Spring priamo podporuje aj ORM framework Hibernate, ktorého konfiguráciu sme si opísali v predchádzajúcej časti.

3.4.1 JPA

Ako sme už vysvetlovali, JPA je jedna zo základných špecifikácií Javy EE, ktorá sa zameriava na objektovo-relačné mapovanie. JPA je v našej aplikácii implementované prostredníctvom Hibernatu. Hibernate je významne využívaný nástroj medzi mnohými vývojovými tímami všade vo svete a umožňuje mapovať Javovské objekty na databázu a dotazovať sa.

Keď hovoríme o mapovaní objektov na relačnú databázu, uchovávanie objektov alebo využívanie query pre objekty, termín entita by mal byť použitý radšej ako termín objekt. Majú schopnosť byť namapované na databázu, môžu byť konkrétne alebo abstraktné a podporujú dedičnosť, vzťahy a iné. Výhodou je, že nám umožňuje dopytovať sa v objektovo-orientovanom spôsobe bez nutnosti použitia databázového dopytovania. Centrálnym miestom v API zodpovedným za riadenie entít je javax.persistence.EntityManager. Jeho rolou je manažovať entity, čítať alebo zapisovať do danej databázy a umožňuje jednoduché CRUD (create, delete, update, delete) operácie s entitami ako aj komplexné JPQL. [1]

3.4.2 Princípy používania entít

- Entitná trieda musí byť oanotovaná s @javax.persistence.Entity
- @javax.persistence.Id anotácia musí byť použitá na indikáciu primárneho kľúča
- Entita musí mať bezargumentový konštruktor, ktorý musí byť public alebo protected
- Entitná trieda nesmie byť final, tak ako jej metódy alebo inštančné premenné
- Entita musí implementovať Serializable interface, pri predaní hodnoty vzdialenému objektu [1]

Nasledujúca ukážka kódu z našej aplikácie a obrázok z databázy zobrazuje, ako vyzerá príklad entity Parkovisko, ktorú sme namapovali na tabuľku v databáze nazvanú parkovisko. Vidíme, že anotácie predstavujú logické vyjadrenie svojich reprezentácií.

```
package com.parkingSystemJV.entity;
/*importy*/
/**
 * Entita ako modelova trieda obsahujuca JPA anotacie
```

```
 */
@Entity
@Table(name = "parkovisko")
@Component("parkovisko")
public class Parkovisko implements java.io.Serializable {

		private static final long serialVersionUID = 1L;
		public int id;
		private String stat;
		private String mesto;
		private String meno;

		// bezparametricky konstruktor
		public Parkovisko() {
		}
		@Id
		@GeneratedValue(strategy = GenerationType.IDENTITY)
		@Column(name = "id", unique = true, nullable = false)
//getter a setter
		@Column(name = "stat", nullable = true, length = 30)
//getter a setter
		@Column(name = "mesto", nullable = true, length = 30)
//getter a setter
		@Column(name = "meno", nullable = true, length = 30)
//getter a setter
		private Set<Miesto> miesta = new HashSet<Miesto>(0);
		@OneToMany(fetch = FetchType.EAGER, mappedBy = "parkovisko")
//getter a setter

		private Set<Parkovanie> parkovania = new HashSet<Parkovanie>(0);
		@OneToMany(fetch = FetchType.EAGER, mappedBy = "parkovisko")
//getter a setter
		}
```

Daná entita parkovisko je namapovaná na tabuľku v databáze s rovnakým názvom:

id	stat	mesto	meno
1	SR	Bratislava	Douhouse
2	Nemecko	Mníchov	Berling - Sachsova 9
3	Chorvátsko	Zagreb	Suchsen

Obrázok 6 Tabuľka parkovisko v databáze

Všetky entity v našej aplikácii sme vytvorili obdobne a namapovali na tabuľky v databáze. Sú uložené v balíčku: com.parkingSystemJV.entity a celkovo sme ich vytvorili 5: Miesto, Parkovanie, Parkovisko, Spolujazda, OdstranJazdu.

4 Výsledky práce

V tejto časti popíšeme ako sme postupovali pri vytváraní webovej aplikácie z programatórskeho hľadiska. V úvodnej časti si uvedieme vývojové prostedie využívané aplikáciou a dátový model, ktorý sme použili ako údajovú základňu systému. V ďalšej časti použitím vizuálnej prezentácie podrobnejšie popíšeme komplexne funkcionalitu systému a v závere uvedieme aké ďalšie funkcionality by mala aplikácia v budúcnosti spĺňať.

Samotná implementácia predstavovala zložitú časť, na ktorej sme si mali možnosť vyskúšať mnohé technológie a mali sme možnosť pochopiť zložitým konceptom objektovo orientovaného jazyka. Návrh aplikácie sa mnoho krát menil počas vývoja, nakoľko sme postupne objavovali možnosti i úskalia funkčnosti systému. Súčasná verzia je orientovaná na splnenie požiadaviek používateľov v čo najväčšej miere a zároveň šetrenie nákladov princípom zdieľanej ekonomiky.

4.1 Vývojové prostredie

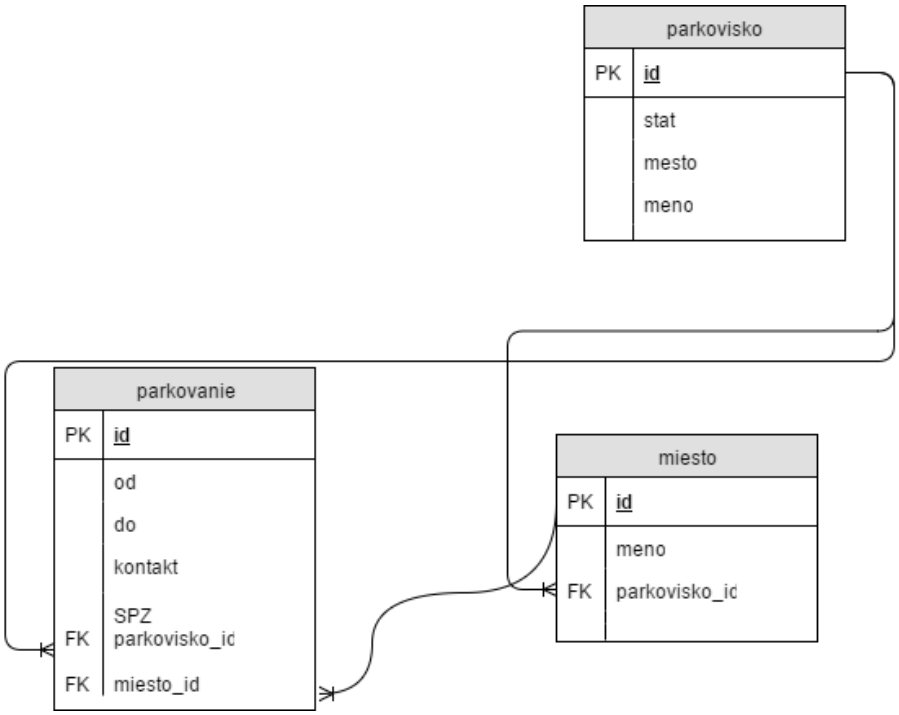
Pre implementáciu aplikácie sme si zvolili vývojové prostredie Eclipse, ktoré je používateľsky priateľské a obsahuje mnohé funkcionality. Toto prostredie je určené na programovanie v jazyku Java, ktorý je ideálny na programovanie webových aplikácií, pretože poskytuje mnohé možnosti a funkcionality na efektívnu tvorbu aplikácií. Medzi pre mňa najpraktickejšie spomeniem využitie pluginov v našej práci použitím nástroja eclipse marketplace. Pomocou neho sme si mohli do projektu stiahnuť mnohé nástroje medzi najdôležitejšími spomeniem Spring Tool Suite STS for Eclipse a Maven Integration for Eclispe, ktoré sme mohli následne využívať v projekte.

4.2 Databáza a dátový model

Naša databáza bola vytvorená s využitím platformy MySQL Workbench 6.3. Projekt založený v Eclipse a písaný v programovacom jazyku Java sme s databázou prepojili pomocou connectora 5.0.8.

Našu databázu sme si pomenovali **parking_system**. Databáza obsahuje **7** tabuliek: miesto, parkovanie, parkovisko, pouzivatelia, roly, spolujazda, odstranjazdu. Na obrázku

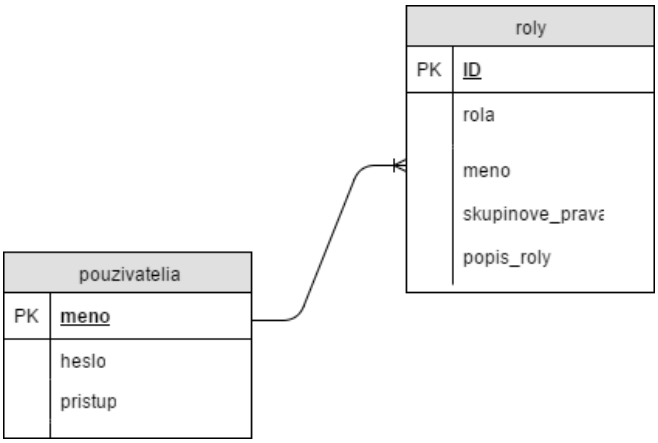
môžeme vidieť databázový model a vzťahy medzi entitami. Na zobrazenie schémy modelu sme využili softvér draw.io.



Obrázok 7 Databázová schéma parkovania

Pre zabezpečenie funkcionality parkovania sme v databáze vytvorili 3 tabuľky s názvom parkovisko, miesto a parkovanie. Najpodstatnejšou tabuľkou je tabuľka parkovanie, do ktorej užívateľ vkladá údaje o jeho záujme o parkovanie. Tabuľka obsahuje 2 cudzie kľúče, ktoré sa viažu na miesto a parkovisko. Potom ako administrátor prostredníctvom aplikácie vytvorí záznamy o existujúcich parkoviskách a ich parkovacích miestach, užívateľ má možnosť si ich vylistovať a vybrať. Tabuľka parkovanie obsahuje jeden primárny kľúč id a cudzie kľúče parkovisko_id a miesto_id. Keď si parkovanie zarezervuje, uloží sa mu automaticky štát, mesto a id parkoviska ako aj id parkovacieho miesta. Ďalej užívateľ vkladá údaje kontakte, jeho SPZ auta, a od kedy do kedy chce mať miesto rezervované. Tabuľka parkovisko je spojená s tabuľkou miesto takisto cudzím kľúčom, nakoľko pri pridávaním nových záznamov o mieste, údaje o parkovisku sa doplnia po zadaní názvu parkoviska.

Ďalšia schéma zobrazuje používatelov a ich roly. Pri autentifikácii sme využili framework Spring Security, ktorý prostredníctvom xml konfiguácie stanovuje aké hodnoty pre daný argument z tabuliek využije. Tento modul obsahuje 2 tabuľky používatelia s primárnym kľúčom meno a tabuľku roly s primárnym kľúčom ID. Cudzí kľúč sme vložili na meno nakoľko to sa vyskytuje logicky v oboch tabuľkách. Tabuľky medzi sebou majú vzťah 1:N. Jeden používateľ môže mať viac rolí a to byť napríklad používateľ i administrátorom.



Obrázok 8 Databázová schéma tabuliek používatelia a roly

Obrázok 9 Databázová schéma spolujazdy

Táto funkcionalita spolujazdy bola vytváraná ako dodatočná funkcia systému, a preto funguje ako samostatný modul pridávania a odstaňovania spolujazdy. Tabuľka spolujazda obsahuje primárny kľúč id a okrem toho do nej užívateľ uloží aj údaje o smere jazdy, cene, počet miest, kontakt na seba a dátume cesty. Pri záujme o jej predčasné odstránenenie sa odošle administrátorovi správa uložená v tabuľke obsahujúcej len údaj (a id), podľa ktorej jazdu overí a odstráni ju.

4.3 Implementácia používateľského rozhrania a funkcií

Táto časť obsahuje prehľad štruktúry aplikácie a stručný opis jednotlivých funkcií, ktoré aplikácia spĺňa.

4.3.1 Prihlasovanie užívateľov prostredníctvom Spring Security

Dôležitou súčasťou našej aplikácie je možnosť autentifikácie a autorizácie používateľov. Pre implementáciu takejto možnosti sme si zvolili využitie frameworku Spring Security, ktorý bol vyvíjaný s cieľom umožnenia bezpečnosti aplikácií.

Spring Security je mocný nástroj na autentifikáciu, umožňujúci kontrolu prístupu. **Spring Security je de-facto štandard pre aplikácie využívajúce Spring.** Spring Security je jeden z najrozšírenejších a najvyspelejších projektov Springu. Dnes je využívaný na zabezpečenie mnohých náročných prostredí vrátane **vladných agentúr, vojenských aplikácií a centrálnych bánk**. Je vydávaný pod licenciou Apache 2.0, takže sa môže používať v rozličných projektoch s absolútnou dôverou. [3]

Dve hlavné oblasti každej aplikačnej bezpečnosti sú autentifikácia a autorizácia. Sú to aj dve hlavné oblasti na ktorú sa Spring Security zameriava. **Autentifikácia** je proces overovania, či užívateľ je ten, za koho sa vydáva, že je.
Autorizácia je proces rozhodovania či používateľ má dovolené vykonávať určitú akciu v rámci aplikácie. [13]

Spring Security sme do nášho projektu implementovali nasledovne. Potom čo sme prostredníctvom pom.xml súboru vložili do projektu Spring security závislosti (dependencies), museli sme si tento framework správne nakonfigurovať. Do súboru web.xml sme museli najprv napamapovať springSecurity filtre, ktoré predstavujú filtre servletou, ktoré zaobchádzajú so všetkými HttpServletRequestami a HttpServletResponsami. Využili sme nasledovné xml argumenty:

```
	<filter>
		<filter-name>springSecurityFilterChain</filter-name>
		<filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
	</filter>

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

Následne sme si vytvorili konfiguráciu pre autentifikovanie a autorizovanie používateľov do aplikácie. Na niektoré stránky v aplikácii je nutné sa autentifikovať (rezervácia parkovania, pridanie spolujazdy, rozšírenie parkovacích možností), no niektoré je možné prehliadať i pre širšiu verejnosť. Tým sme sa snažili docieliť väčšiu informovanosť v oblastiach, kedy je to prospešné. Napríklad stránka, ktorá zobrazí prehľad o ponúkaných spolujazdách je voľne dostupná, nakoľko zníženie dopravných nákladov je priamo úmerné zvýšeniu potencionálnych záujemcov o jazdu. Rovnako kontaktné informácie či informácie o tom ako aplikácia funguje je vhodné ponechať pre záujemcov i bez autentifikačných práv. Na

časti programu zo súboru applicationContext-security.xml môžeme vidieť konfiguráciu autentifikácie užívateľov

```
<authentication-manager alias="authenticationManager">
<authentication-provider>
<user-service>
<user name="student" password="euba123" authorities="ROLE_USER" />
<user name="ucitel" password="fhi123" authorities="ROLE_USER, ROLE_ADMIN"
/>
</user-service>
</authentication-provider>    <authentication-provider>
<jdbc-user-service data-source-ref="dataSource"
users-by-username-query="SELECT * FROM PARKING_SYSTEM.POUZIVATELIA WHERE
MENO=?"
authorities-by-username-query="SELECT MENO, ROLA FROM PARKING_SYSTEM.ROLY
WHERE MENO=?"/>         </authentication-provider>

</authentication-manager>
```

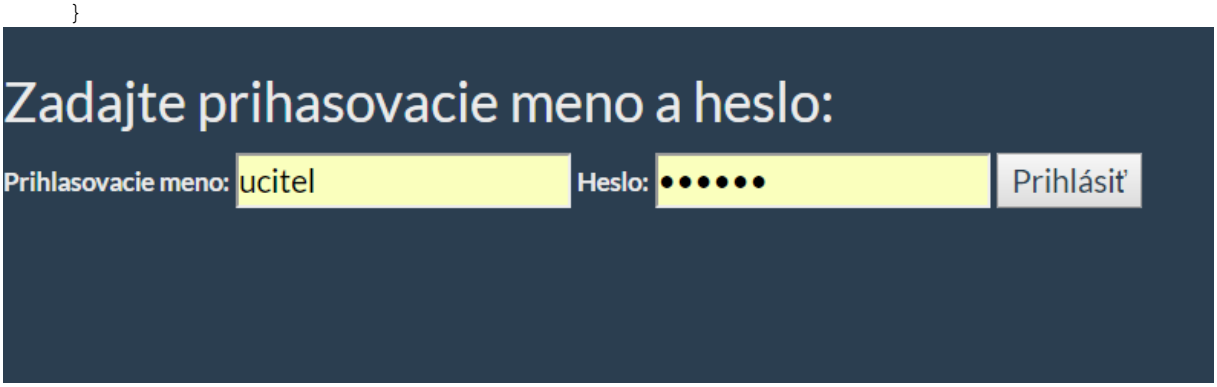
Autentifikácia v aplikácii je zabezpečená na dvoch úrovniach. Údaje o používateľoch potrebné na autorizovanie a autentifikovanie môžeme uviesť iba v konfiguračnom súbore spring-security ako vidíme v programe v časti user name="student atď. Pri prihlasovaní užívateľ uvedie meno student a heslo euba123 a prístup mu bude umožňený. Rola administrátora má okrem užívateľských možností, rozšírený prístup k funkciám. Toto riešenie je vhodné pre aplikácie, kde sa neočakáva vysoká variabilita používateľov a nie je nutné databázové riešenie.

Okrem tohto prístupu sme vypracovali aj možnosť ukladať údaje a roly používateľov do databázy, čo podstatne rozširuje množstvo unikátnych záznamov o používateľoch. Ako testovací údaj sme do databázy vložili markop a repovskyk, čo predstavuje priezvisko a iniciál mena. Jeden má rolu užívateľa a jeden administrátora. Týmto spôsobom môžeme rozšíriť užívateľskú databázu a ak sa jedná o firmu, do ktorej modul budeme integrovať, môžeme využiť už ich existujúcu databázu.

Samotné prihlasovanie je na strane backendu umožnené cez Prístupový Controller, ktorý zobrazí login prístupný pre každého užívateľa, a zobrazí xthml stránku prihlasenie, ktorú môžeme vidieť na obrázku nižšie.

```
@Controller
public class PrístupovýController {

      @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String zobrazLogin( ) {
        return "prihlasenie";
```

}



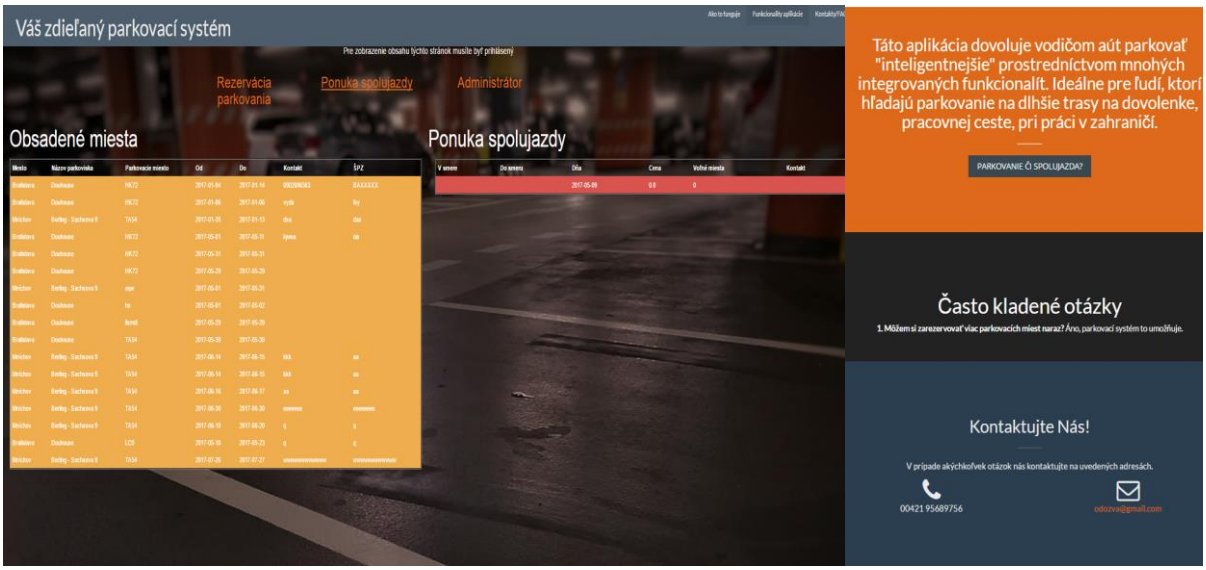
Obrázok 10 Login

Na tomto obrázku môžeme vidieť formu prihlasovania, ktorú sme si vytvorili. Logout je automaticky vytvorený prostredníctvom Spring Security.

Ako sme si už vysvetlili, vytvorili sme aplikáciu na ktorej sa zobrazuje aj obsah bez nutnosti autentifikácie, no na zobrazenie a prácu s niektorými funkcionalitami sa musíme prihlásiť, či už ako používateľ alebo administrátor. V ďalších častiach sa budeme podrobnejšie venovať aké funkcie aplikácie sú pre jednotlivých autorizovaných užívateľov zobrazované, a ako fungujú.

4.3.2 Prístup bez nutnosti autentifikácie

V tejto časti si opíšeme zobrazovanú stránku aplikácie a jej funkcionalitu, ktorá sa zobrazí všetkým užívateľom, ktorý si aplikáciu zobrazia. Po načítaní aplikácie sa užívateľ ocitne na hlavnej stránke, na ktorej sa nachádzajú všetky tlačidlá, ktoré nás navigujú k jednotlivým funkciám.



Obrázok 11 Hlavná stránka

V hornej časti stránky sa nachádza navigačná lišta, kde v ľavom rohu je názov aplikácie a napravo si vie užívateľ kliknutím na tlačidlo zoscrollovať kontaktné informácie, FAQ, a iné, ktoré je uvedené v pravej časti obrázku.

V prehľade obsadených miest sú zobrazené iba obsadené miesta a žiadna ďalšia funkcionalita nie je na tejto stránke umožnená. Zobrazenie funguje prostredníctvom tabuľky, do ktorej sa vylistujú hodnoty volaním metódy listParkovani() zo servisu parkovanie. Táto metóda:

```
public List<Parkovanie> listParkovani();
```

nám vráti List a teda zoznam všetkých parkovaní, uložených v databáze.

```
                    <tbody>
                          <ui:repeat
value="#{parkovanieService.listParkovani()}" var="park">
                                <tr class="warning">
                                      <td>#{park.odkedy}</td>
                                      <td>#{park.dokedy}</td>
                                      <td>#{park.kontakt}</td>
                                      <td>#{park.spz}</td>
                                      <td>#{park.parkovisko.mesto}</td>
                                      <td>#{park.parkovisko.meno}</td>
                                      <td>#{park.miesto.meno}</td>

