FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Rozšíření funkcionalit backendu služby sdílení vozidel Uniqway |
| **Student:** | Jan Pospíšil |
| **Vedoucí:** | Ing. Václav Jirovský, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové a softwarové inženýrství, zaměření Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2022/2023 |

## Pokyny pro vypracování

Navrhněte a implementujte backendovou část kontroly vozidel v rámci Uniqway. Při návrhu spolupracujte se studentským týmem Uniqway.
- Seznamte se s architekturou backendu Uniqway.
- Jako podklad využijte předchozí bakalářskou práci obsahující popis kontroly vozidel a implementaci jeho frontendové části.
- Na základě získaných informací navrhněte a implementujte backendovou část kontroly vozidel.
- Doplňte implementaci o unit testy a API testy.
- Zdokumentujte navržené řešení.

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 14. ledna 2022 v Praze.*

Bakalářská práce

# ROZŠÍŘENÍ FUNKCIONALIT BACKENDU SLUŽBY SDÍLENÍ VOZIDEL UNIQWAY

**Jan Pospíšil**

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Václav Jirovský, Ph.D.
5. prosince 2022

Odkaz na tuto práci: Pospíšil Jan. *Rozšíření funkcionalit backendu služby sdílení vozidel Uniqway.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

# Obsah

# Seznam obrázků



# Seznam tabulek

# Seznam výpisů kódu

*Chtěl bych poděkovat především mému vedoucímu práce Ing. Václavu Jirovskému, Ph.D. za cenné rady a konzultace k bakalářské práci. Dále bych rád poděkoval celému týmu Uniqway za příležitost podílet se na jejich projektu. Na závěr bych rád poděkoval mé rodině za jejich podporu po celou dobu studia.*

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona a to na dobu určitou do skončení trvání ochrany dle Smlouvy. Nakládání s předloženou prací se řídí Smlouvou o spolupráci uzavřenou v návaznosti na spolupráci mezi Českým vysokým učením technickým v Praze a společností ŠKODA AUTO a.s. a Smart City Lab s.r.o na výzkumném projektu „CarSharing pro vysokoškolské studenty", uveřejněné v registru smluv na adrese https://smlouvy.gov.cz/smlouva/5973503. Jsem vázán Smlouvou o zachování mlčenlivosti, že nezpřístupním třetí osobě důvěrné informace, které jsem při své práci na Projektu získal.

V Praze dne 5. prosince 2022 .....

# Abstrakt

Tato bakalářská práce se zabývá vývojem rozšíření backendu webové aplikace pro společnost Uniqway o funkcionalitu evidence kontroly vozidel, pneumatik a vybavení vozidla. Práce pokrývá části analýzy aktuálního stavu aplikace, domény a požadavků na rozšíření. Posléze navazuje část návrhu relačního schématu, komunikačního rozhraní a architektury aplikace. Pro samotnou implementaci byl použit framework Play a relační databáze PostgreSQL. Vytvořený kód je pokryt jednotkovými a API testy. Výsledná aplikace poskytuje své služby pomocí REST API.

**Klíčová slova** Uniqway, sdílení vozidel, vývoj webové aplikace, backend, relační databáze, Play framework, PostgreSQL, REST API

# Abstract

The main topic of this bachelor's thesis is the backend expansion of a web application for the Uniqway company with new features providing car check, vehicle equipment, and tire record keeping. This thesis covers analysis of the current application state, domain, and expansion requirements, which are followed by designing a relational database schema, a communication interface, and an application architecture. For implementation, Play Framework and PostgreSQL relational databases were chosen. The code itself is covered by unit and API tests. The resulting application provides its services through the REST API.

**Keywords** Uniqway, car-sharing, web application development, backend, relational database, Play framework, PostgreSQL, REST API

# Seznam zkratek

| | |
|---|---|
| API | Application Programming Interface |
| CRUD | Create, Read, Update, Delete |
| DAO | Data Accesss Object |
| DRY | Don't Repeat Yourself |
| DTO | Data Transfer Object |
| FR | Functional Requirement |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| JPA | Java Persistence Api |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| NFR | Non-Functional Requirement |
| ORM | Object Relational Mapping |
| REST | Representational State Transfer |
| TDD | Test Driven Development |
| URI | Uniform Resource Identifier |

# Kapitola 1
# Úvod

Práce s webovými aplikacemi se dnes již stává každodenní náplní mnoha společností a jejich zaměstnanců. Nejedná se jen o části, které jsou viditelné pro klienty společnosti, ale i o takové funkce, které slouží pro zefektivnění práce zaměstnanců či k automatizaci firemních postupů. S rozvojem společnosti ovšem vznikají i nové požadavky nejen na modernost a použitelnost uživatelského rozhraní, ale i na samotné funkce poskytované serverovou částí takovýchto aplikací.

Zaměření této bakalářské práce je právě vytvoření takového nového rozšíření serverové části webové aplikace pro firmu Uniqway, které poskytne jejím zaměstnancům funkcionalitu evidence kontroly vozidel, evidence stavu vybavení a evidenci pneumatik a jejich výměn. Tyto nové funkce zlepší vedení evidence a zefektivní pracovní postupy při provádění kontrol, zlepší sdílení informací mezi zaměstnanci a zmenší duplicitní činnosti.

Tato práce navazuje na předchozí práci členů Uniqway, zejména však na práci Jiřího Gutwirtha [1], který se zabýval právě uživatelskou částí rozšíření evidence kontrol vozidel.

Počáteční kapitola je věnována volbě metodiky vývoje nového rozšíření a tím i nastínění struktury práce.

První z kapitol práce je věnována analýze, která je prvotní fází realizace rozšíření backednu. Dojde zde k seznámení s používanými technologiemi v Uniqway, provedení analýzy současného stavu aplikace a identifikování požadavků na nové rozšíření.

Následující kapitola je věnována fázi návrhu rozšíření, kde bude nastíněno řešení rozšíření na základě provedené analýzy. Návrh se bude skládat z návrhu datové struktury, komunikačního rozhraní a architektury aplikace.

Po návrhu nám navazuje kapitola implementace, ve které bude popsán postup realizace řešení a zároveň bude vytvořena jeho samotná implementace. Na konci kapitoly bude zhodnoceno plnění požadavků a také uveden postup dokumentace vytvořeného kódu.

Výsledná implementace bude otestována jednotkovými a API testy. Testování bude proto věnována jedna z kapitol, kde budou uvedeny nejen kategorie testů, ale i používané principy, na které při testování narazíme.

# Kapitola 2
# Cíl Práce

Hlavním cílem této bakalářské práce je návrh a následná implementace rozšíření serverové části webové aplikace a její databáze v Uniqway, která poskytuje služby sdílení vozidel. Nové rozšíření poskytne uživatelům funkcionalitu evidence kontroly vozidel, evidence stavu vybavení a evidenci pneumatik a jejich výměn. Výsledné rozšíření vystaví funkcionality přes webové rozhraní.

V teoretické části dojde k popisu technologií používaných v serverové části aplikace Uniqway. Dále dojde k popisu použitých architektonických vzorů, na kterých je aplikace postavena.

Pro dosažení praktického výsledku bude provedena analýza dodaných podkladů, datové domény a funkčních požadavků, které jsou na novou funkcionalitu požadovány ze strany Uniqway. Po provedené analýze proběhne návrh datové struktury v relačním modelu. Dále bude navrženo webové rozhraní pro komunikaci s uživatelskou částí aplikace a bude vybrán vhodný protokol společně s principem komunikace.

Na základě návrhu proběhne implementace řešení, která se skládá z realizace nových funkcí serverové části aplikace a vytvoření nového rozšíření databáze. Serverová část aplikace vystaví webové komunikační rozhraní, které bude poskytovat požadované funkcionality.

Výsledná implementace funkcionalit bude pokryta Unit testy a komunikační rozhraní bude pokryto API testy. K implementaci bude vytvořena dokumentace rozhraní a databáze.

## 2.1 Sestavení požadavků

Pro sestavení cílů a požadavků na rozšíření bylo vycházeno z předchozí práce Jiřího Gutwirtha, z dodaných podkladů a osobních či online konzultací s členy týmu Uniqway, na kterých došlo k objasnění kontextu práce v rámci celé aplikace a byla specifikována data, která je potřeba pomocí nového rozšíření zpracovávat.

# Kapitola 3
# Metodika

V souvislosti s vývojem softwaru byly vytvořeny různé metodiky, které nám poskytují sadu aktivit a postupů vedoucích k vytvoření požadovaného softwaru a definují nám model životního cyklu aplikace, který nám říká kdy a v jakém pořadí jsou jednotlivé úkony vykonávány. V následujících sekcích bude nastíněn přehled běžně používaných metodik a v závěru bude zvolena metodika pro vývoj našeho rozšíření backendu. [2]

## 3.1 Tradiční metodiky

Takovéto metodiky jsou častokrát velice podrobné a formální. Jejich základní předpoklad je, že vývoj softwaru lze dopředu přesně popsat a naplánovat. Z toho vychází i jejich snaha o přesné definování jednotlivých procesů, činností a jejich návaznost.

Tradiční metodiky jsou nejčastěji založené na vodopádovém modelu životního cyklu, který rozdělil vývoj do postupně prováděných fází: specifikace, analýzy, návrhu, implementace, testování a zavedení. Tento model přináší dobré výsledky v případě, že jsou požadavky na software dopředu dobře známy. Nevýhodami jsou například nedostatečné zapojení zákazníka do vývoje a integrace softwaru až po dokončení všech úkolů a požadovaných funkcionalit.

Pro odstranění nevýhod vodopádového modelu se začaly používat metodiky založené na iterativním modelu vývoje, který vytváří software v několika samostatných částech. Do těchto metodik patří například Retional Unified Process. [3, 2]

## 3.2 Agilní metodiky

Tyto metodiky vycházejí z myšlenky, že vývoj softwaru je dynamický proces a v mnoha případech jej nelze přesně dopředu popsat a je tedy nutné se průběžné přizpůsobovat změnám, na které se v průběhu vývoje narazí. Na rozdíl od tradičních metodik jsou agilní metodiky schopny rychleji reagovat na potřeby zákazníka a dodat použitelný software rychleji zejména díky použití iterativního a inkrementálního vývoje, kdy je práce rozdělena do úseků v rámci kterých je software rozšiřován o další funkcionality či je modifikována funkcionalita stávající. Software je tedy zákazníkovi dodáván postupně. [4, 5, 6]

Základní principy agilních metodik byly popsány v Agilním manifestu z roku 2001 [5] a jedná se o čtyři hlavní body popsané v následující tabulce 3.1.

**Tabulka 3.1** Čtyři body agilního manifestu

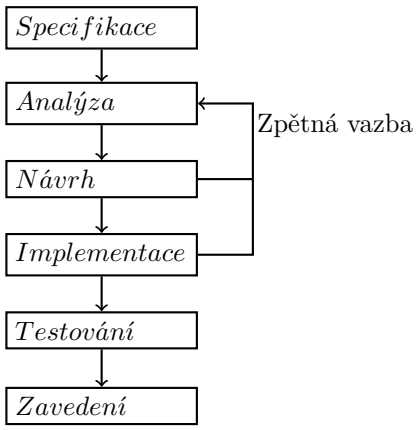
| **Levá strana** | | **Pravá strana** |
|---:|---|---|
| Jednotlivci a interakce | před | procesy a nástroji. |
| Funkční software | před | vyčerpávající dokumentací. |
| Spolupráce se zákazníkem | před | vyjednáváním o smlouvě. |
| Reagování na změny | před | dodržováním plánu. |

I když jsou body na pravé straně hodnotné, těch na levé straně si ceníme více. [5]

Mezi agilní metodiky jsou například zařazeny Scrum, Feature Driven Development či Extreme Programming. [4]

## 3.3 Volba metodiky pro vývoj rozšíření

Jelikož specifikace a všechny požadované funkcionality nového rozšíření jsou dopředu známy a je potřeba je odevzdat jako jeden funkční celek, bude metodika založená na vodopádovém modelu vhodnou volbou. Pro zapracování případné zpětné vazby od Uniqway bude ovšem tento model potřeba modifikovat tak, že se bude možné vracet k předchozím již proběhlým fázím a změny zapracovat. Tento modifikovaný model je vyobrazen na obrázku 3.1.




**Obrázek 3.1** Modifikovaný vodopádový model

# Kapitola 4
# Analýza

Cílem této kapitoly je základní seznámení s aplikací a jejími technologiemi, které se také budou týkat nového rozšíření kontroly vozidel. Dalším cílem je analýza procesů a datového modelu rozšíření, kdy dojde k rozložení řešení na menší uchopitelné části. V poslední části dojde k určení funkčních a nefunkčních požadavků rozšíření.

## 4.1 O webové aplikaci v Uniqway

Aplikace používaná v Uniqway je rozdělena na dva hlavní celky. První celek aplikace je backend s databází běžící na serveru a druhý celek je frontend, který pod sebou zaštiťuje klientské webové stránky a mobilní aplikace.

Jedná se o rozdělení klient-server, kdy tyto dva celky spolu komunikují. Klient je taková část, která vyžaduje po serveru provedení nějaké služby či funkcionality. Server je na druhou stranu takový celek, který požadované služby klientům poskytuje. Jeden program může vystupovat jak v roli klienta, tak i v roli serveru. Klient-server architektura se nejčastěji aplikuje ve spojení s třívrstvou architekturou softwaru, kdy se funkce aplikace rozdělují, jak je popsáno v [7], do tří základních skupin:

**Datové funkce** : jedná se o ukládání, čtení a úpravu dat. V případě aplikace v Uniqway jsou tyto funkce poskytovány PostgreSQL databází.

**Věcně orientované funkce** : zajišťují hlavní byznys logiku aplikace. Jejich úkolem je transformace vstupních dat na výstupní a další zpracování. Tyto funkce poskytuje backend webové aplikace.

**Komunikační funkce** : zahrnují komunikaci s uživatelem, zobrazování dat v požadovaném formátu, navigaci uživatele po funkcích aplikace a další. V našem případě nám tyto funkce poskytuje frontend.

Výhodou takového rozdělení je nezávislost uživatelského rozhraní, které může být v budoucnu vyměněno či upraveno s příchodem nových technologii bez nutnosti změnit část backendu. Dalším benefitem je znovupoužitelnost backendu pro více klientských aplikací, které mohou využívat stejné API.

### 4.1.1 Backend

Jako backend označujeme tu část webové aplikace, která slouží k administraci webu a ke zpracování dat. [8] Je zde napsána hlavní pracovní logika aplikace, která přijímá požadavky, zpra-

covává data a ukládá je do relační databáze. V případě nově plánovaného rozšíření tomu nebude jinak. Jeho hlavním úkolem bude zpracování dat o kontrolách vozidel, defektech a evidence výměn pneumatik spolu s dalšími úkoly.

Důležitou částí backendu je, aby poskytoval přesně definované API (Application Program Interface) pro klienty a uživatele, díky kterému můžou s backendem interagovat. Základem každého API, neboli programové rozhraní, je sada postupů, protokolů a nástrojů, které nám definují jakým způsobem mají jednotlivé softwarové komponenty mezi sebou interagovat. [9]

#### 4.1.1.1 Framework Play

V Uniqway byl pro backend zvolen framework Play napsaný v jazyce Java a Scala, který je často používaný k vytváření webových aplikací založených na architektuře klient-server s komunikačním protokolem HTTP (Hypertext Transfer Protocol). Klienti tedy používají HTTP žádosti pro volání funkcí a interakci se serverem, který jejich žádost zpracuje a pošle zpátky výslednou odpověď. Aplikace může pro svou činnost využívat další služby či databáze.

Funkce serveru jsou klientům poskytovány jako takzvané akce, které jsou logicky seskupeny do kontrolérů. HTTP žádosti jsou zpracovány pomocí routeru, který volá příslušnou akci podle zadané URL a HTTP metody. Toto mapování je však nutné nakonfigurovat.

Play obsahuje velkou řadu komponent, které usnadňují vytváření REST bezstavových webových služeb. Dále přichází již s vestavěnými ORM (Object Relational Mapping), jako Anorm, Slick a JPA (Java Persistence API), což velice usnadňuje práci s databázemi. [10, 11]

#### 4.1.1.2 Java

Java je kolekce softwarových balíčků a specifikací pro stejnojmenný objektově orientovaný programovací jazyk z roku 1995, který je velice populární k vytváření aplikací kompatibilních s více platformami. Přenositelnost je jedna z hlavních výhod, které Java přináší a je možné ji používat na velkém množství operačních systému, včetně Windows, MacOS a Linux. Od jejího vzniku prošel jazyk mnoha vylepšeními a verzemi. [12] Pro nás je ovšem důležitá Java verze 11, která bude použita při implementaci nového rozšíření.

#### 4.1.1.3 Object Relational Mapping

ORM (objektově relační mapování nebo object relational mapping) je soubor funkcionalit, které vytvářejí a udržují vztah mezi objekty a záznamy v databázi. Poskytuje nám možnost provádět různé databázové operace jako je čtení, zápis, úprava a další. Velkou výhodou je pro programátora odstínění od databáze, kdy nepracuje s tabulkami ani sloupci, ale pracuje pouze s mapovanými objekty a tedy nepotřebuje psát kód SQL pro manipulaci s daty, jelikož o jeho generování se stará samotné ORM. [13]

Java poskytuje rozhraní pro ORM zvané jako Java Persisence API (zkráceně JPA), které nám poskytuje specifikaci pro mapování Java objektů do relační databáze. Takto mapované objekty nazveme entitami. JPA nám samo o sobě nestačí pro vykonávání mapování, jelikož se jedná pouze o specifikaci a je nutná konkrétní implementace daného API a může se jednat například o ORM nástroj Hibernate, který je v Uniqway používán. [14]

#### 4.1.1.4 Databáze PostgreSQL

Pro uchování dat byl v Uniqway zvolen open-source (projekt s otevřeným zdrojovým kódem) objektově-relační databázový systém PostgreSQL, který využívá a v určitých směrech expanduje velice rozšířený jazyk SQL, který je určený pro práci se samotnou databází i se zpracovávanými daty v databázi. PostgreSQL pokrývá téměř celou specifikaci SQL:2016. Jedná se o dlouhodobě používaný a spolehlivý systém, který je neustále aktualizován. [15]

V Uniqway je navíc použito rozšíření PostGIS, které přidává podporu pro geografické objekty. Zahrnuje například geometrické typy, prostorové funkce, operátory a indexy pro tyto typy.[16]

### 4.1.2 Architektura backendu

Následující sekce je věnována použitým vzorů na straně backendu, které budeme chtít zohlednit v pozdější implementaci rozšíření.

Jedním z použitých vzorů na backendu je DAO (Data Accesss Object) vzor. DAO tvoří vrstvu mezi naší logikou a přistupovanými daty. Tyto objekty mají za úkol číst, zapisovat či modifikovat data v databázi. Obecně nám poskytují abstrakci a zapouzdření všech přístupů k datovému zdroji a poskytují nám rozhraní nezávislé na technologii použitou k ukládání samotných dat. [17] Zde ovšem objekty DAO nepřistupují do databáze přímo, ale využívají funkcí, které jim poskytuje použité ORM.

Dalším použitým vzorem je DTO (Data Transfer Object) vzor, který na druhou stranu slouží ke zjednodušení komunikace mezi klientem a serverem a poskytuje vrstvu oddělení mezi daty používaných na serveru od klientů. DTO jsou tedy nosičem dat, které nám poskytují zapouzdření a snadnější serializaci dat za účelem přenosu. [18]

Předchozí vzory nám rovněž ukazují na použití třívrstvé architektury v rámci samotného backendu, který tedy lze rozdělit na následující vrstvy, jak jsou zmíněny v [7], a tvoří je:

**Datová vrstva** : Tato vrstva obsahuje objekty DAO a třídy mapující objekty na entity uložené v relační databázi.

**Komunikační vrstva** : Tato vrstva je tvořena kontroléry a akcemi, které nám poskytují koncové body webového API. Dále obsahuje objekty DTO.

**Vrstva služeb** : Tato vrstva obsahuje hlavní logiku aplikace a poskytuje funkce komunikační vrstvě a využívá funkcí datové vrstvy.

### 4.1.3 Frontend

Za frontend považujeme takové části webové aplikace, které jsou viditelné uživatelům a častokrát se jedná se o oddělenou část webové aplikace. [19]. V případě Uniqway se jedná o samostatné softwarové aplikace, které komunikují s backendem přes definované API, jedná se například o mobilní aplikace či webové aplikace určené zákazníkům a administrátorům. V posledním zmíněném případě se nalézá rozšíření frontendu kontroly vozidel, pro které bude vytvořeno na backendu nové rozšíření a API.

## 4.2 Popis procesů

### 4.2.1 Evidence defektu

V případě nalezení defektu na vozidlu se provádí jeho evidence formou slovního popisu, fotografie, typem defektu, závažností a jeho hierarchickou lokací na vozidle navrženou v práci [1]. Defekt je nejčastěji evidován při kontrole interiéru a exteriéru vozidla, ale může být uživatelem evidován i bez provedené kontroly.

V případě, že došlo k opravě defektu, je záznam aktualizován a defekt se již nebere jako aktuální.

### 4.2.2 Evidence kontroly vozidla

Kontrolu vozidla provádí osoba označovaná v Uniqway jako ambasador. Probíhá kontrola interiéru, exteriéru, vybavení a technického stavu vozu. Pořadí jednotlivých kontrol provádí ambasador podle své volby.

Ambasador vytváří nový záznam o kontrole a eviduje nalezené defekty vozidla. Dále zaznamenává informace o funkčnosti jednotlivých druhů světel vozidla. Další část kontroly je vybavení vozu, kdy ambasador zkontroluje přítomnost a stav vybavení a eviduje, zdali je každý kus v pořádku a v případě problému s vybavením zadává popis problému a problémové vybavení označí. Poslední část kontroly se týká technického stavu, kdy je evidováno, že vozidlo obsahuje dostatek motorového oleje, chladící kapaliny a vody do ostřikovačů. Provozní kapaliny navíc obsahují použitý typ oleje a typ chladící kapaliny. Závěrečnou částí technického stavu je kontrola běhu motoru a aktivita kontrolek na palubní desce a potvrzení, že vozidlo je způsobilé k jízdě.

Speciálním případem kontroly vozidla je provedení čištění, kdy dochází k čistění vozidla ozonem, mytí oken, mytí exteriéru, čištění interiéru, luxování či mokrému čištění sedaček. Při této kontrole může být navíc provedena jakákoliv dílčí část běžné kontroly vozidla. Nejčastěji se jedná o kontrolu interiéru a vybavení.

Kontrola vozidla nemusí být evidována najednou, je možné se k poslední nedokončené kontrole vrátit, editovat ji, či ji smazat dokud není ukončena.

### 4.2.3 Evidence výměny pneumatik

Při přezouvání vozidla se vytváří záznam o výměně, kdy se eviduje stav ujetých kilometrů zobrazených na tachometru vozidla k danému dni, dále jaký druh výměny byl proveden (celé kolo, jen pneumatika, nové kolo a další) a jakého typu jsou nově nasazené a sundané ráfky. U sundané sady pneumatik se změří hloubka zbývajícího dezénu jednotlivých pneumatik, zaznamená se pozice, kde byla pneumatika na vozidle namontována, aktuální hmotnost pneumatiky a údaje se poté uloží. U samotných pneumatik se uvádí značka výrobce, DOT (Department of Transport) kód, rychlostní index (kód + maximální rychlost), hmotnostní index (kód + nosnost) a její rozměr. Tyto údaje můžou být využity při další výměně.

## 4.3 Požadavky na rozšíření

Po analýze aktuálního stavu aplikace a seznámením se s procesy a požadavky na rozšíření přichází správný čas na přesnější určení funkcionalit aplikace, které od nového rozšíření očekáváme.

Cílem tedy je přesně specifikovat jaké funkce bude nové rozšíření systému poskytovat klientům a také v jakém rozsahu. K tomuto účelu skvěle poslouží určení funkčních požadavků, které nám nejen přesně specifikují, jaké nové funkcionality budeme po rozšíření požadovat, ale zároveň odstraní případné nejasnosti a upřesní hranice našeho rozšíření.

Druhou částí je i určení nefunkčních požadavků rozšíření, které nepopisují přímo přidávané funkcionality, ale určují jaké mají mít další vlastnosti a chování. [20]

### 4.3.1 Funkční požadavky

#### 4.3.1.1 FR1 Evidence kontrol vozidel

Systém bude umožňovat autorizovanému uživateli provádět evidenci kontrol vozidel. U každé kontroly bude zaznamenáno vozidlo, kterého se daná kontrola týká, uživatel (v Uniqway je používán termín ambasador), který kontrolu provedl. Dále bude evidováno datum vytvoření a dokončení kontroly, typ provedené kontroly, hladina a druh oleje, hladina a druh chladící kapaliny, hladina vody do ostřikovačů, funkčnost světel vozidla, jejich typ a pozice na vozidle, schopnost

startu vozidla, informace o svítících kontrolkách, případné defekty na vozidle a potvrzení celkové způsobilosti vozidla k jízdě. Součástí evidence kontroly je rovněž zaznamenání stavu vybavení vozidla.

Systém umožní ke kontrole vozidla evidovat i provedené čištění vozidla. Mezi evidované údaje patří záznam o provedení desinfekce ozonem, mytí exteriéru, mytí oken, čištění interiéru, luxování a hloubkové čištění sedaček.

#### 4.3.1.2 FR2 Poskytnutí informací ke kontrole vozidla

Systém poskytne autorizovanému uživateli doplňující informace ke kontrole vozidla. Mezi takové informace budou zařazeny typy kontrol vozidel, hladiny kapaliny a typy světel a jejich umístění.

#### 4.3.1.3 FR3 Poskytnutí detailu konkrétní kontroly vozidla

Systém umožní autorizovanému uživateli získat detailní informace o konkrétní kontrole vozidla.

#### 4.3.1.4 FR4 Poskytnutí seznamu kontrol vozidla

Systém umožní získat autorizovanému uživateli seznam všech provedených i probíhajících kontrol konkrétního vozidla.

#### 4.3.1.5 FR5 Modifikace neukončené kontroly vozidla

Systém umožní autorizovanému uživateli modifikovat a případně odstranit pouze neukončenou kontrolu vozidla. Ukončené kontroly už nebude možné upravovat, bude je možné pouze číst. Smazání neukončené kontroly bude mít za důsledek i smazání evidovaných defektů zaznamenaných během dané kontroly.

#### 4.3.1.6 FR6 Evidence defektů

Systém bude umožňovat vytváření a zobrazení informací o jednotlivých defektech vozidla. Defekt může být zaznamenán ke konkrétní kontrole vozidla, nebo pouze k danému vozidlu i bez kontroly. U defektu se bude dále evidovat druh a závažnost defektu, lokace na vozidle, datum zaznamenání, datum opravy defektu, jeho fotografie a slovní popis.

#### 4.3.1.7 FR7 Poskytnutí informací k defektu

Systém poskytne autorizovaným uživatelům doplňující informace k defektu. Mezi takové informace budou patřit možné lokace, kde se může defekt nalézat, druhy defektů a jejich závažnost.

#### 4.3.1.8 FR8 Poskytnutí detailu konkrétního defektu

Systém umožní autorizovanému uživateli získat detailní informace ke konkrétnímu defektu.

#### 4.3.1.9 FR9 Poskytnutí seznamu všech platných defektů

Systém umožní autorizovanému uživateli získat seznam všech platných, tedy neopravených, defektů na konkrétním vozidle.

#### 4.3.1.10 FR10 Poskytnutí seznamu všech defektů

Systém umožní autorizovanému uživateli získat seznam všech defektů na konkrétním vozidle, které kdy byly na vozidle nalezeny.

#### 4.3.1.11 FR11 Modifikace evidovaného defektu

Systém umožní autorizovanému uživateli modifikovat evidované informace o defektu. Modifikace a smazání defektu budu možné v případě, že je navázán k nějaké poslední nedokončené kontrole vozidla či je veden přímo u daného vozidla. Nebude možné vymazat defekt u ukončené kontroly vozidla, zde bude možné pouze modifikovat jeho stav opravy a popis.

#### 4.3.1.12 FR12 Evidence vybavení vozidla

Systém bude evidovat vybavení konkrétního vozidla. Mezi zaznamenávané údaje vybavení budou zařazeny indikátory stavu vybavení a zdali je v pořádku. Dále bude zaznamenán typ a kategorie vybavení, jeho název, popis a doplňující hodnota. Ke každému vybavení bude moct uživatel přidat doplňující poznámky, kde bude uveden text a datum vytvoření poznámky a aktuální stav vybavení. Vytvořený záznam poznámky může být navázán na kontrolu vozidla. Funkce evidence vybavení bude dostupná pouze autorizovaným uživatelům.

#### 4.3.1.13 FR13 Poskytnutí seznamu vybavení vozidla

Systém umožní autorizovanému uživateli získat seznam a informace o vybavení konkrétního vozidla.

#### 4.3.1.14 FR14 Modifikace stavu vybavení vozidla

Sytém umožní autorizovanému uživateli modifikovat stav vybavení ve vozidle a přidávat k němu další poznámky.

#### 4.3.1.15 FR15 Evidence výměny kol

Systém umožní autorizovanému uživateli evidovat výměnu sady pneumatik za jinou sadu. Mezi zaznamenávané údaje výměny patří datum provedení výměny, záznam staré sady, záznam nové sady, aktuální stav tachometru vozidla, typ sundaných a nasazených ráfků. Dále se eviduje typ provedené výměny a typ výměny kol (například výměna celého kola, jen pneumatiky, jen ráfek…). U sundaných pneumatik se zaznamenává hloubka zbývajícího vzorku, její navážená hmotnost a pozice kola, na kterém byla pneumatika namontována.

#### 4.3.1.16 FR16 Poskytnutí seznamu předchozích výměn kol

Systém umožní autorizovanému uživateli získat záznamy předchozích výměn a informace o vyměněných sadách pneumatik včetně jejich naměřených vzorků a hmotností.

#### 4.3.1.17 FR17 Evidence pneumatik

Systém bude evidovat informace o pneumatikách. Mezi zaznamenávané údaje o pneumatikách bude patřit jejich průměr, výška profilu, šířka a DOT kód. Další informace, které se budou evidovat, budou obsahovat typ konstrukce pneumatiky, druh pneumatiky (zimní, letní…), váhový index, rychlostní index a jméno výrobce. Ke každé pneumatice se bude evidovat v rámci přezouvání hloubka zbývajícího vzorku a její hmotnost. Funkce bude dostupná autorizovaným uživatelům.

### 4.3.2 Nefunkční požadavky

#### 4.3.2.1 NFR1 Dostupnost přes webové rozhraní

Rozšíření backendu bude dostupné přes webové rozhraní, tedy pro klienty bude postačovat pro využití funkcionalit jakýkoliv systém disponující připojením k internetu.

#### 4.3.2.2 NFR2 Lokalizace

Rozšíření backendu umožní poskytnout u výčtových entit popisy v českém a anglickém jazyce.

#### 4.3.2.3 NFR3 Volba technologie

V průběhu návrhu budou zohledněny technologie, které již firma Uniqway aktuálně využívá. Nové navazující rozšíření je bude využívat. Jedná se o framework Play a databázi PostgreSQL.

#### 4.3.2.4 NFR4 Ověření uživatelé

Služba, kterou dané rozšíření poskytne, bude dostupná pouze ověřeným uživatelům.

#### 4.3.2.5 NFR5 Nezavádění nových technologii

Realizované rozšíření nepřidá ke stávajícím technologiím žádné nové.

# Kapitola 5
# Návrh

Tato kapitola se zabývá návrhem nového rozšíření podle provedené analýzy v předchozí kapitole. