Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Bakalářská práce

2017 **Michal Szabo**

Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Ověření vlastností webových technologií Java Enterprise Edition a Node.js

Bakalářská práce

Autor: Michal Szabo

Vedoucí práce: Ing. František Drdák, CSc.

České Budějovice 2017

Bibliografické údaje

Szabo M., 2017: Ověření vlastností webových technologií Java Enterprise Edition a Node.js [Verification of capabilities of internet technologies Java EE and Node.js. Bc. Thesis, in Czech.] – 41 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce se zabývá ověřením vlastností webových technologií Java EE a Node.js. V první části jsou vysvětleny rozdílné modely zpracování klientských požadavků. Dále je navržena funkcionalita aplikace, která je poté implementována v obou prostředích. Na závěr jsou stanovena obecná doporučení nabytá z praktické činnosti autora.

Klíčová slova

Java EE, Node.js, webová aplikace

Annotation

This bachelor thesis deals with verification of capabilities of internet technologies Java EE and Node.js. First part contains description of different models of client´s processed demands. If further contains desing of application´s functionality which is later implemented in both technologies. End of this thesis contains general recommendations that are based on author´s progress.

Key words

Java EE, Node.js, web application

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ..... podpis autora .....

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce panu Ing. Františku Drdákovi, CSc. za cenné rady, věcné připomínky a vstřícnost při konzultacích. Také děkuji rodině za podporu při studiu.

Obsah

1 Úvod

V dnešní době, kdy je internet všudypřítomný a každý se může odkudkoliv připojit k síti, jsou velkým trendem webové aplikace, které rychle nahrazují ty desktopové. Důvodů je celá řada. Ať už nutnost kolikrát zdlouhavé instalace programu, nebo dostupnost, kdy se musí uživatel nacházet u počítače, který daný systém obsahuje. Tyto problémy u webových aplikací odpadají a každý se k nim může připojit, stačí jen přístup k internetu a notebook nebo chytrý telefon.

Vývojáři si mohou vybírat z celé řady programovacích jazyků a technologií, s kterými budou při vývoji pracovat. Tato možnost je ovšem staví před nelehký úkol a občas není jasné, co bude pro právě řešený problém nejvhodnější použít. Při výběru záleží na různých aspektech. Některé je vhodné použít při vývoji složitého bankovního systému a jiné zase pro zpracování triviálního formuláře.

Tato bakalářská práce se zabývá dvěma odlišnými platformami pro vývoj právě webových aplikací. Tradiční platformou Java a relativně novým frameworkem Node.js. Hlavním úkolem je shrnutí poznatků nabytých při tvorbě aplikací, která jsou vyhotoveny v obou výše zmíněných prostředích.

V první řadě je jednoduše vysvětlen pojem webové aplikace, uvedeny příklady nejpoužívanějších technologií pro jejich tvorbu a představeny modely používané pro zpracování požadavků od klienta. Dále se práce zabývá návrhem funkčnosti požadované aplikace a popsáním technologií použitých při vývoji a popisem samotné implementace. Na závěr jsou shrnuty poznatky nabyté při praktické činnosti a stanovena obecná doporučení pro práci s výše zmíněnými technologiemi.

1.1 Cíle

Hlavním cílem této práce je stanovení obecných doporučení pro používání platforem Java Enterprise Edition (Java EE) a Node.js. Za tímto účelem byla navržena webová aplikace realizující interní systém pro rezervaci služebních aut. Tento program byl následně dvakrát implementován, a to jednou v technologii Node.js a podruhé v technologii Java.

Dílčí cíle práce jsou:

1. Navrhnout základní funkcionalitu webové aplikace typu rezervační systém pro realizaci a schvalování rezervačních požadavků pro alokaci služebních aut s využitím standardních metodik pro návrh softwaru.
2. Provést výběr vhodných implementačních technologií a nástrojů vývojové platformy JEE a realizovat vývojový proces aplikace včetně základního otestování.
3. Opakovat činnosti v bodu 2. pro platformu Node.js.
4. Stanovit obecnější doporučení ohledně zaměření, použitelnosti a efektivity obou platforem vyplývajících ze zkušeností nabytých při jejich použití v rámci bodů 2. a 3.

2 Webová aplikace

Webová aplikace je počítačový systém, ke kterému uživatel přistupuje pomocí webového prohlížeče. Je kombinací kódu na straně klienta, jehož úkolem je přijímat požadavky od uživatele a ty delegovat na server, kde se požadavek zpracuje a poté se odpověď odešle klientovi zpět. To umožňuje uživatelům komunikovat s firmou pomocí online formulářů, systémů pro správu obsahu, nákupních košíků a dalších. Aplikace také umožňují klientům vytvářet dokumenty, sdílet informace a spolupracovat na projektech bez ohledu na místo, kde se daná osoba nachází, nebo zařízení, které používá [1].

Výhody webové aplikace

- Podpora různých zařízení
- Přístup k nejnovější verzi
- Odpadá potřeba instalace
- Časově i cenově efektivní vývoj

Nevýhody webové aplikace

- Potřeba připojení k internetu
- Nekompatibilita některých prohlížečů

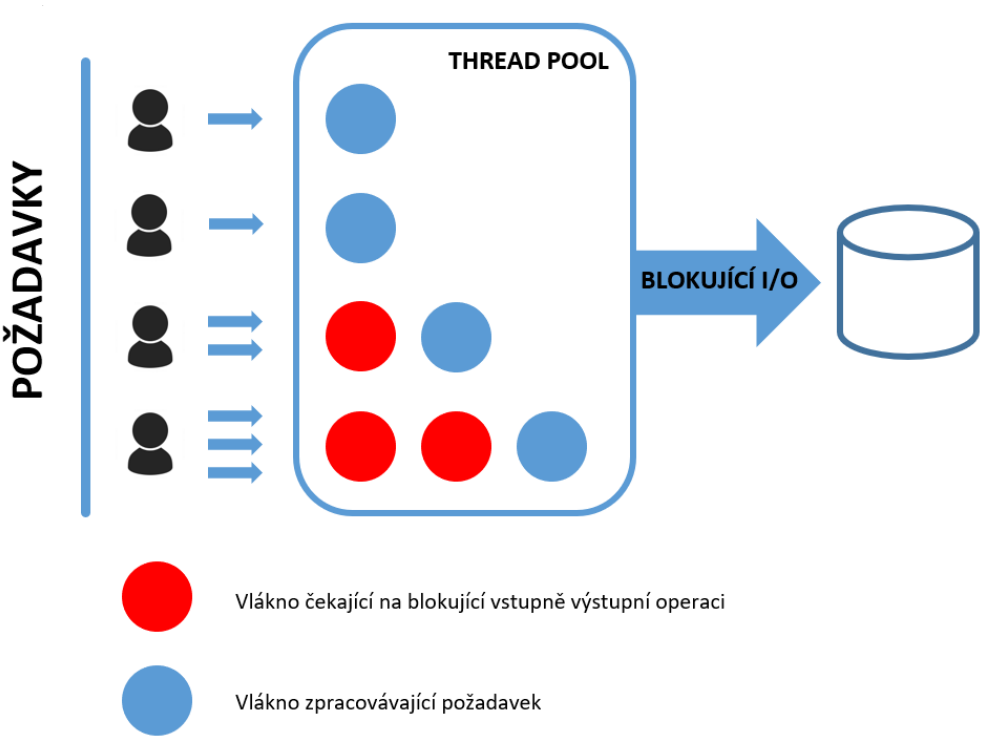
S rozšířením webu narůstal i počet jazyků umožňujících programování na straně serveru. Jazyky mají v podstatě stejné funkce, ale ty vykonávají rozdílným způsobem, rychlostí a technikami.