                                </tr>
                          </ui:repeat>
                    </tbody>
                    </table>
```

Táto funkcia je vhodná nielen pre administrátorov stránky, ktorý môžu sledovať vyťaženosť obsadenia parkovacích miest, ale i pre nezaregistrovaného užívateľa, ktorý by rád našiel informácie o tom ako sú parkovacie miesta zaplnené, prípadne nájsť kontakt na osobu, ktorá miesto obsadila (napríklad v prípade nehody). Táto funkcia zobrazenia obsadenosti má okrem iného slúžiť ako odporúčanie administrátorovi aplikácie k tomu, aby v prípade plného stavu predložil zodpovednej osobe návrh na dojednanie zmluvy s externým parkoviskom, na ktorom môže firma alebo inštitúcia rozšíriť svoje parkovacie možnosti. Externé parkovisko by malo byť v čo najväčšej blízkosti k pôvodnému a následné doimplementovanie týchto miest by nepredstavovalo žiadnu komplikáciu. Dané parkovisko by sa následne len pridalo ako nový druh parkoviska, ktorý by bol k dispozícii používateľom.

Druhá výhoda zo zobrazovania obsadených miest by plynula pre zamestnancov, ktorý týmto spôsobom môžu prezerať kde a v ktorý deň daný užívateľ parkuje. Celkový prehľad miest je dôležitá funkcionalita správne vyvíjanej aplikácie.

Prehľad spolujázd nám slúži na to, aby si prihlásení i neprihlásení používatelia mohli

pozreť a vybrať spolujazdy do nimi vybranej destinácie. Častokrát sa stáva, že ľudia nemajú zabezpečený odvoz či už do práce, na služobnú cestu alebo na dovolenku a radi by využili komfort osobnej dopravy. Tu si môžu vybrať cestu, ktorá im vyhovuje podľa zverejnenej ceny a následne kontaktovať užívateľa na zverjnenom kontakte.

4.3.3 Funkcie pre zaregistrovaných používateľov

4.3.3.1 Rezervácia parkovania

Na hlavnej stránke si užívateľ tlačidlom *Rezervácia parkovania* po úspešnej autentifikácii môže zarezervovať parkovacie miesto na vybranom mieste. Táto funkcionalita bola z hľadiska vývoja najnáročnejšia a zároveň je aj kľúčovým element našej aplikácie, preto jej budeme venovať najviac pozornosti pri vysvetlovaní postupu implementácie.

Obrázok 12 Rezervácia parkovacieho miesta

Užívateľ nasjkôr vyplní od kedy do kedy má záujem mať rezervované miesto. Pri vyplnení údaju obsahujúceho dátum sa zobrazí kalendár s preddefinovaným formátom. Pre zobrazenie kalendára sme použili JQuery funkciu datepicker, viazanu na input html tagy:

```
<script>
	$(function() {
		$('input').datepicker();
	});
</script>
```

Následne sme museli vložiť converter, ktorý je priamo preddefinovaný v JSF knižnici na prekonvertovanie hodnôt dátumu vo vyžadovanej forme do databázy:

```
<tr>
	<td><label>Parkovanie Od:</label></td>
	<td>
		<h:inputText id="datum" value="#{parking.odkedy}">
			<f:convertDateTime pattern="MM/dd/yyyy" />
		</h:inputText>
	</td>
</tr>
```

Po vyplnení požadovaného obdobia, na ktoré chce mať osoba zarezervované parkovacie miesto, vyplní kontaktný údaj a ŠPZ auta. ŠPZ automobilu je vhodné uviesť, aby bolo v prípade komplikácií s preparkovaním auta ľahšia identifikácia.

Posledné údaje, ktoré musia byť uvedené sú výber parkoviska a parkovacieho miesta. Na tento účel sme prostredníctvom tlačidla zavolali metódu ulozRezervaciu() z triedy ParkovanieBean, ktoré uloží entitu parkovanie.

```
<td class="btn-group">
	<h:commandButton id="mybutton"
		action="#{parking.ulozRezervaciu()}" value="Pridaj" class="btn
		btn-primary btn-lg">
	</h:commandButton>
</td>
```

Táto metóda zavolá metódu s rovnakým názvom z triedy ParkovanieService, ktorá však má parametre parkoviskoId, miestoId, odkedy, dokedy, kontakt a spz. V tejto metóde sa vytvorí inštancia entity Parkovanie a priradia sa jednotlivé údaje do tejto inštancie.
Z parametrov miestoId a parkoviskoId sa pomocou metód vratMiestoPodlaId
a vratParkoviskoById získajú z databázy konkrétne objekty miesto a parkovisko s príslušnými id.

```
@Override

public Miesto vratMiestoPodlaId(int id) {

	Session session = this.sessionFactory.getCurrentSession();
	SQLQuery query = session.createSQLQuery("select * from miesto where
	id = :miestoId");
	query.addEntity(Miesto.class);
	query.setParameter("miestoId", id);
	List<Miesto> miesta = query.list();
	if (miesta.size() > 1) {
		return null;
	} else {
		return miesta.get(0);
	}
}
```

Potom sa zavolá metóda pridajANavigujParkovanie, ktorá rezerváciu uloží.

```
@Override
public String pridajANavigujParkovanie(Parkovanie p) {
	java.sql.Date datumOd = vratSqlDatumZDatumu(p.getOdkedy());
	java.sql.Date datumDo = vratSqlDatumZDatumu(p.getDokedy());
	if(parkovanieRepInterface.jeMiestoVolneMedziDatumami(p.getMiesto()
		.getId(), datumOd, datumDo)) {
		return this.parkovanieRepInterface.pridajANavigujParkovanie(p);
	} else {
		return "obsadeneMiesto?faces-redirect=true";
	}
}
```

Podoba volanej metódy nám najskôr prekonvertuje zadaný dátum do SQL podoby Date a následne zavolá metódu, ktorá overí, či je miesto, ktoré by užívateľ rád rezervoval voľné. Ak áno dané miesto sa rezervuje ak nie, tak sa používateľ presmeruje na stránku s chybovým hlásením.

Metóda overujúca, či miesto už nie je v danom období obsadené využíva sql select zobrazený v ukážke kódu nižšie.

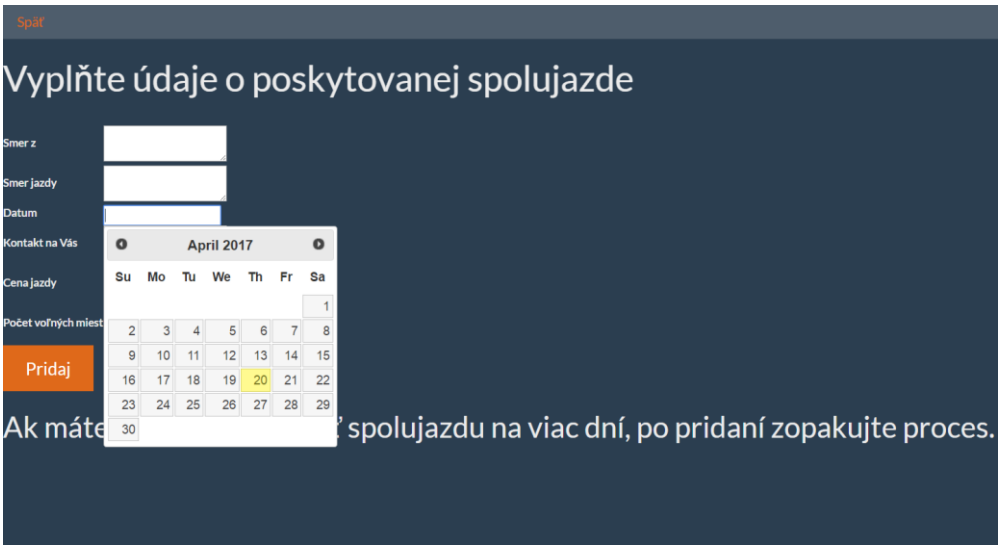
```
@Override
public boolean jeMiestoVolneMedziDatumami(int id, java.sql.Date odDatum,
java.sql.Date doDatum) {
	Session session = this.sessionFactory.getCurrentSession();
	SQLQuery query = session.createSQLQuery(
		"select * from parking_system.parkovanie where miesto_id = :id
		and (od between :odDatum and :doDatum or do between :odDatum
		and :doDatum)");
	query.addEntity(Parkovanie.class);
	query.setParameter("id", id);
	query.setParameter("odDatum", odDatum);
	query.setParameter("doDatum", doDatum);
	if (query.list().isEmpty()) {
		return true;
	}
	return false;
}
```

V pravej časti rozloženia stránky sa nachádza zoznam parkovísk, kde má užívateľ možnosť nahliadnuť na presné údaje o štáte a meste vybraného parkoviska.

Potom ako si užívateľ zarezervuje parkovacie miesto sa mu zobrazí stránka s otázkou, či má záujem o poskytnutie spolujazdy. Na väčšie vzdialenosti je takáto možnosť zaujímavá nielen pre vodiča, ktorý tým získa finančné alebo iné zdroje na krytie nákladov jazdy ale aj pre osobu spoujazdca, ktorá tým získa komfortnú prepravu do cielenej destinácie.

Na stránke sa zobrazia 2 tlačidlá, z ktorých submitovaním budeme presmerovaný na stránku poskytnutia spolujazdy a pri negatívnom potvrdení budeme presmerovaný na hlavnú stránku, čím je rezervácia ukončená.

4.3.3.2 Ponuka spolujazdy



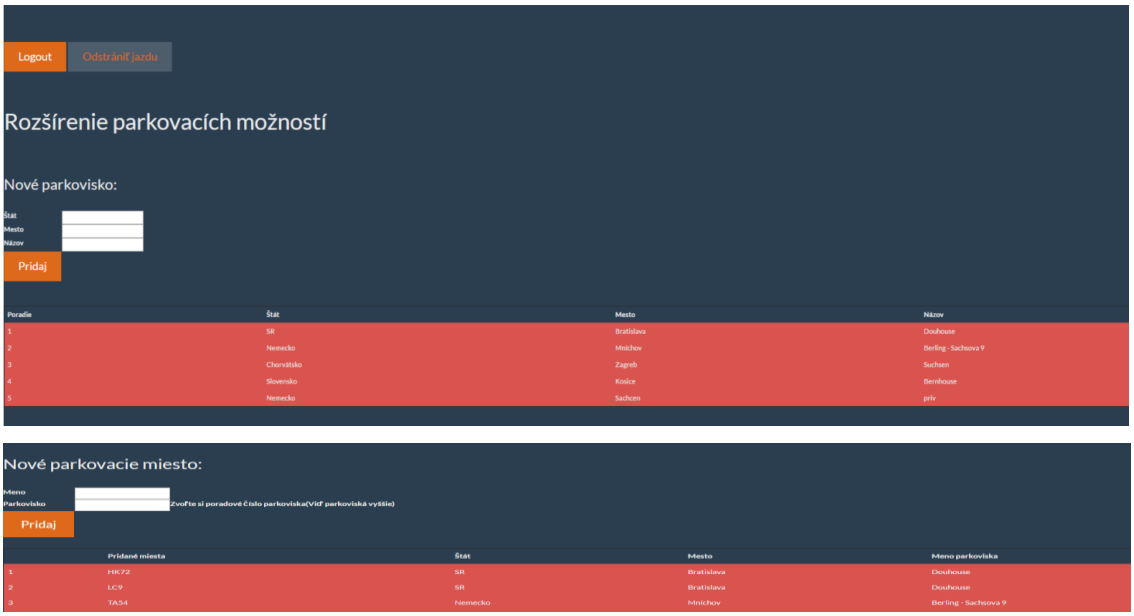
Obrázok 13 Poskytnutie spolujazdy

Registrovaný používatelia majú možnosť okrem rezervovania parkovania aj poskytnúť spolujazdu a tým si ušetriť prípadné náklady na dopravu. Takisto sme využili datepicker funkciu, ktorá zobrazí kaledár s preddefinovanými hodnotami vkladania. Okrem toho používateľ vloží údaje o smere jazdy, kontakt na ktorom ho môžu zastihnúť pri rezervácii, cenu a počet voľných miest.

V prípade, že chce používateľ predčasne odstrániť spolujazdu, má možnosť priamo v aplikácii odoslať žiadosť administrátorovi, ktorý request overí a spolujazdu vymaže. Táto funkcia slúži najmä z toho hľadiska, ak sa používateľovi zaplnia všetky miesta v automobile, nemá záujem byť naďalej bezcielne kontaktovaný. Administrátor následne priamo v aplikácii môže nadbytočnú informáciu o spolujazde odstrániť.

4.3.4 Administrátorské rozhranie

Najdôležitejšou súčasťou administrátorských právomocí je pridávať nové parkovacie miesta a nové parkoviská.



Obrázok 14 Pridávanie parkovísk a miest do aplikácie

Vzhľadom na to, že entity parkovisko a miesto sú previazané cudzím kľúčom v databáze, práca pri pridávaní nových údajov bola výrazne zjednodušená pri pridávaní miest prislúchajúcich k parkoviskám. Vzhľadom na to, že jedna firma alebo hotel môže vlastniť viacero parkovísk vo viacerých štátoch a mestách, je nutné pri pridávaní vyplniť všetky hodnoty parkoviska(okrem id, ktoré sa dopĺňa automaticky):

```
<td><h:inputText id="stat" value="#{parkovisko.stat}"></h:inputText>

<td><h:inputTextid="mesto" value="#{parkovisko.mesto}"></h:inputText>

<td><h:inputText id="meno" value="#{parkovisko.meno}"></h:inputText>
```

To zabezpečíme volaním metódy: `parkovanieService.pridajParkovisko(parkovisko)`

Nepredpokladáme ale, že by bolo často nutné dopĺňať nové parkoviská do aplikácie. Reálnejšie je potom rozširovať parkovacie možnosti o pridanie nových miest, napríklad rozšírením pozemku firmy. Pre tento prípad slúži inputText, do ktorého sa vkladá názov nového parkovacieho miesta:

```
<h:inputText id="miest" value="#{miesto.meno}"></h:inputText
```

a následne len z vyššie uvedeného zoznamu parkovísk zvoliť poradové číslo parkoviska. Štát, mesto aj názov parkoviska sa automaticky do aplikácie doplnia, práve prostredníctvom previazania entít cudzím kľúčom a vďaka jsf converteru. Na túto hodnotu (parkovisko, viď kód nižšie) sme museli naviazať nami vytvorený parkovisko converter (custom converter), ktorého

úlohou bolo skonvertovať zadaný String na integer a na základe neho vybrať objekt z databázy.

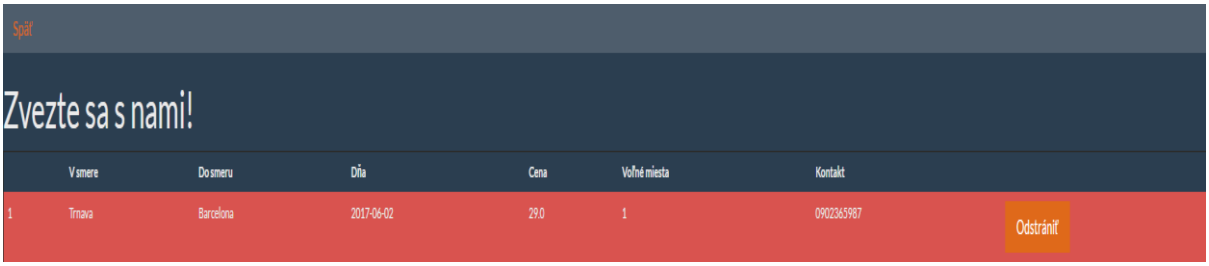
Xhtml kód s previazaným convertorom pre parkovisko a miesto:

```
<tr>
<td><label>Parkovisko</label></td>
<td><h:inputText id="parkovisko"
converter="#{parkoviskoConverter}"
value="#{miesto.parkovisko}"></h:inputText>
</td>
```

Java program pre konvertovanie uvedených hodnôt:

```
package com.parkingSystemJV.converter;

/*importy*/

@Component("parkoviskoConverter")
public class ParkoviskoConverter implements Converter {


	@Autowired
	private transient ParkovanieSerInterface parkovanieSerInterface;

	@Override
	public Object getAsObject(FacesContext context, UIComponent
component, String value) {
		try {
			int id = Integer.parseInt(value.replaceAll("[^\\d]",
""));
			return parkovanieSerInterface.vratParkoviskoById(id);
		} catch (NumberFormatException ignored) {
		}
		return null;
	}
	@Override
	public String getAsString(FacesContext context, UIComponent
component, Object value) {
		Integer id = (value instanceof Parkovisko) ? ((Parkovisko)
value).getId() : null;
	  return (id != null) ? String.valueOf(id) : null;   }
```

Po obdržaní informácie o záujme o predčasné vymazanie spolujazdy, ako sme si už vyššie rozoberali, v prípade, že sa auto naplí záujemcami a užívateľ logicky nemá záujem byť ďalej kontaktovaný administrátor môže priamo v aplikácii kliknúť na odstránenie.



Obrázok 15 Vymazanie údajov spolujazdy

Na priloženom kóde je zobrazená metóda, ktorá z databázy odstráni údaj o poskytovanej spolujazde.

```
public void odstranSpolujazdu(Spolujazda jazda) {
	Session session = this.sessionFactory.getCurrentSession();
	try {
	Spolujazda j = (Spolujazda) session.merge(jazda);
	sessionFactory.getCurrentSession().delete(j);
	}
	catch(Exception e){
		e.printStackTrace();
	}

