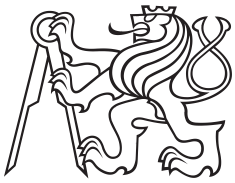
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Informační systém pro správu autoservisu

David Přáda

Vedoucí: Ing. Božena Mannová, Ph.D.
Obor: Softwarové inženýrství a technologie
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Přáda** Jméno: **David** Osobní číslo: **478206**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Informační systém pro správu autoservisu

Název bakalářské práce anglicky:

Information system for car service management

Pokyny pro vypracování:

Seznamte se s problematikou provozu autoservisu a proveďte rešerši dostupných aplikací pro správu autoservisu. Na základě analýzy proveďte specifikaci a návrh aplikace pro správu autoservisu. Navrhněte strukturu databáze, vhodný způsob zabezpečení, uživatelský interface a vhodný způsob testování. Aplikaci implementujte a otestujte, výsledky vyhodnoťte. Při tvorbě aplikace používejte vhodné prostředky SE.

Seznam doporučené literatury:

[1] Pressmann R. S.: Software Enegineering,
[2] https://motofocus.cz/servisni-vybaveni/51352,novy-program-mechanic-pro-spravu-autoservisu
[3] http://jednickanatrhu.cz/program-pro-autoservis

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Božena Mannová, Ph.D., kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Božena Mannová, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Boženě Mannové, Ph.D. za vstřícný přístup a pomoc se všemi otázkami a problémy, které jsem během práce měl. Její rady mi pomohly už od samotného začátku. Dále bych rád poděkoval svojí rodině a přátelům, kteří se mnou měli trpělivost a podporovali mě.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatné a že jsem uvedl veškeré použité informací zdroje v souladu s Metodickými pokyny o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2021

Abstrakt

Tato práce je zaměřena na analýzu, návrh řešení a implementaci informačního systému pro autoservis. Systém je určen pro správu rezervací, faktur, zakázek, zákazníků a skladu. V první části se práce věnuje úvodu do problému. Dále se věnuje analýze a sbíraní požadavků. Na základě požadavků je vytvořen návrh aplikace a následně je podle návrhu aplikace implementována. Práce obsahuje popis aplikace a jejího uživatelského rozhraní. Poslední část této práce se věnuje testování aplikace.

Klíčová slova: Informační systém, autoservis, webová aplikace, analýza, návrh

Vedoucí: Ing. Božena Mannová, Ph.D.
Karlovo náměstí 13,
Praha 2

Abstract

This work is focused on the analysis, design and implementation of an information system for car service. The system is designed for the management of reservations, invoices, orders, customers and the warehouse. The first part deals with the introduction to the problem. Next part deals with the analysis and collection of requirements. Based on the requirements, an application design is created and then implemented according to the design. The thesis contains a description of the application and its user interface. The last part of this work is devoted to testing the application.

Keywords: Information System, car service, web application, analysis, design

Title translation: Information system for car service management

Obsah

1 Úvod **1**

1.1 Motivace 1

1.2 Cíle 1

2 Analýza **3**

2.1 Pojem informační systém 3

2.2 Existující řešení 4

2.2.1 Jednickanatrhu 4

2.2.2 Mechanic 5

2.2.3 Carsys 6

2.3 Funkční požadavky 6

2.3.1 Rezervace 6

2.3.2 Zakázky 7

2.3.3 Auta 7

2.3.4 Zákazníci 7

2.3.5 Sklad 7

2.3.6 Přihlášení 7

2.3.7 Rozvrh 8

2.3.8 Fakturace 8

2.3.9 Nastavení 8

2.4 Role v systému 8

2.4.1 Zaměstnanec 8

2.4.2 Manažer 9

2.4.3 Administrátor 9

2.5 Případy užití 10

2.6 Nefunkční požadavky 11

Platforma 11

Doba odezvy 11

Zabezpečení 11

2.7 Analýza technologií 11

2.7.1 Databáze 11

2.7.2 Server 12

2.7.3 Frontend 13

3 Návrh a implementace řešení **15**

3.1 Architektura 15

3.1.1 Třívrstvá architektura 15

3.1.2 MVC 16

3.2 Doménový model 17

3.3 Java 17

3.4 Java EE 18

3.5 Spring 18

3.5.1 Spring Boot 18

3.6 Spring security 19

3.7 JPA a Hibernate 20

3.7.1 JPA 20

3.7.2 Hibernate 20

3.8 Databáze 21
3.9 Maven 22
3.10 Vývojové prostředí 22
3.11 Struktura projektu 23
3.11.1 Frontend 23
3.11.2 Backend 24
3.12 Uživatelské rozhraní 25
3.12.1 React 25
3.13 Nasazení 26
4 Popis aplikace **27**
4.1 Přihlášení do aplikace 27
4.2 Rozvrh 28
4.3 Zakázky a rezervace 29
4.3.1 Úprava a vytvoření zakázky . 30
4.4 Zákazníci 30
4.4.1 Úprava a vytvoření zákazníka 31
4.5 Sklad 32
4.6 Uživatelé 34
4.6.1 Změna hesla 34
5 Testování **35**
6 Závěr **37**
Literatura **39**
A Seznam použitých zkratek **43**

Obrázky

2.1 Informační proces z pohledu toku informací [1] 3

2.2 Uživatelské rozhraní systému Jednickanatrhu [2] 5

2.3 Uživatelské rozhraní systému mechanic [3] 6

2.4 Diagram případů užití 10

3.1 Doménový model 17

3.2 Princip spring security filter chain[25] 19

3.3 Struktura frontend části aplikace 23

3.4 Struktura backend části aplikace 24

4.1 Přihlašovací obrazovka 28

4.2 Ukázka rozvrhu 29

4.3 Ukázka seznamu zakázek a rezervací 29

4.4 Ukázka editace zakázky 30

4.5 Ukázka seznamu zákazníků 31

4.6 Ukázka úpravy zákazníka 31

4.7 Ukázka seznamu aut zákazníka . 32

4.8 Ukázka seznamu položek na skladu 32

4.9 Ukázka úpravy položky na skladě 33

4.10 Ukázka seznamu aut kompatibilních s danou položkou . . 33

4.11 Ukázka vyhledávání položek podle typu auta 33

4.12 Ukázka seznamu uživatelů 34

4.13 Ukázka změny hesla 34

Kapitola 1

Úvod

1.1 Motivace

Automobilový průmysl se stále rozvíjí a i přes inovace ve spolehlivosti jsou autoservisy nepostradatelnou složkou našich služeb. Pomáhají nám dnes a denně. Pro každý autoservis je však velmi důležité minimalizovat náklady a co nejlépe využít své zdroje. Informační systém je tak ve správné formě velmi užitečný nástroj každého servisu. S budoucností v elektromobilitě je také důležité, aby byl systém adaptivní a dokázal pracovat s novými vozidly, jejich prvky, procesy oprav a údržeb.

1.2 Cíle

Cílem bakalářské práce je provést analýzu, navrhnout a implementovat aplikaci pro správu malých a středně velkých autoservisů. Aplikace by měla být intuitivní, přehledná a jednoduchá na požívání. Cílem aplikace je zefektivnit dělbu práce, využití času a zpříjemnit zaměstnancům práci se systémem. Aplikace bude umožňovat správu zákazníků, zakázek, skladu a samotných uživatelů systému.

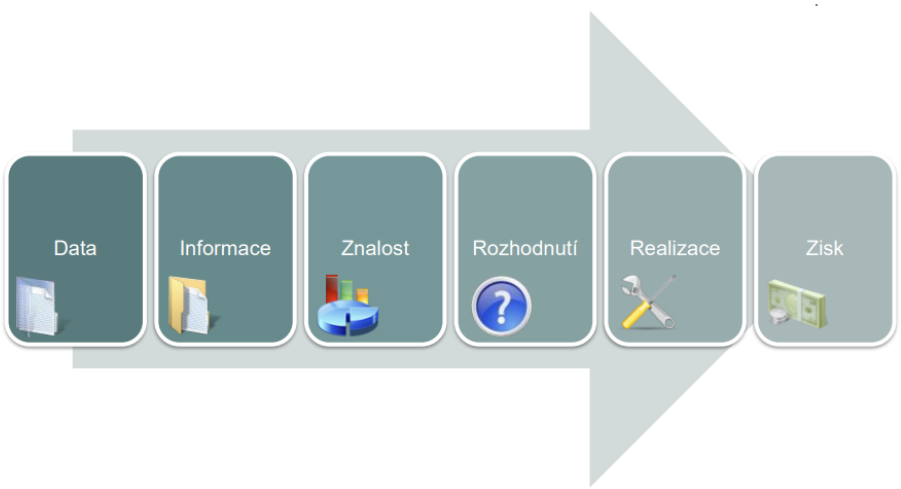
Kapitola 2

Analýza

2.1 Pojem informační systém

Definice: IS neboli informační systém můžeme definovat jako systém sběru, uchování, analýzy a prezentace dat. [1]

Informace jsou v praxi například data, účetní evidence, evidence skladu, evidence zákazníků a mnoho dalšího. [1] Znalost takových informací vede k lepšímu rozhodování a to při správném řízení vede k lepším službám a ziskům.



Obrázek 2.1: Informační proces z pohledu toku informací [1]

Tyto informace je dobré mít "po ruce", nejlépe všechny a rychle. K tomu se dnes z velké části používají právě informační systémy. Poskytují informace různým uživatelům. Proto je to velmi důležitý a užitečný nástroj, pokud se

nachází v rukou uživatele, který s ním umí pracovat.

Shrnutí hlavních přínosů informačního systému:

- Centralizace zpracování informací
- Zjednodušení evidence
- Optimalizace práce
- Zvýšení efektivity práce
- Rychlost získávání informací

[1]

Vývoj informačního sytému začíná sběrem požadavků. Požadavky určují co by měl systém umět a jak by měl fungovat. Bez sběru požadavků je velmi pravděpodobné, že systému bude chybět nějaká vyžadovaná funkcionalita nebo bude mít funkci zcela odlišnou.

Druhou fází životního cyklu informačního systému je analýza. Porovnání všech možných řešení a návrh systému.

Třetí fází je samotná implementace, která však není poslední.