2.1 Modely zpracování vstupně výstupních operací

Všechny tradiční technologie pro vývoj webových aplikací mají stejný úkol – přijmout požadavek od klienta, provést potřebné operace na serveru a odeslat zpátky odpověď. Existuje ovšem více možností, jak této funkcionality dosáhnout. Zatímco Node.js běží v jednom, neblokujícím vlákně, a navíc využívá událostmi řízený model, tak Java EE využívá model standardní. To znamená, že každý požadavek obsluhuje právě jedno vlákno a využívá blokujícího zpracovávání vstupně výstupních operací. Oba zmíněné modely a způsoby zpracování kódu jsou popsány v následujících podkapitolách.

2.1.1 Více-vláknový blokující I/O model

Tento model zpracovává každý požadavek právě jedním vláknem. Toto vlákno často stráví nejvíce času čekáním na časově náročné operace, jako je například práce s databází. Problém nastává, pokud více uživatelů vyvolá požadavky, které provádí blokující vstupně výstupní operace. Poté se může stát, že budou všechna vlákna zaneprázdněna přípravou odpovědí a to způsobí, že bude klient čekat na odpověď delší dobu. Na následujícím obrázku (Obr. 1) je jednoduché schéma tohoto modelu.



Obrázek 1 Multi-trhead blocking I/O [2]

2.1.1.1 Synchronní programování

Jako příklad může posloužit čtení souborů v jazyce Java (Obr. 2).

```
try (FileInputStream is = new FileInputStream("file.txt")) {
    Session IOUtils;
    String content = IOUtils.toString(is);
}
```

Obrázek 2 Příklad synchronního programování v jazyce Java

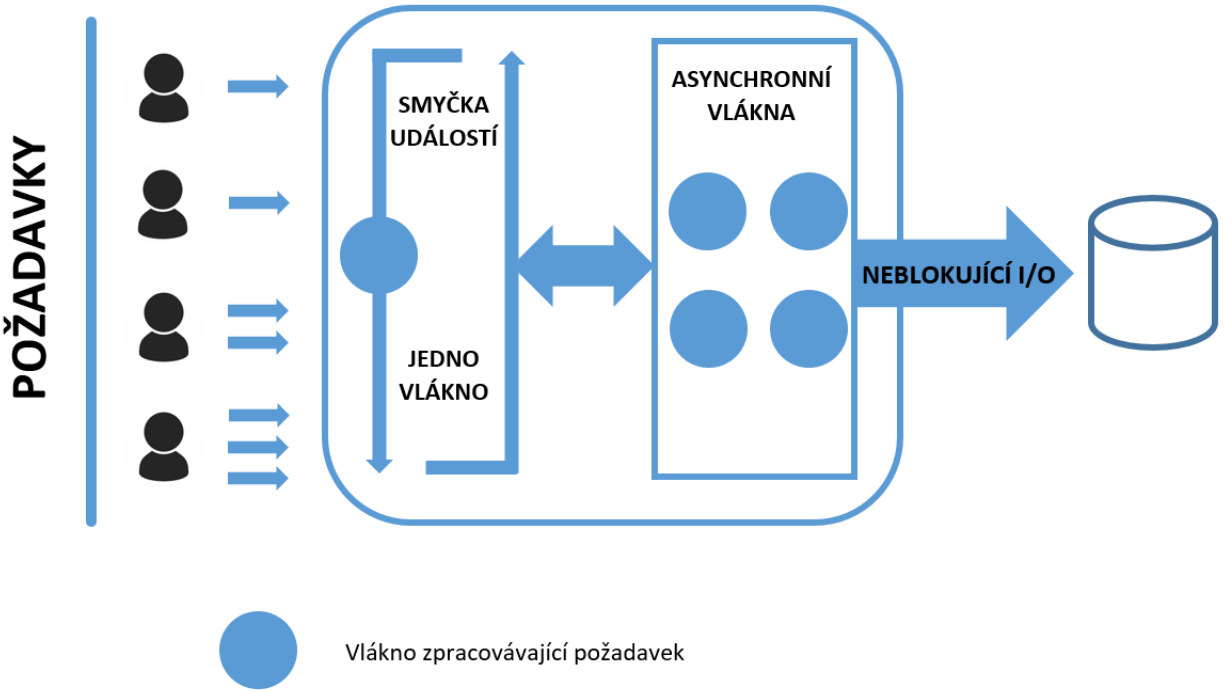
Tento kód zablokuje hlavní vlákno, dokud nebude soubor přečten, což znamená, že se před dokončením čtení nedostane vlákno k žádné jiné operaci. Pokud tedy nastane případ, že jedno vlákno musí najednou spravovat více náročných operací, tak může uživatel strávit poměrně

dlouhou dobu čekáním na odpověď od serveru. K vyřešení tohoto problému a lepšímu využití procesoru je pak nutné spravovat další vlákna.

2.1.2 Jedno-vláknový neblokující I/O model

Tento model funguje tak, že si před zpracováním prvního požadavku načte a inicializuje vše potřebné a jakmile je vše připraveno, tak naslouchá příchozím spojením. Jednotlivé požadavky se poté směřují na konkrétní controllery.

V synchronním programování se klasicky píše jeden příkaz na každý řádek. Jakmile program začne kód zpracovávat, tak to dělá postupně a není možné začít vykonávat další příkaz, dokud není ten předchozí dokončen, a tak blokuje ostatní. V Node.js je samozřejmě možné psát také tímto způsobem, ale pouze v případech, ve kterých nedochází ke zpracovávání uživatelových požadavků – inicializace frameworku, načítání konfigurace atp. Jakmile dojde na zpracování konkrétních HTTP požadavků, tak se ovšem musí vše programovat asynchronně, a to z důvodu práce právě v jednom vlákně, které by v případě dlouhotrvající operace nemohlo obsluhovat další požadavky a tím blokovalo celý server, dokud by se synchronní požadavek nezpracoval [3]. Následující obrázek (Obr. 3) tento model jednoduše prezentuje.



Obrázek 3 Single-thread non-blocking I/O [2]

Všechno to funguje tak, že uživatel odešle požadavek, který si přebere smyčka událostí (event loop). Ta pošle požadavek ke zpracování dalším vláknům na pozadí, která provádí náročnější úlohy. S těmito vlákny vývojář vůbec nepřijde do styku. Mezitím ale smyčka stále naslouchá příchozím požadavkům a směřuje je dále ke zpracování. Jakmile je připravena odpověď, tak se předá zpátky do smyčky událostí a ta ho dále směřuje danému klientovi.

2.1.2.1 Asynchronní programování

Oproti tradičnímu synchronnímu přístupu dovoluje ještě před dokončením pokračovat v práci dalším funkcím. Jednou z výhod neblokujících, asynchronních operací je, že můžou maximalizovat výkon jednoho procesoru a paměti.

K dosažení asynchronního programování slouží metody tzv. zpětného volání (callback functions). Callback funkce je funkce (funkce A), která je předaná jiné funkci (funkce B) jako

parametr a funkce A je volána uvnitř funkce B. Příklad callback funkce je uveden na obrázku níže (Obr. 4), na kterém funkce `readFile` představuje zmíněnou funkci B, která obsahuje dva parametry, kde právě druhým parametrem je funkce A.

```
var fs = require('fs');

fs.readFile('input.txt', function (err, data) {
    if(err) return console.error(err);
    console.log(data.toString());
});

console.log("Konec programu");
```

Obrázek 4 Callback funkce

Výstup pak vypadá následovně. Nejprve se do konzole vypíše „Konec programu“ a teprve potom obsah textového souboru. Nejdřív proběhne funkce `readFile`, která se pokusí číst ze souboru. Oproti synchronnímu programování se ovšem nečeká na dokončení čtení, ale kód pokračuje dál a čtení probíhá na pozadí. Dále se pokračuje kódem mimo funkci `readFile`. Po dokončení čtení ze souboru se pokračuje druhým parametrem (callback funkce) a pokud byl soubor úspěšně přečten, je proveden výpis jeho obsahu. Pokud ne, tak se do konzole vypíše chybová hláška.