}
```

4.4 Zhrnutie výsledku testovania

Výsledok našej práce, webovú aplikáciu sme testovali manuálnou formou, ručne, ale v budúcnosti by sme radi doplnili aplikáciu i o automatickú verziu testovania.

Charakter testovania bol vopred naplánovaný na overenie funkčnosti systému obohatený o overenie bezpečnosti aplikácie.

Pred zahájením testovania sme si naplnili databázu testovacími dátami a postupne overovali funkčnosť jednotlivých častí aplikácie. V tomto procese sme odhalili isté množstvo chýb, ktoré vyplívali z nesprávne nastavenej konfigurácie systému.

Hotovú verziu aplikácie vyvíjanej v našom bakalárskom projekte sme následne dali otestovať niekoľkým používateľom a sledovali sme ich reakcie na ovládanie a jednoduchosť funkcií. Následne sme zobrali do úvahy validné pripomienky a prispôsobili dizajn i vlastnosti aplikácie podľa ich pripomienok. V relatívne krátkom časovom rozsahu menej ako dvoch týždňov, sme boli schopný odstrániť všetky problematické oblasti.

4.5 Plány vylepšenia do budúcnosti

Pri vývoji aplikácie sme priebežne dostávali nápady na vylepšenie, ktoré by boli z hľadiska využitia zaujímavé a radi by sme ich v budúcnosti zaimplementovali do systému.

Z časového hľadiska a z hľadiska rozsahu bakalárskej práce sme vytvorili aktuálnu verziu aplikácie, ktorú by sme v budúcnosti radi rozšírili o niektoré nasledovné funkcionality, ktorých význam stručne charakterizujeme:

1. Vývoj tejto aplikácie pre mobilné zariadenia vo zvolenom mobilnom operačnom systéme – Android systéme. V súčasnosti je parkovací systém dostupný iba online prostredníctvom internetu.

2. Kamerový záznam áut, ktorý by mal v reálnom čase zaznamenávať výskyt automobilu na parkovisku, prípadne v časových intervaloch odosielať fotografie užívateľovi o stave auta.
3. GPS lokalizácia na mapách
4. Vzhľad parkoviska prispôsobiť reálnej podobe, akú napríklad používa Google maps.
5. Jazykové rozšírenie aplikácie v anglickom a nemeckom jazyku – vzhľadom na fakt, že aplikácia je primárne vytvorená pre užívateľov vo viacerých krajinách/ vzdialených oblastiach, je pravdepodobné využívanie niektorých zo svetových jazykov. My sme si predbežne zvolili anglický a nemecký jazyk, ktorý by sme radi implementovali v ďalšej fáze vývoja aplikácie.
6. Automatické testovanie aplikácie softvérom Selenium, ktoré by skúmalo bez využitia ľudského elementu prítomnosť chýb v kóde, priebežným testovaním funkcionality aplikácie.
7. Generovanie štatistík a grafov pre administrátorov aplikácie ako i užívateľov.
8. Mailový systém a notifikáciu medzi užívateľmi aplikácie, aj ako kontakt uvedený na stránke aplikácie.

ZÁVER

Cieľom našej bakalárskej práce bolo analyzovať, navrhnúť a implementovať integrovaný systém na rezerváciu parkovacích miest. Pre tento účel bola vyvinutá webová aplikácia v jazyku Java EE.

V prvých kapitolách práce sme sa zamerali na opis súčasnej problematiky danej oblasti, na ktorom bolo ilustrované aktuálnosť riešenia. Ďalej sme rozoberali aké požiadavky sú na systém kladené a aké softvérové riešenie sme si zvolili, pri návrhu architektúry daného systému. Pri vývoji aplikácie i pri štúdiu literatúry sme mali možnosť oboznámiť sa s mnohými konceptami zo súčasnej praxe a výrazne rozšíriť svoje poznatky v danej oblasti.

V druhej časti už konkrétne popisujeme funkcionality aplikácie, ktoré sú zobrazené na obrázkoch i priložených kódoch. Parkovací systém v tomto prípade princípom zdieľanej ekonomiky integruje viacero činností, ktoré uľahčujú život a fungovanie ľudí.

Nami vyvinutá aplikácia ešte vyžaduje niektoré zlepšenia, ktoré sme načrtli v podkapitole plány na zlepšenia v budúcnosti, ale pevne veríme, že už teraz má potenciál pre využívanie v rôznych oblastiach.

Zoznam použitej literatúry

[1] PILGRIM, P. 2013. *Java EE 7 Developer Handbook.* Birmingham: 2013. 634s. ISBN 9781849687942

[2] GONCALVES, A. 2013 *Beginning Java EE 7.* New York: 2013. 608 s. ISBN 978-1-4302-4627-5

[3] Spring Security. In: Spring [online]. [cit. 23.4.2017]. Dostupné na internete: < http://projects.spring.io/spring-security/>

[4] The Java EE 6 Tutorial. In: Java - docs.oracle [online]. 2013 [cit. 8.9.2016]. Dostupné na internete: < http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html/>

[5]LDAP Authentication. In: docs.oracle [online]. [cit. 20.2.2017]. Dostupné na internete: < http://docs.oracle.com/javase/jndi/tutorial/ldap/security/ldap.html/>

[6] What is Bootstrap? In: w3schools [online]. [cit. 14.4.2017]. Dostupné na internete: <http://www.w3schools.com/bootstrap/bootstrap_get_started.asp/>

[7] ŠPIRENG, P. JavaEE – sprievodca úplných začiatočníkov. In: robímeit [online]. 04.01.2014. [cit. 5.3.2017]. Dostupné na internete: < http://www.spireng.sk/content/java-ee-sprievodca-uplnych-zaciatocnikov/>

[8] MySQL 5.7 Reference Manual. In:MySQL [online]. [cit. 14.4.2017]. Dostupné na internete: <http://dev.mysql.com/doc/refman/5.1/en/>

[9] MKYONG. Spring Tutorial. In: Mkyong [online]. April 2010 [cit. 1.2.2017]. Dostupné na internete:<http://www.mkyong.com/tutorials/spring-tutorials/>

[10] Beans, BeanFactory and the ApplicationContext [online]. [cit. 23.4.2017]. Dostupné na internete:<http://docs.spring.io/spring/docs/1.2.9/reference/beans.html/>

[11] AASHISH, D. Parking Today. In:Media[online]. Marec 2014 [cit. 20.4.2017]. Dostupné na internete: <http://www.parkingtoday.com/articledetails.php?id=1600/>

[12] MVC Architecture. In: Tutorialsteacher [online]. [cit. 8.11.2016]. Dostupné na internete: <http://www.tutorialsteacher.com/mvc/mvc-architecture/>

[13] KNÁPEK, L.Webová aplikace v Javě od A do Z. In: Nullpointer.cz [online]. 2013[cit. 15.2.2017]. Dostupné na internete: < http://www.nullpointer.cz/webova-aplikace-v-jave-od-a-do-z-maven-plugin-pro-eclipse-a-pridani-struts2-do-projektu-2-dil/>

[14] Spring Tutorial. In: Javatpoint [online]. [cit. 20.2.2017]. Dostupné na internete: < https://www.javatpoint.com/spring-tutorial/>

[15] ODKLALAL, M. Nový fenomén doby – zdieľaná ekonomika. In: aktuality [online]. 19.08.2015 [cit. 23.11.2016]. Dostupné na internete:

https://www.aktuality.sk/clanok/302141/novy-fenomen-doby-zdielana-ekonomika-kazdy-moze-byt-podnikatelom/