Poslední fází informační systému je podpora, která spočívá například ve školení uživatelů nebo opravování chyb. [1]

2.2 Existující řešení

Nejen proto abychom si lépe představili existující řešení, ale také proto, abychom na základě jejich znalostí věděli co by měl informační systém pro autoservis obsahovat, a jaké by měl splňovat požadavky, je dobré podívat se na již existující systémy.

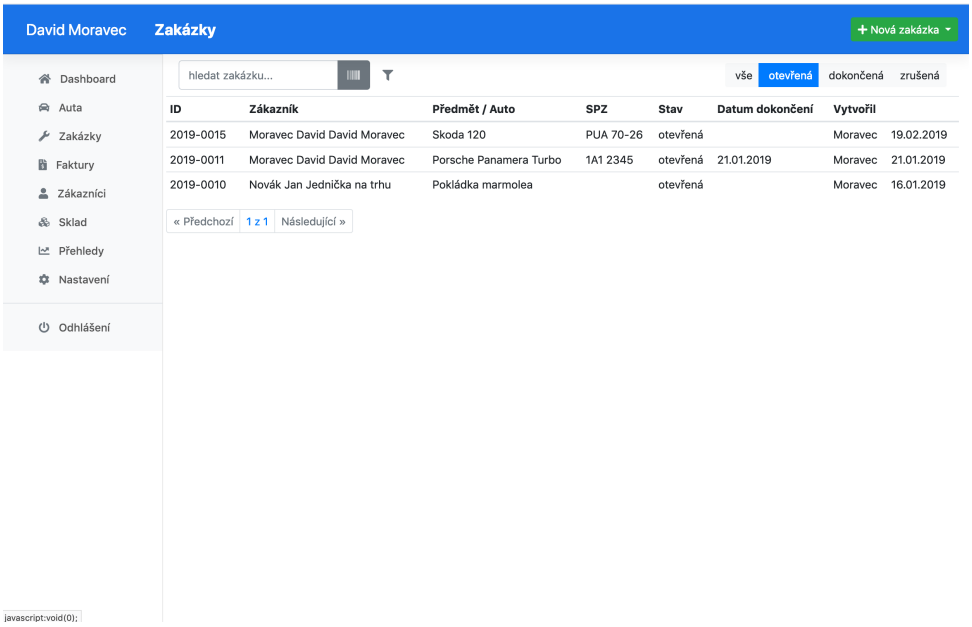
2.2.1 Jednickanatrhu

Software pro podnikatele, řemeslníky, firmy a autoservisy. Umožňuje vedení zakázek, vystavování faktur, skladové hospodářství, evidenci zákazníků. [2]

Příklady funkcí systému Jednickanatrhu:

Při otevření zakázky vyberete vozidlo, které se automaticky načte z databáze vozidel. Doplníte datum otevření zakázky a stav km při přijetí vozidla do opravy. Dále můžete vypsat závady, zákazníkem požadované opravy a odhadovanou cenu za opravu. [2]

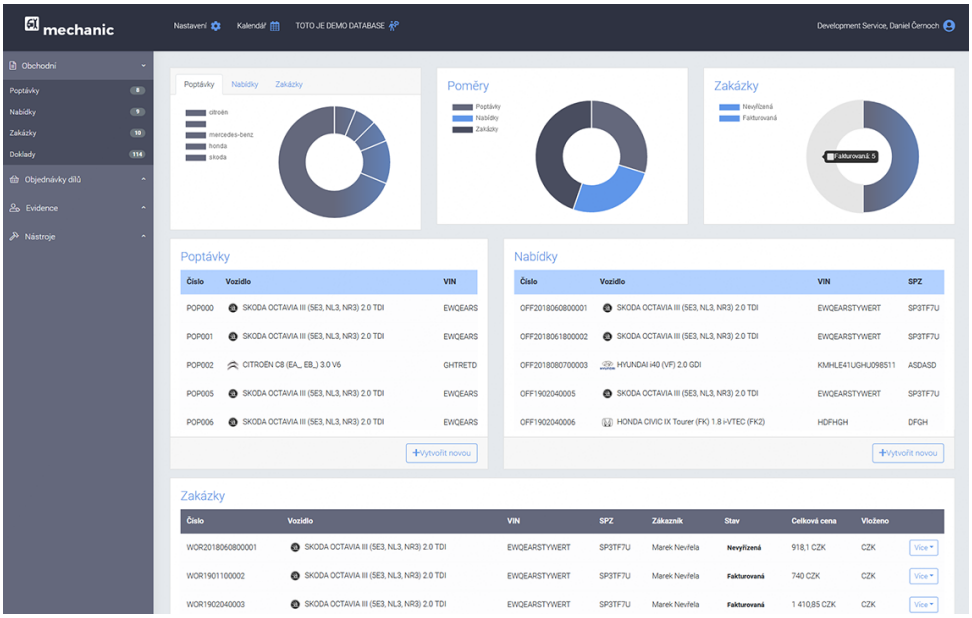
Na kartě zákazníka uvidíte seznam všech provedených oprav, zakázkových listů, faktur a vozidel zákazníka. Přímo z karty zákazníka můžete vytvořit zakázku, fakturu a zobrazit detaily jeho automobilu. Jednoduché vyhledávání umožňuje rychle nalézt požadované vozidlo. [2]



Obrázek 2.2: Uživatelské rozhraní systému Jednickanatrhu [2]

2.2.2 Mechanic

Program od firmy Nextis. Program Mechanic je určen pro malé nebo středně velké servisy, kterým umožní pohodlné a přesné vedení agendy. Mechanic nepotřebuje žádnou instalaci a vše se odehrává v prostředí internetového prohlížeče. [3]



Obrázek 2.3: Uživatelské rozhraní systému mechanic [3]

2.2.3 Carsys

Carsystem je informační systém pro prodejce a servisy automobilů, stavebních nebo zemědělských strojů. Dodává se ve třech verzí, které se liší rozsahem funkcí a cenou. [4]

2.3 Funkční požadavky

Funkční požadavky byly vytvořeny na základě analýzy již existujících řešení, vlastních nápadů a také díky vlastní zkušenosti s prací v autoservisu. Při práci v autoservisu jsem zjistil jak některé procesy autoservisu fungují, což mi s vytvářením těchto požadavků pomohlo.

2.3.1 Rezervace

Systém bude umožňovat zobrazení, vytvoření, úpravu, zrušení a vyhledávání rezervací. Rezervací se myslí zakázka ve stavu rezervace. Při vytvoření rezervace je potřeba určit datum, čas, popis opravy nebo údržby. Rezervace se mohou časově překrývat, ale díky rozvrhu uživatelé vidí kolik rezervací je v jaký den a čas již vytvořeno.

2.3.2 Zakázky

Systém bude umožňovat zobrazení, vytvoření, ukončení, úpravu, zrušení a vyhledávání zakázek. Zakázka má přiřazené auto, zákazníka a zaměstnance. Zahrnuta je v tomto požadavku také historie zakázek, která by mohla například umožnit generovat statistická data.

2.3.3 Auta

Systém bude umožňovat zobrazení, vytvoření, smazání a úpravu informací o autech zákazníků. Auto má výrobce, model, rok výroby a poznávací značku, díky které můžeme auto jednoduše identifikovat.

2.3.4 Zákazníci

Systém bude umožňovat zobrazení, vytvoření, úpravu, odstranění a vyhledávání informací o zákaznících. Zákazník má jméno, příjmení, email, adresu a kontaktní telefon. Jediný povinný atribut u zákazníka je email. Tímto způsobem se mohou vytvářet i malé jednorázové zakázky bez zbytečného vyplňování kontaktních údajů.

2.3.5 Sklad

Systém bude umožňovat zobrazení, vytvoření, úpravu, odstranění, vyskladnění a vyhledávání položek na skladě. Systém u položky rozlišuje název, umístění, počet kusů na skladě a cenu. Uživatel tak bude mít možnost například rychle vyhledat položku a zjistit v jaké části skladu se nachází. Položky mohou mít také seznam aut, se kterými jsou kompatibilní. Tímto způsobem můžeme položky vyhledávat také podle typu auta.

2.3.6 Přihlášení

Systém bude umožňovat přihlášení a odhlášení uživatele. Uživatel se přihlásí pomocí uživatelského jména a hesla. Uživatelské jméno je unikátní a tak dva uživatelé nemohou mít stejné.

2.3.7 Rozvrh

Systém bude umožňovat zobrazit rozvrh probíhajících a budoucích zakázek. Zaměstnanci servisu tak budou mít přehled v zakázkách jak na aktuální den, tak například na aktuální nebo nadcházející týden. Mohou díky tomu tak zjistit případné problémy, například překrytí zakázek, na které jsou potřeba stejná zařízení a nebylo by je tak možné vyřídit najednou. Interval zakázek je možné rychle přepínat na stránce s rozvrhem.

2.3.8 Fakturace

Systém bude umožňovat generování a zobrazení faktur zakázek. V systému je přednastavená šablona pro generování faktur.

2.3.9 Nastavení

Systém bude umožňovat uživateli změnu hesla. Heslo musí mít minimálně 6 znaků. Při změně hesla je nutné vyplnit nové heslo dvakrát.

2.4 Role v systému

Systém bude obsahovat několik rolí především pro rozdělení přístupu uživatelů. Nemůže se tak stát, že by mohl například zaměstnanec spravovat účty ostatních zaměstnanců v systému.

2.4.1 Zaměstnanec

Zaměstnanec může:

- Přihlásit se do systému
- Odhlásit se ze systému
- Změnit svoje heslo
- Změnit barvy uživatelského rozhraní
- Zobrazit všechny rezervace, zobrazit detaily rezervace, upravit, zrušit a vytvořit rezervaci

- Zobrazit všechny zakázky, zobrazit detaily zakázky, upravit, ukončit, zrušit a vytvořit zakázku
- Zobrazit všechny zákazníky, zobrazit detaily zákazníka, upravit, zrušit a vytvořit zákazníka
- Zobrazit všechny položky na skladě, zobrazit detail položky, vyskladnit, upravit, zrušit a vytvořit položku
- Zobrazit všechny kompatibilní typy automobilů s konkrétní položkou, tyto typy přidávat a mazat
- Zvolit zakázku a přiřadit ji sobě
- Generovat a zobrazit faktury zakázek
- Vyhledávat rezervace, zakázky, položky na skladě a zákazníky

2.4.2 Manažer

Manažer servisu může:

- vykonávat všechny funkce jako zaměstnanec
- vytvářet a mazat zaměstnance

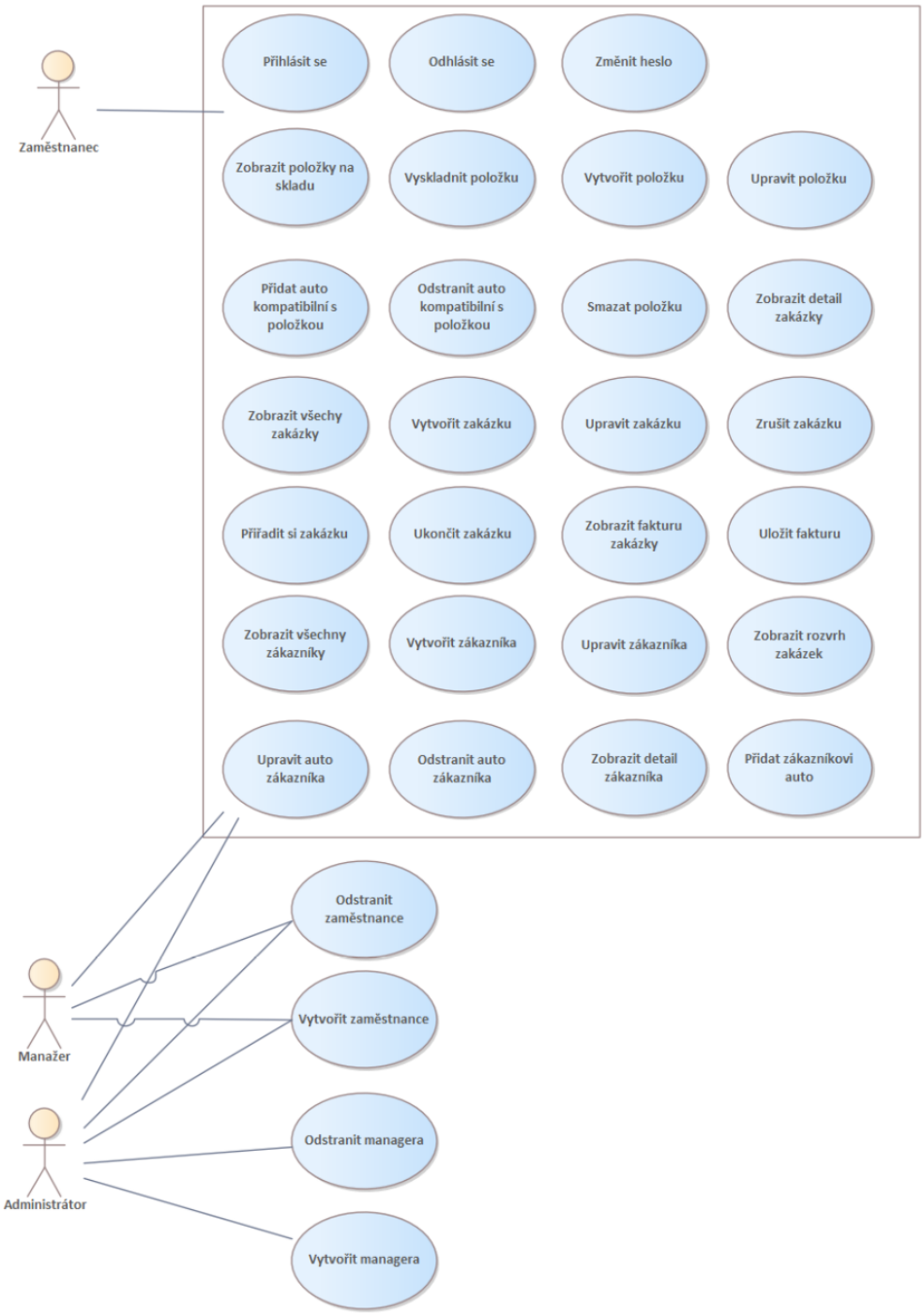
2.4.3 Administrátor

Administrátor může:

- vykonávat všechny funkce jako zaměstnanec
- vytvářet a mazat manažery a zaměstnance

2.5 Případy užití

Případy užití vycházejí ze systémových požadavků z předchozí sekce.



Obrázek 2.4: Diagram případů užití

2.6 Nefunkční požadavky

Abychom udrželi dostatečnou kvalitu softwaru, specifikujeme i požadavky nefunkční.

Platforma

Systém poběží jako webová aplikace na webových prohlížečích Google Chrome, Mozilla Firefox, Microsoft Edge a Opera. Tyto prohlížeče patří k nejpoužívanějším a je tak nutné aby minimálně tyto prohlížeče aplikace podporovala. Zároveň jsou zvoleny tak, aby měl uživatel možnost používat aplikaci v prostředí, které dobře zná.

Doba odezvy

95% žádostí by měl systém zpracovat do 4 sec a žádná žádost nesmí trvat déle než 15 sec.

Zabezpečení

Systém bude obsahovat citlivá data zákazníků a servisu, proto je nutné aby se k datům nedostal nikdo jiný než zaměstnanci a uživatelé s povolením. Přihlášení do systému vyžaduje uživatelské jméno a heslo. Pokud uživatel zadá špatné jméno nebo heslo, systém ho upozorní a nepustí do systému.

2.7 Analýza technologií

Zvolení vhodných technologií může výrazně zefektivnit a zrychlit vývoj softwaru. Je tak důležité se na tuto část zaměřit a porovnat si možnosti řešení pro jednotlivé části systému.

2.7.1 Databáze

Výběr databázového systému není tolik důležitý jako ostatní části, protože při implementaci systému s ním budeme pracovat minimálně. Výhodou u této

části je pro mě zkušenost s velkým počtem databázových systémů a to díky střední škole, kde byly tyto systémy značnou částí výuky.

Oracle

Oracle má zkušenosti s tvorbou databázových systému od roku 1979. Jeho DBMS neboli database management system je výkonný ale zároveň komplexní a pro začátečníky složitější. Je tak určena pro vetší projekty a zároveň přichází s velkými náklady na pořízení. [5]

Microsoft SQL

Jeden z nejpoužívanějších DBMS od společnosti Microsoft, který vznikl v roce 1989. Jako výhodu bych zmínil, že je jednoduchý na pochopení a má intuitivní uživatelské rozhraní. Jeho nevýhoda je nutnost použití operačního systému Windows. [6, 8]

MySQL

Open-source alternativa k MicrosoftSQL dnes vlastněna společností Oracle, kterou používá několik významných technologických společností jako například Facebook, Google a Adobe. Klade důraz na spolehlivost více než na počet funkcí. [5, 8]

PostgreSQL

PostgreSQL je open-source objektově-relační databázový systém. Tento systém je vyvíjen přes 30 let a za tu dobu si vysloužil reputaci jako spolehlivý, odolný a výkonný databázový systém. [7]

2.7.2 Server

Aplikace bude samozřejmě potřebovat server, který bude zpracovávat požadavky od klientů. Opět se nám nabízí několik možností.

Apache Tomcat

Apache Tomcat je open-source Java servlet a Java server page kontejner. Je to nejpoužívanější web server v oblasti Java aplikací. Díky svému open-source přístupu prošel od svého vydání v roce 1999 spoustou vylepšení a nyní je již ve verzi 8.5. [9, 10]

Jetty

Jetty začal jako open-source projekt v roce 1995 a jeho hlavními výhodami jsou jeho kompaktnost a efektivita. Používá ho řada aplikací jako například Google App Engine, Apache Hadoop nebo Apache ActiveMQ. [10, 11]

GlassFish

Glassfish je JavaEE server vyvíjený společností Oracle. [10]

2.7.3 Frontend

React

Zřejmě nejpopulárnější frontend framework React je javascriptová knihovna pro tvorbu interaktivních webových rozhraní a rozhraní mobilních aplikací. Je open-source a stará se pouze o prezenční vrstvu. [12]

Angular

Na rozdíl od Reactu, Angular využívá takzvanou oboustrannou datovou vazbu. To znamená, že dokáže v reálném čase synchronizovat model a view, kde jakákoliv změna v modelu se okamžitě projeví i na view. Je vhodný na vícestránkové aplikace a používají ho firmy jako například BMW, Xbox, Forbes nebo Blender. V porovnání s Reactem je sice složitější na naučení, ale pro tvorbu enterprise aplikací je vhodný. [13]

Vue

Vue je progresivní framework pro tvorbu uživatelského rozhraní. Stejně jako Angular používá oboustrannou datovou vazbu. Základní knihovna je zaměřena pouze na prezenční vrstvu a je jednoduchá na pochopení. Nevýhodou je bohužel stabilita komponent a malá podpora od uživatelů. [13, 14]

Kapitola 3

Návrh a implementace řešení

3.1 Architektura

3.1.1 Třívrstvá architektura

Třívrstvá architektura (anglicky Three-tier architecture) označuje jeden z typů architektury informačních systémů (resp. aplikací). Tedy to, jakým způsobem je aplikace rozdělena mezi to, co vidí a používá uživatel (tzv. prezentační vrstva) a to, co se odehrává na pozadí na straně serveru (aplikační a datová vrstva). [17]

Třívrstvou architekturu využívá velké množství aplikací, které pracují s daty. Takto je postavena většina moderních podnikových aplikací, některá portálová řešení a webové stránky. Tří a vícevrstvá architektura je v dnešní době trendem a využívá se pro tvorbu robustnějších řešení. Její výhodou je pružnější rozdělení výkonu mezi zařízení uživatele a server, prezentační vrstva může běžet i na velmi levných zařízeních. [17]

Vrstvy architektury:

- Prezentační vrstva - ta část, která je viditelná pro uživatele, zajišťuje vstup požadavků a prezentaci výsledků. Je závislá na platformě (např. webová aplikace, Aplikace pro windows, Android aplikace atd.). Může být tedy různá pro různá zařízení či platformy.
- Aplikační vrstva (také funkční) - prostřední vrstva modelu (middleware), zajišťuje výpočty a operace prováděné mezi vstupně-výstupními požadavky a daty. Také nazývána jako aplikační server.

- Datová vrstva (také databázová) - nejnižší vrstva modelu, zajišťuje práci s daty, tedy s systém řízení báze dat a základní datově-funkční operace zajišťující ukládání, výběr, agregaci, předzpracování, integritu a audit dat.

[17]

Pro typ systému, na který se tato práce zaměřuje je tato architektura vhodná. Poskytuje samostatnost jednotlivých částí aplikace, díky které můžeme provádět rychlé změny a úpravy. To znamená, že například změna v aplikační vrstvě nijak neovlivní vrstvu prezentační a naopak.

V našem případě jsou vrstvy zastoupeny takto:

- Prezentační vrstva - uživatelské rozhraní implementované frameworkem React.
- Aplikační vrstva - implementovaná pomocí Spring Boot frameworku v jazyce Java.
- Datová vrstva - databáze typu PostgreSQL.

3.1.2 MVC

MVC je velmi oblíbený architektonický vzor, který se uchytil zejména na webu, ačkoli původně vznikl na desktopech. Základní myšlenkou MVC architektury je oddělení logiky od výstupu. [18] K tomu používáme takzvané komponenty, které jsou tři, a to model, view a controller.

Model obsahuje logiku a vše, co do ní spadá. Mohou to být výpočty, databázové dotazy, validace a podobně. Model vůbec neví o výstupu. Jeho funkce spočívá v přijetí parametrů zvenku a vydání dat ven. [18]

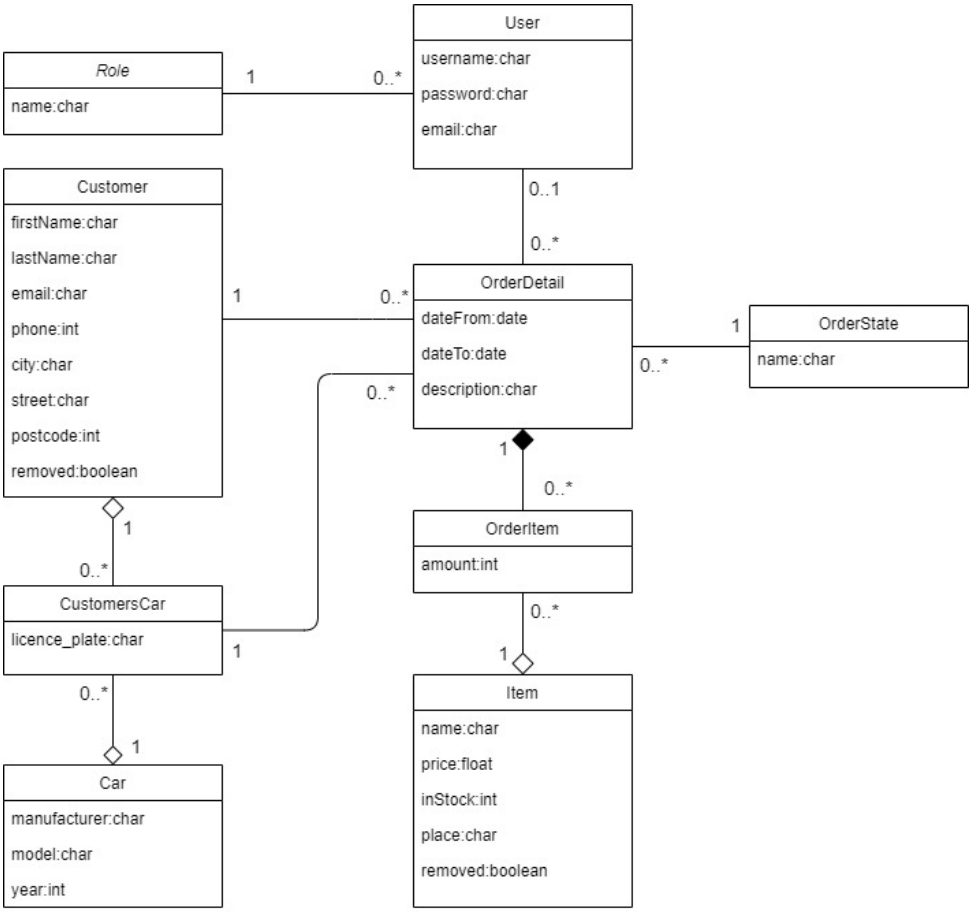
View se stará o zobrazení výstupu uživateli. Nejčastěji se jedná o phtml šablonu, obsahující HTML stránku a tagy nějakého značkovacího jazyka, který umožňuje do šablony vkládat proměnné, případně provádět iterace (cykly) a podmínky. View není jen šablona, ale zobrazovač výstupu. Obsahuje tedy minimální množství logiky, která je pro výpis nutná (např. kontrola, zda si uživatel vyplnil přezdívku před jejím vypsáním nebo cyklus s komentáři, které se vypisují). [18]

Controller je nyní onen chybějící prvek, který osvětlí funkčnost celého vzoru. Jedná se o jakéhosi prostředníka, se kterým komunikuje uživatel, model i view. Drží tedy celý systém pohromadě a komponenty propojuje. [18]

3.2 Doménový model

Doménový model popisuje klíčové entity, jejich atributy a vztahy v systému.

Tento model zároveň popisuje strukturu databáze s tím rozdílem, že pro Role a OrderState nemusíme vytvářet v databázi tabulky. Budou reprezentovány jako atributy příslušných entit a budou nabývat celočíselných hodnot. V systému se tak bude například role administrátora rovnat hodnotě 2 v databázi.



Obrázek 3.1: Doménový model

3.3 Java

Aplikace bude implementována z velké části v programovacím jazyku Java. Tento jazyk jsem zvolil, jelikož s ním mám nejvíce zkušeností. Java je programovací jazyk vyvinutý v roce 1995 firmou Sun Microsystems. Nyní ho vlastní

společnost Oracle, která Sun Microsystems v roce 2009 koupila. Dnes Java běží na více než 3 miliardách zařízení a je tak jeden z nejrozšířenějších programovacích jazyků na světě. Funguje na mnoha platformách, je open-source, má velmi obsáhlou dokumentaci a to vše zdarma.[20] V tomto projektu je použita verze 8.

3.4 Java EE

Java EE je součást platformy Java. Umožňuje jednodušší a rychlejší vývoj enterprise aplikací. Vývojářům dodává sadu rozhraní s cílem menší komplexity aplikace, zrychlení vývoje a zlepšení chodu aplikace. [15]

3.5 Spring

Spring je jeden z nejvíce používaných JavaEE frameworků pro tvorbu aplikací. Původně byl vyvinut Rod Johnsonem a vydán pod licencí Apache 2.0 v roce 2003. Používá již zmíněný Model-Vier-Controller a zjednodušuje tak vývoj. [21, 22]

Hlavní funkcí frameworku Spring je takzvaná Dependeci Injection a Inversion of Control. Inversion of Control je obecný pojem, ve zkratce se jedná o způsob vytváření programového kódu, kdy jeho potřebné závislosti jsou mu nastavovány z vnějšího prostředí, které jej využívá, namísto aby si je opatřoval resp. vytvářel sám. Dependency injection je konkrétním příkladem IoC, v aplikačním prostředí podporujícím IoC je DI obvykle úkolem specializované komponenty, tzv. DI kontejneru, která je implementací návrhového vzoru factory a která obstarává vytváření (poskytování) instancí opakovaně použitelných objektů, kterým pak předává potřebné závislé objekty jedním ze tří možných způsobů. [23]

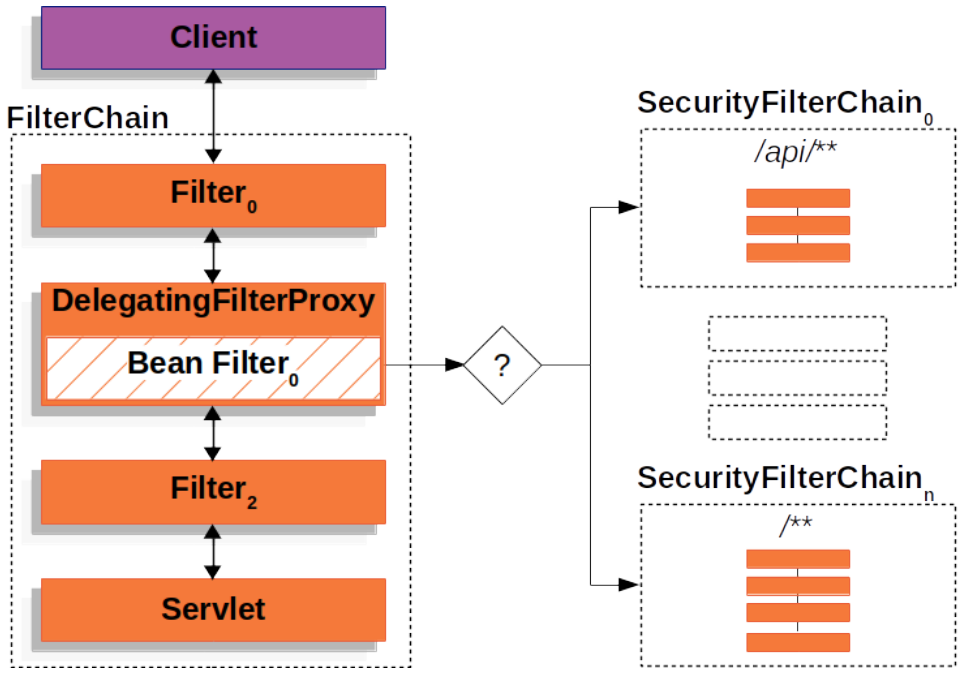
Spring jsem vybral také kvůli zkušenosti z předchozích školních předmětů, jako například Enterprise Architektury.