3 Analytická část

3.1 Funkcionalita aplikace

Aplikace má simulovat jednoduchou správu pro rezervaci firemních automobilů. Na hlavní stránce se objeví pouze přihlašovací formulář, kde je vyžadována emailová adresa a heslo. Po úspěšném přihlášení se uživatel dostane na úvodní stránku a bude disponovat možnostmi dle své role, které jsou 3 – správce, schvalovatel, uživatel.

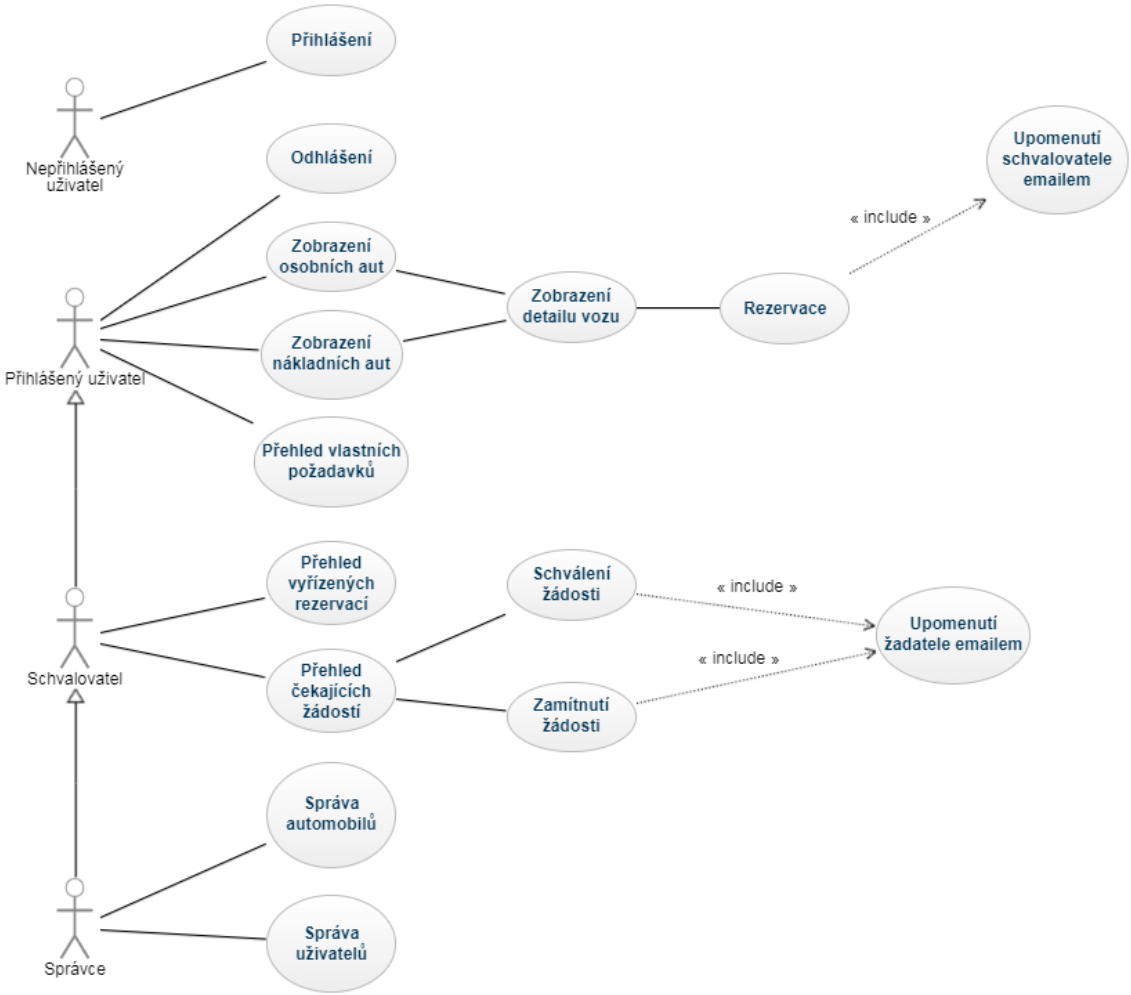
Správce má neomezené množství možností, které aplikace nabízí: zakládání, prohlížení, editaci a mazání všech uživatelů a automobilů, přehled a schvalování požadavků a samozřejmě disponuje také možností si auta rezervovat. Jelikož se jedná o interní systém, tak je zakládání nových uživatelů realizováno právě správcem a není možné se do ní jinak registrovat.

Roli schvalovatele si můžeme představit jako vedoucího oddělení, který má na starost žádosti od lidí jemu podřízených a má tedy mimo možností typické rezervace navíc na starost i schvalování jemu určených požadavků. O těch se dozví prostřednictvím emailové zprávy a nemusí proto aplikaci neustále sledovat. Po přijmutí, nebo zamítnutí rezervace přijde danému uživateli, o jehož požadavku bylo rozhodnuto, opět email s výsledkem.

Klasický uživatel má povolenou pouze rezervaci automobilů a má dostupný přehled svých požadavků.

Samotná rezervace probíhá tak, že si klient nejprve vybere ze dvou sekcí – osobních a nákladních automobilů. Po výběru se zobrazí všechna auta dané sekce, ze kterých si uživatel vybere preferovaný výsledek a rozklikne si detail, ve kterém se nachází podrobnější informace o voze. Na spodní straně detailu nalezneme vypsané termíny, na kdy je auto zarezervované a pod nimi kalendář, kde je možné vybrat požadovaný termín rezervace a odeslat požadavek. Systém ověří, zda na toto datum není auto rezervované, nebo zda se jakýmkoliv způsobem nekryjí data již schválených rezervací a pokud je vše v pořádku, odešle rezervaci zvolenému schvalovateli, který obdrží email s informací o novém nevyřízeném požadavku.

Na následujícím obrázku (Obr. 5) je zobrazen use-case diagram vytvořený v rámci návrhu aplikace.



Obrázek 5 Use-case diagram

V některých složitějších případech bylo zapotřebí funkcionalitu před samotnou implementací detailněji rozebrat. První use-case specifikace s názvem **Rezervace** (Obr. 6) popisuje, jaké kroky je zapotřebí provést před úspěšným odesláním požadavku.

Rezervace

Popis

Umožňuje rezervovat požadovaný automobil na zvolený termín.

Aktéři

- Uživatel
- Aplikace

Podmínky pro spuštění

Uživatel musí být přihlášený v aplikaci a musí se nacházet na stránce s detailem vozu.

Základní tok

1. Aplikace vygeneruje formulář se 4 vstupními poli – dva pro datum, jedno textové pole a jedno výběrové pole.
2. Uživatel vyplní povinná pole a odešle formulář.
3. Aplikace provede validaci dat od uživatele.
4. Aplikace prověří, zda se zvolená data nekříží s již potvrzenou žádostí.
5. Aplikace uloží požadavek, odešle email schvalovateli a přesměruje uživatele na hlavní stránku.

Alternativní tok 1

3.1 Pokud uživatel zadal neplatný vstup, nebo vstupy, systém ho na tuto skutečnost upozorní a nedovolí požadavek odeslat.

3.2 Uživatel opraví neplatný vstup, nebo vstupy a tok pokračuje na 2. bodu základního toku.

Alternativní tok 2

4.1 Pokud už existuje odsouhlasená rezervace na požadovaný vůz s kolidujícím datem, tak systém upozorní uživatele a vyzve ho, aby si vybral jiný termín.

4.2 Uživatel si vybere termín, který se nebude křížit s přijatou žádostí a tok pokračuje na 2. bodu základního toku.

Obrázek 6 Use-case specifikace – Rezervace

Specifikace s názvem **Tvorba uživatele** (Obr. 7) popisuje vytvoření nového účtu.

Tvorba uživatele

Popis

Vytvoří nový účet s přidělenými právy.

Aktéři

- Správce
- Aplikace

Podmínky pro spuštění

Správce musí být přihlášený v aplikaci a musí se nacházet na stránce určené pro zakládání nového uživatele.

Základní tok

1. Aplikace vygeneruje formulář s 5 vstupními poli – čtyři textová pole a jedno výběrové pole.
2. Správce vyplní povinná pole a odešle formulář.
3. Aplikace provede validaci dat.
4. Aplikace podle zadaného emailu ověří, zda se účet v systému již nevyskytuje.
5. Aplikace uloží nového uživatele a přesměruje správce na stránku s přehledem všech uživatelů.

Alternativní tok 1

3.1 Pokud správce zadal neplatný vstup, nebo vstupy, systém ho na tuto skutečnost upozorní a nedovolí požadavek odeslat.

3.2 Správce opraví neplatný vstup, nebo vstupy a tok pokračuje na 2. bodu základního toku.

Alternativní tok 2

4.1 Pokud se v systému nachází účet se stejnou emailovou adresou, tak systém na tuto skutečnost správce upozorní.

4.2 Správce zadá jiný email a tok pokračuje na 2. bodu základního toku.

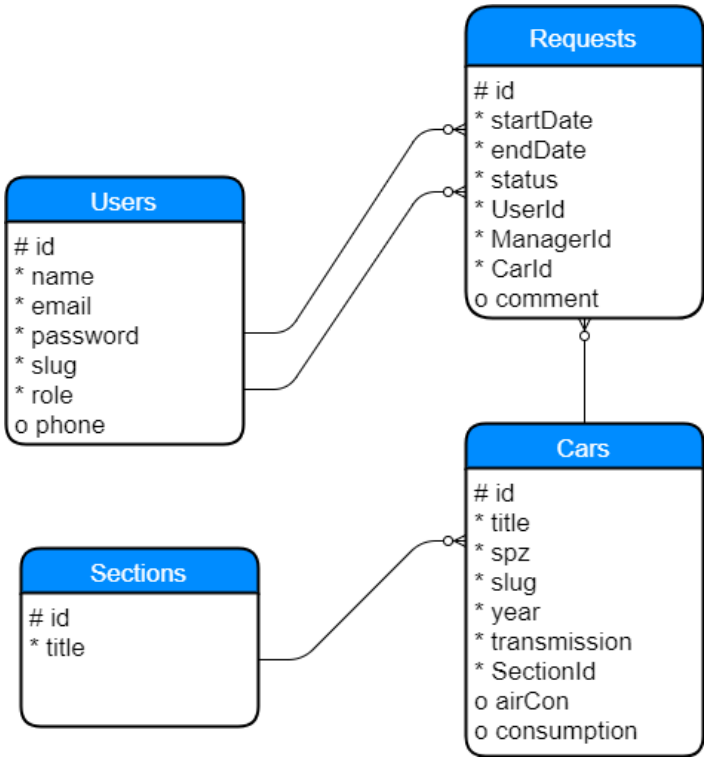
Obrázek 7 Use-case specifikace – Tvorba uživatele

3.2 Návrh databázové struktury

Po návrhu funkčnosti bylo zapotřebí navrhnout strukturu databáze (Obr. 8), jejíž data budou využívat obě aplikace.

Základním prvkem databázové struktury projektu je tabulka `Users`, která udržuje data o uživatelích – jméno, email, heslo, slug (část URL), roli a telefon. Tato tabulka má dvě 1:N (One

To Many) vazby na tabulku `Requests`, obsahující všechny informace o jednotlivých požadavcích. Konkrétně data značící od kdy do kdy chce uživatel vůz rezervovat, poté status, id uživatele, který o půjčku zažádal, id vybraného schvalovatele, id vozu, a nepovinný komentář. Entita s názvem `Sections` byla navržena pro budoucí snadnější rozšiřování o další možné sekce automobilů a zahrnuje pouze atribut název. Díky 1:N vazbě je propojena s tabulkou `Cars`, která obsahuje informace o vozech, a to název, spz, slug, rok výroby, typ převodovky, id sekce, klimatizaci a spotřebu. Všechny tabulky lze samozřejmě v budoucnu rozšířit o další atributy, ale pro potřeby popisované aplikace je tento návrh dostačující.



Obrázek 8 Entitně relační model

4 Webové technologie pro platformu Java

Na základě předešlé analýzy byl vyvozen závěr, že by standardní Java EE byla pro menší projekt, jakým se tato práce zabývá, příliš komplexní, a ne dost dobře použitelná. Po konzultaci s vedoucím práce proto došlo k rozhodnutí použít webový rámec v prostředí Java, který je popsán níže.

4.1 Java EE

Java EE obsahuje sadu specifikací, které rozšiřují Java Standard Edition (Java SE) o tzv. podnikové funkce, jako například distribuované výpočty, nebo webové služby. Java EE poskytuje API a běhové prostředí pro vývoj a provoz rozsáhlých, škálovatelných, spolehlivých a bezpečných síťových aplikací, které se zpravidla označují jako podnikové (enterprise applications), protože jsou navrženy tak, aby uspokojily poptávku po náročných řešeních pro bankovní aplikace, státní registry atp. Funkce, které poskytují těmto podnikovým aplikacím sílu, jako například bezpečnost a spolehlivost, často činí tyto systémy poměrně složité [4].

Pro běh Java EE aplikací je nutný aplikační server, který zastřešuje všechny knihovny, zajišťuje požadovanou funkcionalitu a mimo jiné také nabízí další klasické služby, jako administrátorskou konzoli nebo logování. Mezi nejznámější aplikační servery patří například GlassFish od Oracle, nebo JBoss of firmy Red Hat [5].

Z výše uvedeného popisu vyplývá, že standardní Java EE by byla pro zadaný projekt ne příliš dobře použitelná.

4.2 Volba sady technologií

4.2.1 Apache Tomcat

Apache Tomcat je open-source aplikační server poskytující webové prostředí pro spuštění programů v jazyce Java [6]. Tento kontejner byl použit především kvůli své jednoduchosti, transparentnosti a oproti svým komerčním konkurentům vyniká také poměrně malou náročností na výpočetní výkon.

4.2.2 Apache Maven

Apache Maven je nástroj navržený pro usnadnění práce a slouží ke správě a automatizaci sestavení aplikací napsaných v programovacím jazyce Java [7]. Díky jeho použití odpadá závislost na konkrétním IDE, protože v `pom.xml` (Project Object Model) souboru se nachází všechny informace potřebné k sestavení programu. Soubor je definován jednoduchou XML strukturou a popisuje projekt z pohledu závislostí na externích knihovnách. Pokud použijeme ovladač, který není k dispozici, ale je zároveň uveden jako externí knihovna právě v dokumentu `pom.xml`, tak Maven danou knihovnu stáhne a nainstaluje, ještě před přeložením zdrojového kódu. Vyhledávání poté probíhá v globálním Maven repozitáři, nebo v námi definovaných úložištích.

4.2.3 Apache Wicket

Apache Wicket je webový rámec založený na komponentech, sloužící k tvorbě webových aplikací v jazyce Java. Vyznačuje se vysokou abstrakcí nad protokolem HTTP a přináší událostmi řízený model, který je znám především z programování Swing aplikací [8].

Webová aplikace psaná pomocí Apache Wicket se pak skládá z HTML šablon pro prezentační vrstvu a Java kódu pro business logiku. Propojení šablony a Java kódu se poté realizuje v HTML souboru, kde se každé komponentě pomocí atributu `wicket:id` přidají potřebné vlastnosti a obsah.

4.2.4 Hibernate

Hibernate je framework umožňující objektově relační mapování (ORM). Jedná se o jednu z nejrozšířenějších implementací Java Persistence API (JPA). Jako takový může být snadno použit v jakémkoliv prostředí podporující JPA [9].

4.2.5 Souhrn technologií

- Apache Tomcat ve verzi 8.0.27.0
- Apache Maven ve verzi 3.3.9
- Apache Wicket ve verzi 7.9.0
- Hibernate ve verzi 5.2.6

5 Webové technologie pro platformu Node.js

Node.js je výkonné prostředí pro JavaScript, které umožňuje spouštět javascriptový kód na straně serveru a slouží k tvorbě rychlých, škálovatelných webových aplikací. Základem je javascriptový interpret V8 od Google [10]. Node.js používá událostmi řízený (event-driven) model a neblokující (asynchronní) vstupně-výstupní operace. Událostmi řízené programování je programovací paradigma, v němž je tok programu určen akcemi uživatele, jako například kliknutím myši, nebo stisknutím klávesy. Tato volba návrhu má za úkol minimalizaci režie procesoru a maximalizaci výkonu. Na rozdíl od tradičních serverů, jako například HTTP server od Apache, může Node.js obsluhovat více požadavků najednou, protože není omezen počtem dostupných vláken, ale pracuje pouze v jednom vlákně. Z tohoto důvodu musí být vždy při zpracování konkrétního HTTP požadavku použity asynchronní funkce, protože dlouhotrvající synchronní funkce by blokovaly celý server, dokud by nebyly zpracovány [11].

Jazyk JavaScript se dříve používal primárně na straně klienta, kde se skripty vkládají do HTML souborů a spouští se ve webovém prohlížeči uživatele. Node.js umožňuje použití jazyka JavaScript na straně serveru k vytvoření obsahu dynamické webové stránky předtím, než je zaslána do prohlížeče uživatele. V důsledku toho se Node.js stal jedním ze základních prvků paradigmatu JavaScript všude, který umožňuje sjednotit vývoj webových aplikací pouze kolem jednoho programovacího jazyka použitého jak na straně serveru, tak na straně klienta.

5.1 Volba sady technologií

5.1.1 NPM

NMP (Node Package Manager) je balíčkovací systém primárně sloužící ke sdílení kódu a jeho lehké aktualizaci. Ulehčuje práci dalším JavaScriptovým vývojářům, kteří mohou touto cestou jednoduše použít balíček, který zrovna potřebují, bez potřeby psát kód sami, ať už se jedná o malý, pěti řádkový skript nebo rozsáhlý testovací nástroj [12]. NPM není potřeba nijak složitě instalovat,

je totiž součástí základní instalace Node.js. Jeho pomocí můžeme velice jednoduše nainstalovat potřebné balíčky z příkazové řádky.

Pří každém startu aplikace si NPM zkontroluje, zda jsou dostupné všechny potřebné závislosti, které jsou uvedeny v souboru `package.json`. Tento soubor udržuje základní informace o aplikaci. Na následujícím obrázku (Obr. 9) je uveden příklad takového souboru, který byl vytvořen příkazem `npm init`. Příkaz spustí jednoduchý dotazník, kde je potřeba vyplnit pár informací o projektu, jako například název, verzi, klíčová slova, nebo jméno autora.

```
{
  "name": "ukazka",
  "version": "1.0.0",
  "description": "Ukazka npm init..",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Michal Szabo",
  "license": "ISC"
}
```

Obrázek 9 Ukázka package.json

5.1.2 Express.js

Express.js je dnes již považován jako standard při vývoji v Node.js. Jedná se o webový framework, který nabízí robustní sadu funkcí pro webové a mobilní aplikace. Poskytuje minimální rozhraní a umožňuje používat nástroje potřebné k vytváření aplikací. Je flexibilní, protože existuje celá řada balíčků, které lze na Express přímo napojit.

Mezi jeho hlavní výhody patří zejména nastavení směrování (route), které odkazuje na koncové body aplikace (URL) a říká, jak mají odpovídat na klientovy požadavky pomocí HTTP metod (Obr. 10). Dále umožňuje dynamicky vykreslovat HTML stránky na základě předávání argumentů daným šablonám [13]. Na následujícím obrázku je zobrazen jednoduchý příklad směrování pomocí Express.js.

```
var express = require('express');
var app = express();

// Po provedení GET požadavku na domovskou stránku se vypíše odpověď "Hello world"
app.get('/', function (req, res) {
    res.send('Hello World')
});
```

Obrázek 10 Ukázka Express.js

5.1.3 Sequelize

Sequelize je ORM framework pro Node.js. Je napsán v čistém JavaScriptu a podporuje spoustu různých databází, jako například PostgreSQL, MySQL, SQLite a MSSQL [14].

5.1.4 EJS

EJS (Embedded JavaScript) je nástroj pro vytváření šablon na straně klienta. Kombinuje data a šablonu pro vytvoření výsledného HTML souboru. V EJS souboru se před zobrazením uživateli, díky speciálním tagům, provede JavaScriptový kód a až poté se vygeneruje výsledek v čistém HTML [15].

5.1.5 Passport

Passport je autentizační nástroj, který může být použit v kterékoliv aplikaci založené na Express.js. Se stoupající oblibou sociálních sítí jako je Facebook, nebo Twitter se stala populární metoda přihlašování pomocí právě těchto služeb a díky Passport balíčku je možné tyto strategie využít [16].

5.1.6 Nodemailer

Nodemailer je modul umožňující snadné odesílání e-mailů. Před nastavením všeho potřebného je nutné založit emailovou schránku, která bude sloužit jako odesílatel [17].

5.1.7 Souhrn technologií

- NPM ve verzi 3.10.8
- Express.js ve verzi 4.14.0
- Sequelize ve verzi 3.30.1
- EJS ve verzi 2.5.5
- Passport ve verzi 0.3.2
- Nodemailer ve verzi 3.1.4

6 Implementační část

Pro uchovávání dat byla navržena relační databáze PostgreSQL, která je společná pro obě aplikace. Tento databázový systém byl vybrán kvůli podpoře v obou platformách a na základě pozitivní zkušenosti autora.

6.1 Java

Jako vývojové prostředí bylo použito Netbeans IDE ve verzi 8.2.

6.1.1 Adresářová struktura

RezervaceJava

Obsahuje jedinou třídu `WicketApplication` sloužící pro spuštění celé aplikace.

BasePage

Tento balíček obsahuje abstraktní třídu `BasePage` a stejnojmenný HTML soubor. Třída `BasePage` slouží jako kostra pro vzhled ostatních stránek, které z ní dědí. Obsahuje všechny menu komponenty, abstraktní metodu `getTitle` pro získání názvu stránky.

Controller

Obsahuje třídy, které jsou volány třídami z balíčků `view` a vykonávají potřebné operace nad daty.

- AuthenticationSession.java
- CarController.java
- RequestController.java
- SectionController.java
- UserController.java

Model

Obsahuje třídy jednotlivých entit. Jednotlivé třídy a atributy jsou označeny anotacemi z Java Persistence API (JPA). Dále poskytují metody pro získání a nastavení hodnot jednotlivým atributům.

- Car.java
- Request.java
- Section.java
- User.java

View

Třídy z tohoto balíčku slouží pro přímou komunikaci s uživatelem, obsluhují jeho požadavky, k čemuž využívají metody ze tříd obsažených v balíčku `Controller` a dále obsahují výčet webových komponent. Tyto komponenty s určeným obsahem se následně pomocí `wicket:id` tagu předají stejnojmennému HTML souboru, který je tak se třídou spjatý. Balíček `View` obsahuje pro větší přehlednost ještě další podbalíčky – `Cars`, `Requests` a `Users`. Výčet obsahuje pouze názvy bez přípon symbolizující oba stejnojmenné typy souboru – java a HTML.

- Homepage
- LoginPage
- (balíček) Cars
    - CreateCar
    - EditCar
    - PersonalCars
    - RequiredCar
    - Trucks
- (balíček) Requests
    - AllRequests
    - DoneRequests
    - MyRequests

    - RequestMenu
- (balíček) Users
    - AllUsers
    - EditUser
    - RegisterUser
    - RequiredUser
    - UsersMenu

Util

Tento balíček obsahuje dvě třídy – `EmailUtil` pro odesílání emailů pomocí JavaMail API a třídu `HibernateUtil`, která slouží pro připojení a práci s databází.

6.1.2 Vývoj aplikace

V první řadě byl vygenerován projekt. Na oficiálních stránkách Apache Wicket byl pomocí jednoduchého formuláře vygenerován Maven příkaz pro vytvoření základní kostry. Poté byly přidány potřebné závislosti do souboru `pom.xml`. Dále byla vytvořena třída `HibernateUtil` z balíčku `Util`, která zajišťuje připojení k databázi a obsahuje další nastavení nutná pro správnou funkci Hibernate. Následně byla naprogramována třída `EmailUtil` ze stejného balíčku. Ta využívá JavaMail API sloužící pro odesílání emailů. Dalším krokem bylo vytvoření tříd z balíčku `Model`, které obsahují objekty reprezentující auto, uživatele, sekci a požadavek. Poté byly naprogramovány třídy z balíčku `Controller`, jejichž metody obsluhují žádosti ze strany klienta. Na následujícím příkladu (Obr. 11) je metoda `rejectRequest` s parametrem `id` sloužící k zamítnutí požadavků a odeslání emailu danému klientovi. V těle metody se nejprve naváže spojení s databází. Podle `id` parametru se zjistí konkrétní požadavek a jeho status se změní na zamítnutý. Dále už se provede pomocí metody `update` aktualizace statusu a proběhne transakce. Nakonec se specifikují vlastnosti pro odeslání emailu uživateli – emailová adresa a tělo zprávy.

```java
/**
 * Metoda pro zamítnutí požadavku
 *
 * @param id id požadavku
 */
public void rejectRequest(int id) {

    Session sess = HibernateUtil.getSessionFactory().openSession();
    Transaction tx = null;

    try {
        tx = sess.beginTransaction();

        Request request = sess.load(Request.class, id);

        // Nastavení statusu na "rejected"
        request.setStatus(Request.Status.rejected);

        sess.update(request);

        tx.commit();

        // Tělo emailu + odeslání
        String body = "Dobrý den,<br>Vaše žádost byla bohužel"
                + " <b>zamítnuta</b>.<br><br>"
                + "Zkuste si prosím na stránce"
                + " <a href='http://localhost:8084/JavaRezervace'>"
                + "Rezervace aut</a> vybrat něco jiného.";
        mail.mailProperties(request.getUser().getEmail(), body);

    } catch (HibernateException e) {
        if (tx != null) {
            tx.rollback();
        }
        e.printStackTrace();
    } finally {
        sess.close();
    }

}
```

Obrázek 11 Metoda rejectRequest

Po naprogramování tříd z balíčku `Controller` byla vytvořena třída `BasePage` z balíčku `BasePage`, který navíc obsahuje stejnojmenný HTML soubor. Z této třídy dále dědí třídy z balíčků `View` a slouží jako kostra obsahující menu a základní vzhled stránek. Dále bylo přistoupeno k programování již zmíněných tříd z balíčků `View`. Každá třída má svůj stejnojmenný HTML

soubor. Následující příklad (Obr. 12, 13) ukazuje, jak dochází k předávání potřebných informací mezi Javou a HTML. Třída `LoginPage` obsahuje bezparametrický konstruktor, ve kterém jsou komponenty, které musí být definované i v HTML. Obsahuje jednoduchý formulář se dvěma vstupními poli – email, heslo. Komponenta pro formulář obsahuje překrytou metodu `onSubmit`, která se zavolá po pokusu o přihlášení a obsahuje podmínku, která v případě úspěšného přihlášení uživatele přesměruje na úvodní stránku, nebo v případě špatných údajů vyhodí hlášku, která o této skutečnosti klienta informuje.

```
public class LoginPage extends WebPage {

    private String username;
    private String password;

    public LoginPage() {
        setDefaultModel(new CompoundPropertyModel(this));

        FeedbackPanel feedback = new FeedbackPanel("feedback");
        add(feedback);

        Form loginForm = new Form("loginForm") {
            @Override
            protected void onSubmit() {
                // Ověření správnosti přihlašovacích údajů
                if (AuthenticatedWebSession.get().signIn(username, password)) {
                    setResponsePage(HomePage.class);
                } else {
                    info("Špatné uživatelské údaje!");
                }
            }
        };
        add(loginForm);

        TextField usernameTF = new TextField("username",
                new PropertyModel<String>(this, "username"));
        usernameTF.setRequired(true);
        loginForm.add(usernameTF);

        PasswordTextField passwordTF = new PasswordTextField("password",
                new PropertyModel<String>(this, "password"));
        loginForm.add(passwordTF);
    }

}
```

Obrázek 12 Třída LoginPage

```
<form wicket:id="loginForm">
    <input type="text" wicket:id="username" placeholder="E-mail"/>
    <input type="password" wicket:id="password" placeholder="Heslo"/>
    <input type="submit" value="Přihlášení"/>
</form>
```

Obrázek 13 Formulář z LoginPage.html

6.2 Node

Jako vývojové prostředí bylo použito IntelliJ IDEA ve verzi 2016.3.7.

6.2.1 Adresářová struktura

Kořenová složka

Obsahuje hlavní skript `app.js` sloužící k nastavení a spuštění celého serveru. Dále obsahuje soubor `package.json`, který udržuje seznam závislostí programu na různých balíčcích (node-modules).

config

Obsahuje soubor passport.js, který slouží k autentizaci uživatele pomocí balíčku Passport.

models

Obsahuje soubor `index.js` sloužící pro připojení k databázi a modely jednotlivých entit. Pro vytvoření modelů je použit ORM framework Sequelize. Model `user.js` navíc obsahuje funkci pro hashování hesla před vytvořením nového uživatele a funkci pro kontrolu hesla při přihlašování.

- car.js
- index.js
- request.js
- section.js
- user.js

node_modules

Obsahuje všechny balíčky definované v `package.json`.

public

Obsahuje složku stylesheets, ve které se nachází kaskádové styly pro výsledné HTML soubory.

routes

Obsahuje soubory s jednotlivými funkcemi obsluhující požadavky uživatele. Tyto funkce se následně volají ve skriptu `app.js`, kde s nimi může pomocí HTTP metod pracovat framework Express.