3.5.1 Spring Boot

Spring Boot je nadstavba frameworku Spring. Implementuje tedy veškeré funkce Springu, ale ještě k tomu nám poskytuje jednu velkou funkci navíc, která nám tvorbu projektu velice zrychlí a usnadní. Má totiž v sobě embed Tomcat, tedy server, na kterém aplikace běží a který není potřeba nijak složitě nastavovat. [21, 24]

3.6 Spring security

Součástí Springu je také Spring security. Jedná se o flexibilní framework, který poskytuje autentifikaci, autorizaci a ochranu před základními útoky na java aplikace. Autentifikací se myslí kontrola uživatelských údajů, jako například jméno a heslo. Autorizací se myslí kontrola práv uživatele na danou operaci. Stejně jako Spring Boot je Spring security velmi jednoduchý na nastavení a konfiguraci. Funguje díky takzvaným servlet filters nebo chain of filters. Servlet je ve zkratce java program, který zpracovává HTTP požadavky. Filtry každý požadavek zastaví před tím, než se dostane na servlet a ověří zda může požadavek pokračovat dál.[25]



Obrázek 3.2: Princip spring security filter chain[25]

Zde je nastavení Spring security aplikace:

Listing 3.1: nastavení spring security

```
http
        .cors().and()
        .authorizeRequests()
        .antMatchers("/*").hasAnyRole("EMPLOYEE","ADMIN","MANAGER")
        .anyRequest().permitAll().and()
        .exceptionHandling().authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.U
        .and().headers().frameOptions().sameOrigin()
        .and().authenticationProvider(authenticationProvider)
        .csrf().disable()
        .formLogin().successHandler(authenticationSuccessHandler)
        .failureHandler(authenticationFailureHandler)
```

```
    .loginProcessingUrl("/loginCheck")
    .usernameParameter("username").passwordParameter("password")
    .and()
    .logout()
    .logoutUrl("/logout").logoutSuccessHandler(logoutSuccessHandler)
    .and().sessionManagement().maximumSessions(1);
```

V nastavení můžeme vidět, že zabezpečení zakazuje všem uživatelům, kteří nemají přidělenou roli, vstup na jakoukoliv stránku. My samozřejmě chceme, aby takový uživatel měl přístup na přihlašovací formulář. To můžeme provést takto:

Listing 3.2: nastavení spring security č.2

```
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers("/","/login");
}
```

Díky spring security můžeme také u REST controllerů používat anotace @PreAuthorize. Tímto můžeme používání endpointů omezit například podle rolí uživatelů.

Listing 3.3: ukázka anotace @PreAuthorize

```
@PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_MANAGER')")
@GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
public List<User> getUsers(){
    return userService.findAll();
}
```

3.7 JPA a Hibernate

3.7.1 JPA

JPA neboli Java Persistence API je technologie, která umožňuje objektově relační mapování. Díky tomu můžeme pracovat přímo s objekty oproti SQL dotazům. Tato technologie výrazně zjednoduší práci s daty. Samotné JPA není nástroj nebo framework, ale pouze popisuje rozhraní a chování knihoven pro objektově relační mapování.[16]

3.7.2 Hibernate

K JPA tedy potřebujeme i samotnou implementaci. V dnešní době máme na výběr hned z několika možností, ale v této práci je použita implementace

Hibernate. Hibernate podporuje mnoho databázových dialektů. Umožňuje pracovat například s MySQL, PostgreSQL, Oracle, Microsoft SQL Server aj. [26]

Silným nástrojem Hiberate ORM je popis vztahů mezi entitami. Každé dvě entity spolu mohou být v nějakém vztahu (relaci), přičemž jedna strana je vlastníkem relace a druhou nazýváme „inverzní strana“. Vlastník relace je zodpovědný za udržování konzistence v datech na obou stranách relace. Pro Hibernate je entita jedna Java třída, jejíž instance se ukládají jako záznamy do jedné databázové tabulky. Každá instance entity má svůj identifikátor, který se obvykle použije jako primární klíč do tabulky. Místo mapovacích souborů se dají použít anotace přímo u entitních tříd a jejich členských proměnných. V mapovacích souborech definujeme mapování entit na databázové tabulky. Názvy anotací odpovídají názvům vlastností v mapovacích souborech.[26]

Listing 3.4: ukázka anotací entit

```
@Entity
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotBlank
    private String manufacturer;

    @NotBlank
    private String model;

    @Min(value = 1886)
    private int year;

    @JsonIgnore
    @ManyToMany(fetch = FetchType.LAZY, mappedBy = "cars")
    private Set<Item> items = new HashSet<>();

    }
```

@NotBlank a @Min jsou validační anotace. Vytváří omezení na atributech entit. Například @Min(value = 1886) znamená, že při uložení objektu nesmí být tento atribut menší než 1886. @NotBlank pak nedovoluje prázdné atributy.

3.8 Databáze

Jako databázový systém je zvolen PostgreSQL, protože mám zkušenosti s propojením této databáze a Java aplikací. Postgres má také jednoduchého

klienta pgAdmin, který může sloužit pro testování funkčnosti aplikace.

K vytvoření databáze je použita služba Heroku, která poskytuje vytvoření zdarma. Aby aplikace používala námi vytvořenou databázi, je potřeba nastavit v souboru application.properties několik parametrů.

Listing 3.5: nastavení postgresql databáze pro spring

```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:postgresql:
//ec2-52-50-171-4.eu-west-1.compute.amazonaws.com:5432/ddtu2dfjbhrqfv
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver
```

3.9 Maven

Maven je rozšířený systém pro správu a sestavování aplikací postavených nad platformou Java. Jeho využitím odpadá závislost na konkrétním IDE, protože všechny informace potřebné ke kompilaci a sestavení programu jsou přímo obsaženy ve speciálních souborech pom.xml (POM = project object model). Systém Maven je již plně integrován do všech velkých IDE (Eclipse, Netbeans, IDEA) a tak je práce s ním opravdu velmi snadná i přes uživatelské rozhraní. [27]

Principem systému Maven je vytvoření objektového modelu nad zdrojovým kódem, se kterým lze provádět různé operace. Nejčastěji se jedná o kompilaci, kontrolu, vytvoření balíků, atd. Model projektu je definován v souborech pom.xml, které se nachází v kořenovém adresáři každého projektu. V těchto adresářích lze spouštět příkazy mvn s parametry, které načtou model z odpovídajícího souboru pom.xml a provedou zadanou akci.[27]

3.10 Vývojové prostředí

Jako hlavní vývojové prostředí bylo použito IntelliJ Idea. Toto prostředí je přehledné, intuitivní a má velké množství návodů a pomůcek jak od vývojářů tak od komunity uživatelů. Jedním z důvodů pro výběr IntelliJ Idea byla v neposlední řadě také předchozí zkušenost s vývojem v tomto prostředí. V tomto prostředí bylo implementováno i uživatelské rozhraní.

3.11 Struktura projektu

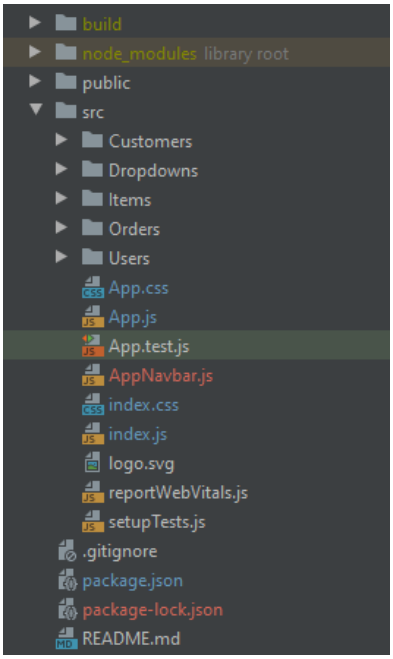
Projekt je rozdělený na dvě hlavní části, a to sice na frontend implementaci a backend implementaci.

3.11.1 Frontend

React komponenty se nachází ve složce /frontend/src a navíc jsou rozděleny podle objektů, se kterými pracují.

- /items/ komponenty, které se starají o položky na skladě.
- /customer/ komponenty, které se starají o zákazníky.
- /orders/ komponenty, které se starají o rezervace a zakázky.
- /users/ komponenty, které se starají o uživatele aplikace.
- /dropdowns/ komponenty pro výběr z možností.
- React router se nachází v souboru App.js.
- AppNavbar.js obsahuje horní lištu aplikace.

Složka build obsahuje vygenerovaný build, který používá backend.

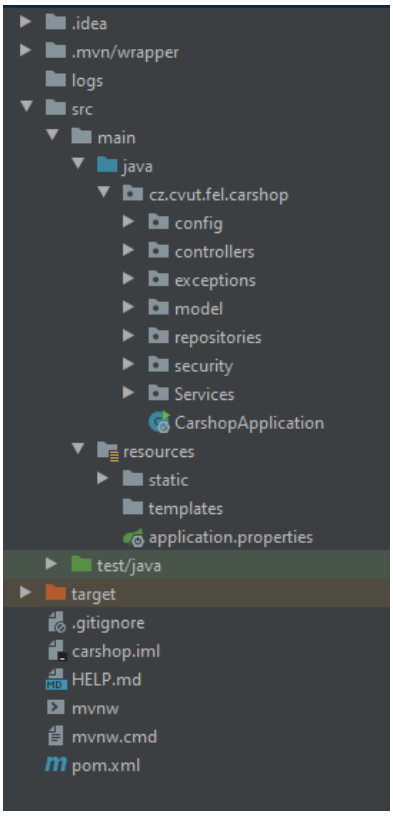


Obrázek 3.3: Struktura frontend části aplikace

3.11.2 Backend

Kód aplikace se nachází ve složce /backend/carshop/src/main/java/cz/cvut/-fel/carshop/ Dále je rozdělen do standardní struktury Spring projektu:

- /config/ obsahuje nastavení zabezpečení, JPA a ostatní nastavení
- /controllers/ obsahuje REST controllery, tedy API pro frontend
- /exception/ obsahuje vlastní výjimky
- /model/ obsahuje třídy/entity, které reprezentují data v aplikaci
- /repositories/ obsahuje rozhraní, které pracují přímo s databází
- /security/ obsahuje logiku a dodatečné nastavení zabezpečení
- /services/ obsahuje logiku aplikace



Obrázek 3.4: Struktura backend části aplikace

Ve složce /resources se nachází soubor application.properities a složka static, která obsahuje vygenerovaný build frontend části aplikace.

Ve složce /src/main/test/ se nachází JUnit testy.

3.12 Uživatelské rozhraní

Pravděpodobně jste již někdy narazili na systém nebo aplikaci, která uměla přesně to co jste chtěli, ale její uživatelské rozhraní bylo nepromyšlené a ošklivé na pohled. Chtěli jste aplikaci použít, ale nevěděli jste jak, na co kliknout, kde se nachází sekce, kterou potřebujete zobrazit. Takové uživatelské rozhraní dokáže být frustrující a může být příčinou nevyhnutelné odinstalace.

Při návrhu uživatelského rozhraní informační systému je tak důležitá především přehlednost. Dále by mělo být rozhraní pro uživatele intuitivní a neobsahovat mnoho informací najednou. A správný výběr barevné palety dokáže zpříjemnit používání systému.