- cars.js
- home.js
- index.js
- requests.js
- users.js

views

Obsahuje EJS soubory, které kombinují data získaná ze serveru s předurčenou šablonou a vytváří tak požadované HTML soubory.

- Složka partials
    - footer.ejs
    - head.ejs
    - header.ejs
    - sidebar.ejs
- createCar.ejs
- doneRequests.ejs
- homepage.ejs
- myRequests.ejs
- personalCar.ejs
- register.ejs
- requestMenu.ejs
- requests.ejs

- requiredCar.ejs
- requiredUser.ejs
- trucks.ejs
- updateCar.ejs
- updateUser.ejs
- users.ejs

6.3 Vývoj aplikace

V první řadě byl vygenerován `package.json`, který obsahuje informace potřebné pro spuštění aplikace. Tento soubor byl vytvořen pomocí příkazu `npm init` a následným vyplněním jednoduchého formuláře. Poté byly staženy všechny balíčky příkazem `npm install jmeno_balicku --save`. Přepínač `--save` zajistí automatické přidání do závislostí (dependencies) právě v souboru `package.json`. Dále byl naprogramován hlavní spouštěcí skript `app.js`, kde proběhlo načtení všech balíčků (Obr. 14) a potřebné nastavení aplikace (Obr. 15).

```
// Načtení balíčků
const express = require('express');
const app = express();
const http = require('http');
const passport = require('passport');
const bodyParser = require('body-parser');
const session = require('express-session');
const cookieParser = require('cookie-parser');
const flash = require('connect-flash');
const expressValidator = require('express-validator');
```

Obrázek 14 Načtení balíčků v app.js

```
// Konfigurace aplikace

app.use(flash());

//Nastavení sessions a cookie parser
app.use(cookieParser());
app.use(session({
    secret: 'blablasecretismorebEtta',
    cookie: {expires: false},
    resave: false,
    saveUninitialized: false
}));
app.use(passport.initialize());
app.use(passport.session());

// Specifikace adresáře obsahujícího statické soubory - js, css, apod.
app.use("/public", express.static('public'));

//Nastavení EJS jako template enginu
app.set('view engine', 'ejs');

//Body parser pro parsování příchozích požadavků
app.use(bodyParser.urlencoded({extended: true}));
app.use(expressValidator());
```

Obrázek 15 Konfigurace aplikace v app.js

Poté byly vytvořeny soubory v adresáři `models`, tedy objekty reprezentující auto, sekci, uživatele, požadavek a soubor `index.js` sloužící pro připojení k databázi. Ukázka mapování pro entitu `Car` je zobrazena na následujícím obrázku (Obr. 16).

```
'use strict';
// Objekt reprezentující auto
module.exports = function (sequelize, DataTypes) {
    var Car = sequelize.define('Car', {
        title: DataTypes.STRING,
        spz: {
            type: DataTypes.STRING,
            unique: true
        },
        slug: {
            type: DataTypes.STRING,
            unique: true
        },
        year: DataTypes.INTEGER,
        transmission: DataTypes.STRING,
        consumption: DataTypes.STRING,
        airCon: DataTypes.BOOLEAN,
        SectionId: {
            type: DataTypes.INTEGER,
            references: 'Sections',
            referencesKey: 'id'
        }
    }, {
        classMethods: {
            associate: function (models) {
                Car.belongsTo(models.Section);
                Car.hasMany(models.Request);
            }
        }
    });
    return Car;
};
```

Obrázek 16 Ukázka mapování pro entitu auto

Dalším krokem bylo vytvoření adresáře `config` a v něm souboru `passport.js`, ve kterém jsou implementovány metody pro potřeby autentizace. Poté byl vytvořen adresář `routes`, do kterého byly umístěny soubory obsahující funkce pro obsluhu požadavků uživatele, pro práci s databází a nachází se zde i funkce obstarávající odesílání emailů. Tyto metody jsou dále volány v hlavním skriptu `app.js`, kde se s nimi pracuje prostřednictvím potřebných HTTP metod (Obr. 17).

```
app.post('/delete-car', cars.deleteCar);
app.get('/admin/edit-car/:slug', cars.editCar);
app.post('/save-edited-car', cars.saveEditedCar);
app.post('/send-reservation', cars.sendReservation);

app.get('/my-requests', home.IsAuthenticated, requests.myRequests);
```

Obrázek 17 Ukázka využití HTTP metod pomocí Express.js

Dále byl vytvořen adresář `views` obsahující EJS šablony, v nichž se na straně serveru vytvoří požadované HTML, které se pak odešle klientovi.

```
<div>
    <ul>
        <% if(cars){ %>
        <% cars.forEach(function(car){ %>
        <input type="hidden" name="carId" value="<%= car.id %>">
        <li><a href="/cars/personal-cars/<%= car.slug %>"><%= car.title %></a></li>
        <% if(role === "admin"){ %>
        <li><a href="/admin/edit-car/<%= car.slug %>">Editovat <%= car.title %></a></li>
        <form action="/delete-car" method="post">
            <input type="hidden" name="carId" value="<%= car.id %>">
            <li>
                <button type="submit" class="btn-link"
                        onclick="return confirm('Jste si jistý?')">Vymazat <%= car.title %>
                </button>
            </li>
        </form>
        <hr>
        <% } %>
        <% }) %>
        <% } %>
    </ul>
</div>
```

Obrázek 18 Ukázka EJS pro zobrazení seznamu aut

Jako poslední pak byly vytvořeny CSS soubory pro úpravu vzhledu šablon.

7 Doporučení ohledně zaměření

7.1 Obecná doporučení

7.1.1 Java EE

Java EE je určena pro programátory, kteří mají dobré zkušenosti s tvorbou desktopových aplikací pomocí programovacího jazyka Java. Je vhodné, ale ne nezbytné, mít přehled o základních webových funkcích, do kterých v případě neznalosti dostatečně zasvětí kvalitně propracovaná dokumentace.

V praxi a zejména při práci v týmech je určitě vhodné využít nástroj pro automatizaci sestavení projektu, který dokáže poskytnout přehledné informace o projektu, nebo možnost jednoduchého přidávání nových funkcí. Pro tyto potřeby může posloužit Maven od společnosti Apache, nebo třeba Gradle.

Java EE slouží zejména pro tvorbu velkých systémů. Proto bych při řešení menších projektů doporučoval využití webového frameworku pro platformu Java, který nebude pro řešení daného problému tak těžkopádný. Tyto frameworky obsahují mnoho užitečných a otestovaných komponent, které se při vývoji běžně využívají a není tedy nutné znovu vytvářet něco, co již existuje. Také pomáhají s uspořádáním kódu do snadno udržitelného modelu, jako je třeba MVC.

7.1.2 Node.js

Node.js bych doporučil vývojářům, kteří již vyzkoušeli tvorbu webových aplikací v jiných jazycích a mají dobrou znalost JavaScriptu. Oficiální dokumentace je totiž zaměřena na zkušenější programátory a nezabývá se základními pojmy, jako jsou HTTP metody, cookies atp. Oproti běžným technologiím, využívaných pro psaní webových aplikací je při práci s Node.js ovšem nejdůležitější a zároveň nejtěžší, pochopit způsob a zpracování kódu pomocí asynchronních funkcí.

Programátoři, kteří přecházejí z vývoje uživatelského rozhraní na straně klienta, kde jazyk JavaScript hojně využívali, k programování na straně serveru, budou mít práci značně ulehčenou. Odpadá totiž potřeba učit se nový programovací jazyk a proniknutí do Node.js je díky tomu jednodušší.

U Node.js je velmi výhodné využívat rozsáhlého a kvalitně zpracovaného správce balíčků, kterým NPM beze sporu je. Místo psaní kolikrát složitého kódu je možné si vyhledat potřebný modul, jednoduše nainstalovat a můžeme využívat jeho funkcionalitu, která je dobře propracovaná a otestovaná řadou vývojářů. Při rozhodování mezi jednotlivými balíčky, které nabízejí podobnou funkcionalitu, si můžeme při výběru dopomoci prozkoumáním hodnocení a počtem stažení.

Dále je vhodné využít webový framework pro usnadnění tvorby a snadnou organizaci kódu. Jako standard pří vývoji Node.js aplikací se považuje Express.js, který je využit i v této práci.

7.2 Srovnání

Následující tabulka obsahuje srovnání vlastností platforem Java EE a Node.js.

Vlastnost	Java EE	Node.js
Vývojář	Oracle	Ryan Dahl
Datum zveřejnění	1999	2009
Jazyk	Java	JavaScript
I/O zpracování	Synchronní	Asynchronní
Vícevláknové programování	Ano	Ne
Správce knihoven	Maven, Gradle, atd	NPM

Tabulka 1 Srovnání vlastností Java EE a Node.js

Další tabulka obsahuje ohodnocená kritéria použitých frameworků v obou platformách, založena na zkušenostech nabytých při vývoji aplikací.

Kritérium	**Apache Wicket**	**Express.js**
Složitost kódu	Jednoduchý, lehce srozumitelný kód.	Složitější kód, zejména díky callback funkcím.
Srozumitelnost kódu	Srozumitelný a přehledný.	Místy může být nepřehledný.
Efektivita kódu	Méně efektivní, pokud se jedná o náročné časové operace.	Velmi efektivní díky asynchronnímu přístupu.
Náročnost implementace	Implementace není složitá, ale jednotlivé části, které je potřeba předem nastavit zaberou poměrně dost času.	Pro zkušeného programátora jednoduchá a rychlá.

Tabulka 2 Srovnání použitých webových frameworků

7.3 Shrnutí

Pro nepříliš rozsáhlou aplikaci, která byla vypracována v této práci je příhodnější využít Node.js. Java EE se přeci jen zabývá mnohem komplexnějšími systémy a pro vývoj aplikace tohoto typu je příliš těžkopádná.

8 Závěr

Práce v teoretické části jednoduše shrnuje, co je to pojem webová aplikace. Dále vysvětluje dva modely pro práci se vstupně výstupními operacemi, kde jeden využívá platforma Java a druhý Node.js. Tato část zahrnuje i popis a ukázky rozdílného přístupu programování pro oba modely.

Následuje analytická část, ve které je popsána funkcionalita požadované aplikace. Funkcionalita je formulována nejprve slovně a poté pomocí use-case diagramu, který znázorňuje kompletní možnosti systému. Analýza obsahuje také dvě use-case specifikace popisující podrobný postup řešení složitějších funkcí.

V praktické části jsou následně vybrány vhodné technologie použité při vývoji. Poté již práce popisuje tvorbu aplikací v obou prostředích. Nejprve je popsána adresářová struktura projektů a výčet všech obsažených souborů s krátkým popisem funkcionality. Dále je popsáno, jak bylo postupováno při samotné implementaci obou aplikací s ukázkami naprogramovaného kódu. Obě aplikace byly průběžně testovány formou ručního testování vývojářem.

Závěrečná část pak obsahuje základní porovnání obou platforem a obsahuje obecná doporučení, která jsou vyvozena jak ze zkušeností nabytých při praktické činnosti, tak z nabytých vědomostí získaných při teoretickém prozkoumávání technologií. Tyto doporučení shrnují, komu by autor dané platformy doporučil a jaké znalosti s vývojem webových aplikací je vhodné mít ještě před prací s nimi a následně je provedeno porovnání na základě složitosti, efektivity kódu a náročnosti implementace.

Citovaná literatura

[1] NDEGWA, Amos. *What is a Web Application?* [online]. MaxCDN One, 2016 [cit. 2017-12-04]. Dostupné z: https://www.maxcdn.com/one/visual-glossary/web-application/

[2] MARDAN, Azat. *Webapplog [programming weblog]: You Don't Know Node: Quick Intro to Core Features* [online]. 2016, April 4, 2016 [cit. 2017-12-13]. Dostupné z: https://webapplog.com/you-dont-know-node/

[3] MROZEK, Jakub. *JavaScript na serveru: Architektura a první Hello World: Asynchronní zpracování* [online]. b.r. [cit. 2017-12-13]. Dostupné z: https://www.zdrojak.cz/clanky/javascript-na-serveru-architektura-a-prvni-hello-world/

[4] *Overview of Enterprise Applications* [online]. Oracle, 2012 [cit. 2017-12-03]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

[5] *Your First Cup: An Introduction to the Java EE Platform* [online]. Oracle, 2012 [cit. 2017-12-03]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf

[6] *Apache Tomcat* [online]. The Apache Software Foundation, 2017 [cit. 2017-12-04]. Dostupné z: http://tomcat.apache.org/

[7] *Introduction* [online]. The Apache Software Foundation, 2017, 2017-11-29 [cit. 2017-12-06]. Dostupné z: https://maven.apache.org/what-is-maven.html

[8] DASHORST, Martijn. a Eelco. HILLENIUS. *Wicket in action*. Greenwich, CT: Manning, 2009. ISBN 19-323-9498-2.

[9] *Hibernate ORM* [online]. Red Hat, Inc., 2017 [cit. 2017-12-08]. Dostupné z: http://hibernate.org/orm/

[10] *Node.js* [online]. Node.js Foundation, 2017 [cit. 2017-12-05]. Dostupné z: https://nodejs.org/en/

[11] NGUYEN, Don. *Jump start Node.js*. New Edition. Collingwood, Vic., Australia: SitePoint Pty, 2012. ISBN 978-098-7332-103.

[12] *What is npm?* [online]. npm, Inc., 2017, September 19, 2017 [cit. 2017-12-05]. Dostupné z: https://docs.npmjs.com/getting-started/what-is-npm

[13] *Express: Routing* [online]. StrongLoop, 2017 [cit. 2017-12-10]. Dostupné z: http://expressjs.com/en/guide/routing.html

[14] DEPOLD, Sascha. *Sequelize* [online]. Sequelize, 2017 [cit. 2017-12-10]. Dostupné z: http://docs.sequelizejs.com/

[15] *EJS* [online]. Bitovi, 2017 [cit. 2017-12-05]. Dostupné z: http://www.embeddedjs.com/

[16] *Passport: Overview* [online]. 2017 [cit. 2017-12-09]. Dostupné z: http://www.passportjs.org/docs/

[17] REINMAN, Andris. *Nodemailer* [online]. 2017 [cit. 2017-12-10]. Dostupné z: https://nodemailer.com/about/

Seznam zkratek

API Application Programming Interface

CSS Cascading Style Sheets

EJS Embedded Javascript

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

MVC Model-view-controller

NPM Node package manager

ORM Object Relation Mapping

POM Project Object Model

URL Uniform Resource Locator

XML Extensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam příloh

K této práci je přiloženo CD/DVD s následující adresářovou strukturou.

- Praktická část
    - Script pro vygenerování databáze
    - javaRezervace – Aplikace vytvořená pomocí platformy Java
    - nodeRezervace – Aplikace vytvořena pomocí platformy Node
- Szabo_BP_2017 – Bakalářská práce ve formátu PDF