3.12.1 React

Uživatelské rozhraní bylo implementováno s pomocí frameworku React. Před touto prací jsem s tímto frameworkem neměl žádné zkušenosti, ale ukázalo se, že pro mě nebyl moc složitý na pochopení. Implementace kvůli nulové předchozí zkušenosti určitě není perfektní, ale je funkční.

Hlavním rysem tohoto frameworku jsou takzvané komponenty, které je možné vytvářet jako třídy nebo jako funkce. Tyto komponenty používají stavy, ve kterých se ukládají data komponent, často se jedná například o data od uživatele. Data ve stavech potom můžou předávat svým potomkům pomocí takzvaných props. Komponenta může mít funkci, která definuje, co se vykreslí na obrazovce při jejím zavolání.

Speciální knihovna React Router potom synchronizuje URL webového prohlížeče s komponentami, které se mají na daném URL vykreslit.

Listing 3.6: react router aplikace

```
<Router>
  <Switch>
    <Route path='/' exact={true} component={Login}/>
    <Route path='/login' exact={true} component={Login}/>
    <Route path='/home' component={TimeTable}/>
    <Route path='/customers' exact={true} component={CustomerList}/>
    <Route path='/orders' exact={true} component={OrderList}/>
    <Route path='/users' exact={true} component={UserList}/>
    <Route exact path='/customers/:cid/cars/:id' component={CustomersCarEdit}/>
    <Route exact path='/customers/:id/cars' component={CustomersCarsList}/>
    <Route path='/customers/:id' component={CustomerEdit}/>
    <Route path='/orders/invoice/:id' component={OrderInvoice}/>
    <Route path='/orders/:id' component={OrderEdit}/>
    <Route path='/users/:id' component={UserEdit}/>
    <Route path='/items/:manufacturer/:model/:year/edit/:id' component={ItemEdit}/>
    <Route path='/items/:manufacturer/:model/:year' component={ItemsBySpecificCarList}/>
    <Route path='/items' component={ItemsSearchByCar}/>
    <Route path='/itemsList/edit/:itemId/cars/:carId' component={ItemsListCarsListEdit}/>
    <Route path='/itemsList/edit/:id/cars' component={ItemsListCarsList}/>
    <Route path='/itemsList/edit/:id' component={ItemsListEdit}/>
    <Route path='/itemsList' component={ItemsList}/>
    <Route path='/settings' component={PassChange}/>
  </Switch>
</Router>
```

3.13 Nasazení

Aplikace je nasazena pomocí služby Heroku a je možné jí vyzkoušet na adrese `https://davidscarshop.herokuapp.com/`. Pro přihlášení do systému můžete použít uživatelské jméno **david** a heslo **password**.

Pozor, Heroku aplikace po 30 minutách nepoužívání přejde do režimu spánku, proto může první načtení trvat i přes minutu. Po prvním načtení už aplikace reaguje rychle.

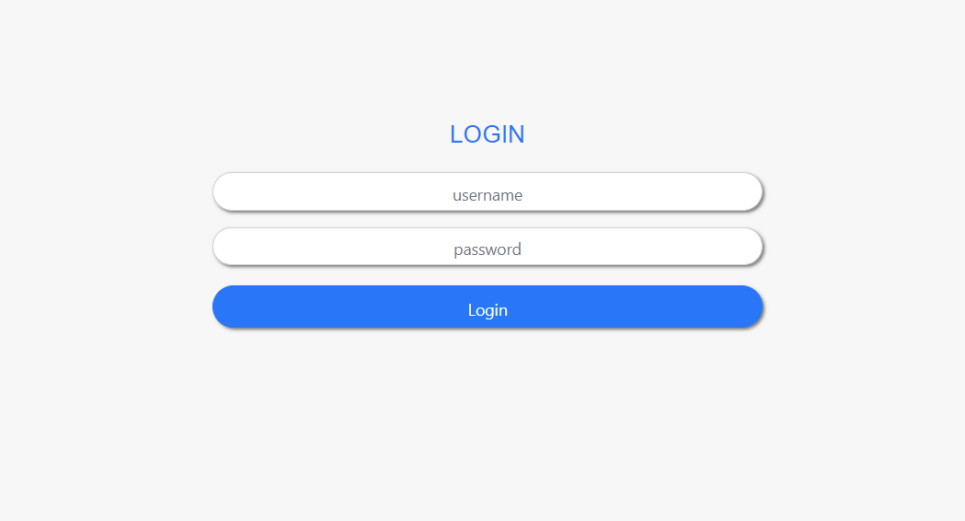
Kapitola 4

Popis aplikace

Tato kapitola je věnována představení uživatelského rozhraní a jeho fungování. U každé části je krátký popis akcí, které může uživatel na konkrétní stránce provádět.

4.1 Přihlášení do aplikace

Pro přístup do aplikace je nutné přihlášení uživatele. Stránka pro přihlášení je první která se objeví. Formulář má dva vstupy na uživatelské jméno a heslo. Pokud uživatel zadá špatné jméno nebo heslo, stránka ho upozorní a nepustí do aplikace. Pokud zadá správné údaje, stránka ho automaticky přesměruje na domovskou stránku aplikace pro přihlášené uživatele. Pokud se přihlášený uživatel odhlásí ze systému, bude automaticky přesměrován na tuto obrazovku. Přihlášení se provede po stisknutí tlačítka Login.

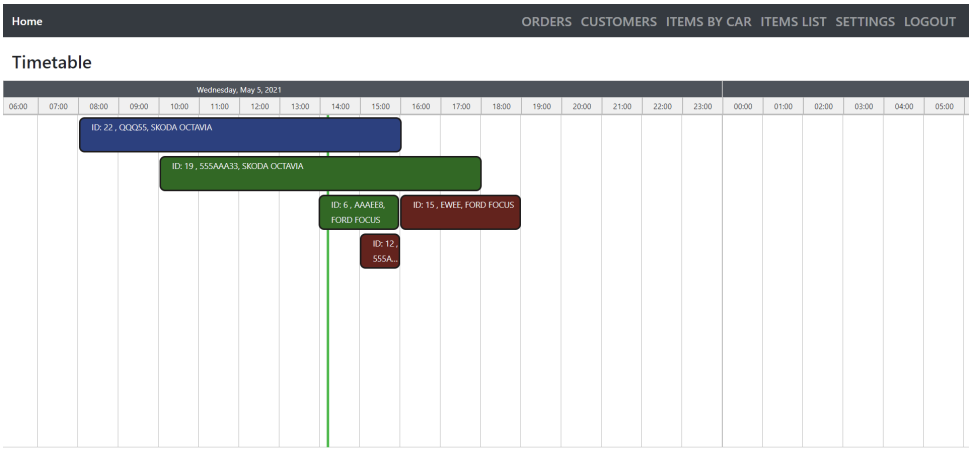


Obrázek 4.1: Přihlašovací obrazovka

Uživatel se může odhlásit tlačítkem Logout v horní liště.

4.2 Rozvrh

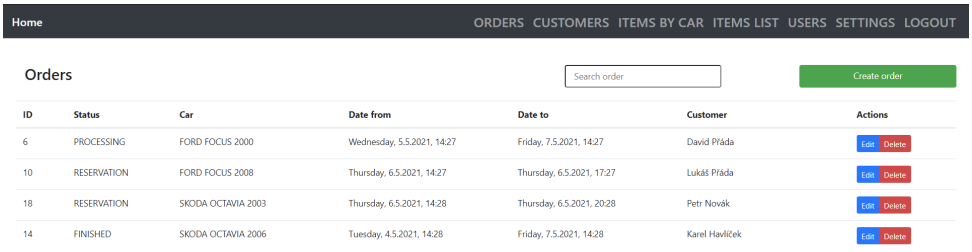
Po přihlášení se uživateli objeví stránka s rozvrhem zakázek a rezervací. Okamžitě tak vidí denní přehled a může si práci naplánovat. Rozvrh má dvě lišty, na obrázku jedna ukazuje aktuální den a druhá jednotlivé hodiny. Kliknutím na první ze zmíněných lišt může uživatel přepnout interval ze dne na měsíc a poté z měsíce na rok. Druhá lišta funguje opačně, interval zmenšuje. Zakázky jsou v rozvrhu barevně odlišené, modrá barva znamená, že zakázku má na starost právě přihlášený uživatel. Zelená zakázka není přiřazena nikomu. Červená zakázka je přiřazena jinému uživateli. Zelená vertikální čára znázorňuje aktuální čas a pohybuje se v reálném čase. Kliknutím na jakoukoliv zakázku se uživatel dostane na stránku s detaily dané zakázky a pokud není nikomu přiřazena, může si jí přiřadit sobě.



Obrázek 4.2: Ukázka rozvrhu

4.3 Zakázky a rezervace

Kliknutí na Orders v horní liště aplikace se dostaneme na seznam zakázek a rezervací. Seznam ukazuje nejdůležitější informace o zakázkách. Jejich číslo, stav, typ auta, datum zahájení a ukončení a jméno zákazníka. U každé zakázky jsou pak dvě tlačítka. Edit otevře stránku s detaily zakázky a umožní nám zakázku upravit, například pokud jsme zadali něco špatně. Delete zakázku odstraní. V seznamu můžeme také vyhledávat podle všech sloupců seznamu. Vyhledávání se aktualizuje s každou změnou vstupu, nemusíme tak na nic klikat. Tlačítko Create Order vedle vyhledávání otevře stránku, kde můžeme vytvořit novou zakázku nebo rezervaci. Stav zakázek se automaticky aktualizuje s časem.

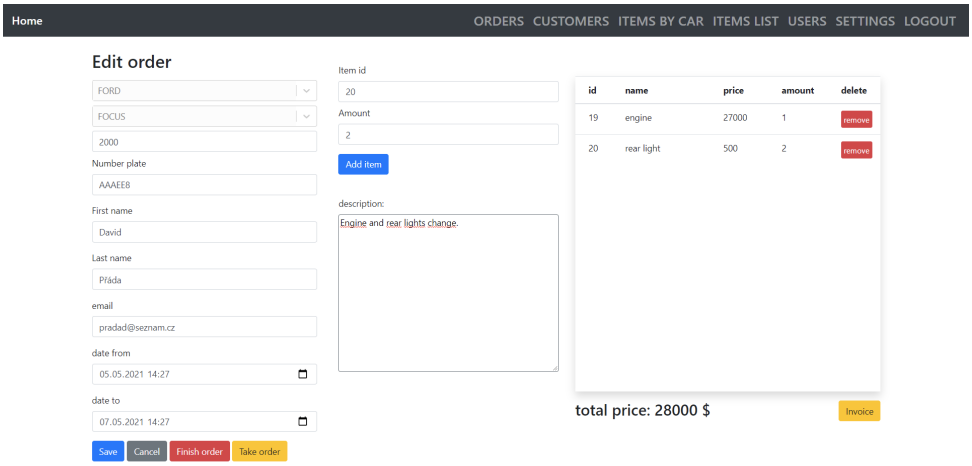


Obrázek 4.3: Ukázka seznamu zakázek a rezervací

4.3.1 Úprava a vytvoření zakázky

Na stránku s úpravou zakázky se můžeme dostat dvěma způsoby. Přes seznam zakázek, nebo přes rozvrh. Upravit můžeme typ auta, poznávací značku, jméno a email zákazníka, popis, datum zahájení a ukončení zakázky. Dále můžeme k zakázce přidat položky ze skladu, které jsou k zakázce potřeba. Stačí si najít číslo položky, zadat počet a kliknout na Add item. Položka se poté objeví v seznamu. Pokud položka s takovým číslem neexistuje nebo potřebný počet položek není na skladě, systém na to uživatele upozorní a akci zamítne. Tlačítkem remove vedle položky ji můžeme ze seznamu odstranit. Pod seznamem položek je celková cena a tlačítko Invoice. Tím můžeme vygenerovat fakturu zakázky. Vlevo obrazovky se nachází tlačítka Save, Cancel, Finish order a Take order. Save zakázku uloží, ale pokud je něco ve formuláři vyplněno špatně, systém na to upozorní a zakázku neuloží. Cancel změnu zruší a vrátí uživatele zpátky na seznam zakázek. Finish order zakázku ukončí a Take order přiřadí k zakázce aktuálně přihlášeného uživatele.

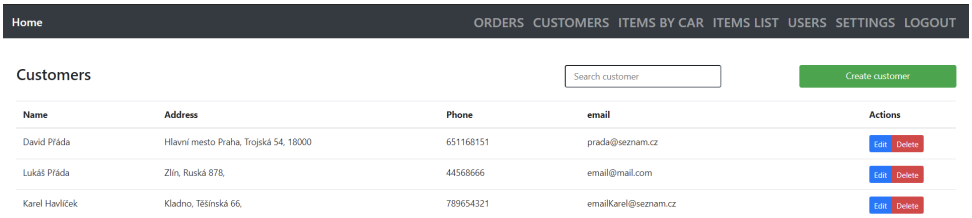
Na stránku pro vytváření zakázek se dostaneme přes seznam zakázek tlačítkem Create order. Funguje podobně jako úprava zakázky, ale neobsahuje funkce pro již vytvořenou zakázku. To znamená, že nejde ukončit, přiřadit zaměstnanci a nemůžeme u ní vygenerovat fakturu.



Obrázek 4.4: Ukázka editace zakázky

4.4 Zákazníci

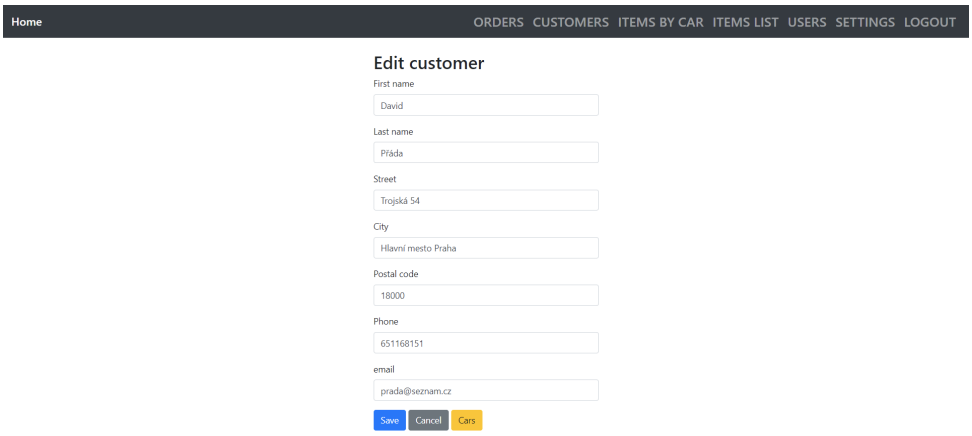
Na stránku se seznamem zákazníků se dostaneme přes tlačítko Customers v horní liště aplikace. U každého zákazníka můžeme vidět jméno, adresu, telefonní číslo a emailovou adresu. Stejně jako u zakázek můžeme zákazníky upravit, mazat a vyhledávat podle všech atributů.



Obrázek 4.5: Ukázka seznamu zákazníků

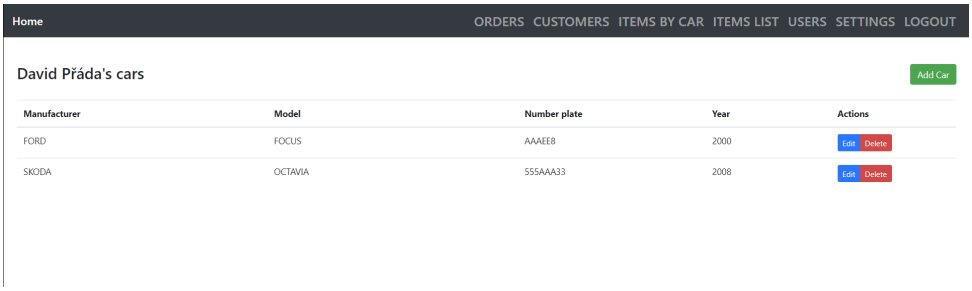
4.4.1 Úprava a vytvoření zákazníka

Pří vytváření zákazníka můžeme zadat jméno, adresu, telefon a email. Jediný povinný atribut je však email, který nám zároveň pomáhá zákazníka jednoznačně identifikovat. Úprava zákazníka funguje stejně jako vytvoření s tím rozdílem, že máme předvyplněný text s detaily zákazníka.



Obrázek 4.6: Ukázka úpravy zákazníka

U úpravy zákazníka je navíc tlačítko Cars, které otevře stránku se všemi auty, které zákazník vlastní.

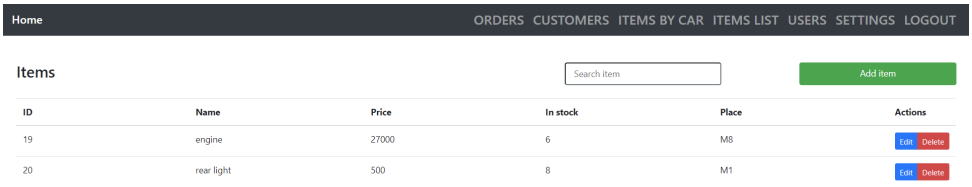


Obrázek 4.7: Ukázka seznamu aut zákazníka

Tyto auta můžeme upravovat a mazat. Pokud k autu vyplní uživatel poznávací značku, která už je uložena u jiného zákazníka, systém uživatele upozorní a auto neuloží.

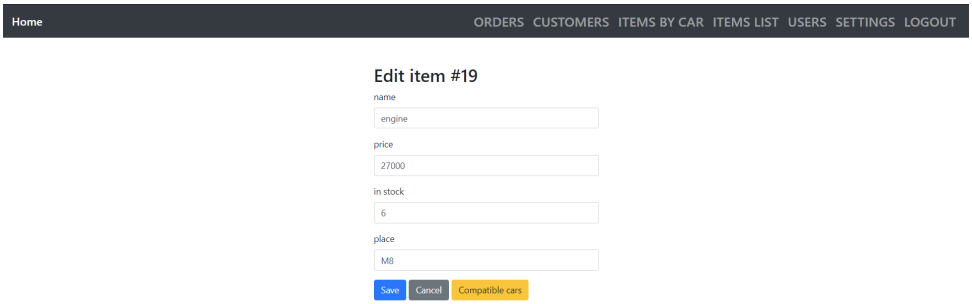
4.5 Sklad

Tlačítko Items list v horní liště aplikace otevře stránku se všemi položkami na skladě. Položky můžeme vyhledávat, upravovat, vytvářet a mazat.



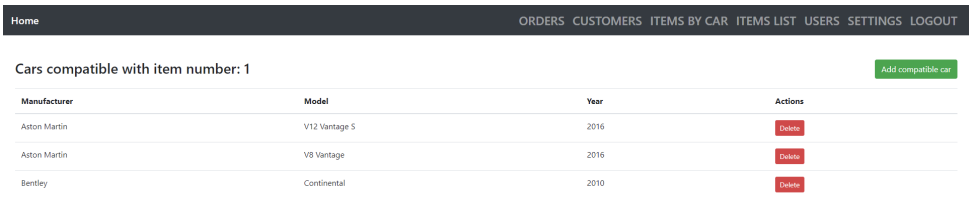
Obrázek 4.8: Ukázka seznamu položek na skladu

Při vytváření nové položky nebo při její úpravě můžeme specifikovat její název, cenu, počet kusů a místo kde se na skladě nachází.



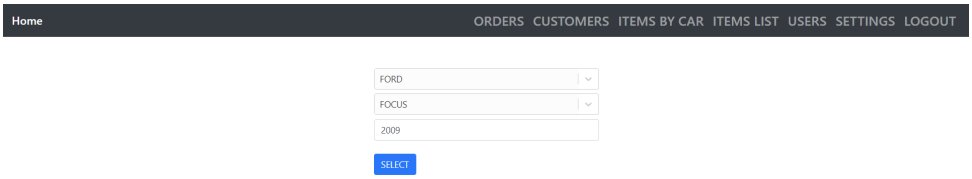
Obrázek 4.9: Ukázka úpravy položky na skladě

Při úpravě položky máme k dispozici také tlačítko Compatible cars, které nás přesune na stránku se seznamem aut, které jsou s danou položkou kompatibilní. Tyto auta můžeme přidávat a mazat.



Obrázek 4.10: Ukázka seznamu aut kompatibilních s danou položkou

Položky můžeme vyhledávat také právě podle typu auta, se kterým jsou kompatibilní. Můžeme tak udělat skrz tlačítko Items by car v horní liště aplikace. Uživatel musí vyplnit celý formulář, pokud tak neudělá, systém ho upozorní.

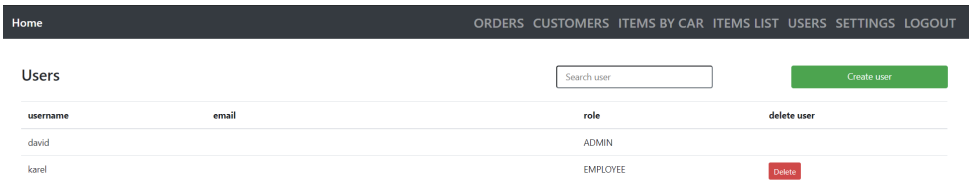


Obrázek 4.11: Ukázka vyhledávání položek podle typu auta

Po zobrazení seznamu položek může uživatel vytvářet nové položky, které se po uložení automaticky nastaví jako kompatibilní s typem automobilu, který byl vyhledán.

4.6 Uživatelé

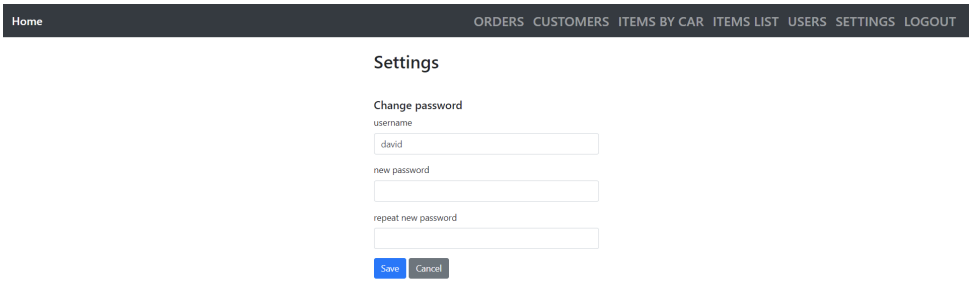
Pokud je uživatel přihlášen jako administrátor nebo manažer, zobrazí se mu v horní liště tlačítko Users. Může tak vytvářet, vyhledávat a mazat uživatele systému. Systém nedovoluje roli manager vytvářet a mazat uživatele s rolí admin. Systém také kontroluje unikátnost uživatelského jména. Pokud se uživatel pokusí vytvořit nového uživatele a jeho jméno už existuje, systém ho na to upozorní.



Obrázek 4.12: Ukázka seznamu uživatelů

4.6.1 Změna hesla

Systém umožňuje změnu hesla pro právě přihlášeného uživatele. Může tak provést přes tlačítko Settings v horní liště. Systém požaduje nové heslo vyplnit dvakrát, aby nedošlo k možnému překliku. Po úspěšné změně hesla je uživatel odhlášen a musí se přihlásit pod novým jménem. Heslo musí obsahovat minimálně 6 znaků.



Obrázek 4.13: Ukázka změny hesla

Kapitola 5

Testování

Nedílnou součástí vývoje softwaru je i testování. Průběžné a pečlivé testování dokáže odhalit problémy dříve než bude pozdě a může tak ušetřit drahocenný čas. Testovat můžeme mnoha způsoby, konkrétně v této práci bylo využito JUnit testů. Testy se zaměřují na dvě části systému, repositories a services, tedy na SQL dotazy, které jsou upraveny pro tento systém a logiku systému. Tyto testy je možné spouštět jednotlivě nebo všechny najednou.

Na testování rest rozhraní byl použitý software Postman. V tomto programu můžeme simulovat HTTP požadavky na náš server a sledovat jejich výsledky. Jde o velmi jednoduchý a rychlý způsob jak naše rozhraní otestovat.

Všechny chyby, které byly průběžným testováním odhaleny, byly odstraněny.

Aplikace byla otestována na prohlížečích Mozzila Firefox, Google chrome, Opera a Microsoft Edge. Na všech těchto prohlížečích fungovala aplikace správně a bez problémů.

Zároveň by bylo vhodné provést testování uživatelského rozhraní samotnými uživateli. Bohužel jsme tak nemohli kvůli pandemické situaci učinit. Proto po konzultaci s vedoucí práce bude testování provedeno později.

Kapitola 6

Závěr

V práci jsem se seznámil s existujícími řešeními jako Jednickanatrhu nebo Mechanic. Díky této rešerši a vlastní zkušenosti s prací v autoservisu jsem sestavil požadavky pro systém.

V analytické častí jsem také porovnal technologie, které bylo možné pro tuto práci použít. Z těchto možností jsem vybral technologie, se kterými jsem měl předchozí zkušenosti, především ze školních předmětů. U Reactu jsem tyto zkušenosti neměl, proto jsem se framework musel naučit od základů. Na pochopení byl React jednoduchý a intuitivní.

Při implementaci backend části projektu jsem využil zkušeností z předmětů Enterprise Architektury a Databázové systémy. Systém byl pravidelně testován pomocí softwaru Postman a pomocí JUnit testů.

Výsledkem práce je funkční prototyp aplikace, který je možné vyzkoušet na adrese `https://davidscarshop.herokuapp.com/`. Systém umožňuje správu zakázek, zákazníků, skladu, rezervací a faktur. Systém je implementován tak, aby ho bylo možné i nadále rozšiřovat o nové funkcionality, jako například generování statistických údajů.

Literatura

[1] *Ing. Pavel Náplava* In: Úvod do předmětu a problematiky Informačních systémů [online] 2019. [cit. 2020-11-30] Dostupné z `https://moodle.fel.cvut.cz/mod/page/view.php?id=111142`.

[2] *jednickanatrhu* In: Program pro autoservisy [online]. [cit. 2020-11-30] Dostupné z `https://jednickanatrhu.cz/program-pro-autoservis`.

[3] *Mechanic* [online]. [cit. 2020-12-1] Dostupné z `https://www.nextis.cz/mechanic/`.

[4] *carsys* In: Software pro prodejce a servisy automobilů [online]. [cit. 2020-12-1] Dostupné z `https://www.carsys.cz/carsystem/`.

[5] *The 10 Best Database Software Systems For Business Professionals* [online] 2017. [cit. 2020-11-30] Dostupné z `https://mytechdecisions.com/it-infrastructure/10-best-database-software-systems-business-professionals/`.

[6] *The history of SQL Server* In: Introduction [online] 2018. [cit. 2020-12-2] Dostupné z `https://www.sqlshack.com/history-sql-server-evolution-sql-server-features/`.

[7] *Postgresql* In: New to PostgreSQL? [online]. [cit. 2020-12-2] Dostupné z `https://www.postgresql.org/` .

[8] *Matyáš Gallas, Informační systém pro správu fyzických dokumentů* [online] 2019. [cit. 2020-12-3] Dostupné z `https://dspace.cvut.cz/handle/10467/88164` .

[9] *Apache Tomcat* [online]. [cit. 2020-12-4] Dostupné z `https://tomcat.apache.org/index.html` .

[10] *Web and Application Servers for Java* [online] 2020. [cit. 2020-12-4] Dostupné z `https://www.baeldung.com/java-servers` .

[11] *Jetty* In: About Jetty [online]. [cit. 2020-12-4] Dostupné z https://www.eclipse.org/jetty/about.html .

[12] *ReactJS* In: What is ReactJS: Introduction To React and Its Features [online] 2020. [cit. 2020-12-7] Dostupné z https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs .

[13] *Best Frontend Frameworks of 2020 for Web Development* [online] 2020. [cit. 2020-12-7] Dostupné z https://www.simform.com/best-frontend-frameworks/ .

[14] *Vuejs* In: Introduction [online]. [cit. 2020-12-7] Dostupné z https://vuejs.org/v2/guide/ .

[15] *Java EE* In: Introduction to Java EE [online]. [cit. 2020-12-1] Dostupné z https://javaee.github.io/tutorial/overview001.html .

[16] *JPA* In: JPA Tutorial [online]. [cit. 2020-12-1] Dostupné z https://www.javatpoint.com/jpa-tutorial .

[17] *Třívrstvá architektura* [online] 2015. [cit. 2020-12-3] Dostupné z https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture .

[18] *David Čápka, MVC architektura* [online]. [cit. 2020-12-3] Dostupné z https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor .

[19] *Axure RP* [online]. [cit. 2020-12-9] Dostupné z https://bit.ly/37GYLys.

[20] *Java Introduction* [online]. [cit. 2020-12-1] Dostupné z https://bit.ly/3h9JkFy .

[21] *Understanding the Basics of Spring vs. Spring Boot* [online] 2018. [cit. 2021-4-8] Dostupné z https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot .

[22] *Spring Framework - Overview* [online]. [cit. 2021-4-8] Dostupné z https://bit.ly/3o2ynXN .

[23] *Tomáš Zajíček, Využití principů IoC/DI v moderních webových aplikacích aneb „dostaneš co potřebuješ“* [online] 2014. [cit. 2021-5-2] Dostupné z http://www.web-integration.info/cs/blog/vyuziti-principu-iocdi-v-modernich-webovych-aplikacich/ .

[24] *Úvod do Spring Boot frameworku pro Javu* [online]. [cit. 2021-4-8] Dostupné z https://www.itnetwork.cz/java/spring-boot/uvod-do-spring-boot-frameworku-pro-javu .

[25] *Introduction* [online]. [cit. 2021-4-20] Dostupné z https://docs.spring.io/spring-security/site/docs/current/reference/html5/ .

[26] *Seznámení s Hibernate ORM* [online] 2014. [cit. 2021-4-8] Dostupné z `https://blog.bcvsolutions.eu/seznameni-s-hibernate-orm/` .

[27] *Maven* [online]. [cit. 2021-5-1] Dostupné z `http://voho.eu/wiki/maven/` .

[28] *Nový program Mechanic pro správu autoservisu* [online] 2019. Dostupné z `https://motofocus.cz/servisni-vybaveni/51352,novy-program-mechanic-pro-spravu-autoservisu`.

[29] *Roger S. Pressman, Software Engineering: A Practitioner's Approach* 1982.

Příloha A

Seznam použitých zkratek

DI Dependency injection

IoC Inversion of Control

HTTP Hypertext Transfer Protocol

IS Informační systém

SQL Structured Query Language

DBMS Database Management System

JPA Java Persistence API

API Application Programming Interface

IDE Integrated Development Environment

ORM Object Relational Mapping

MVC Model View Controller

REST Representational State Transfer

URL Uniform Resource Locator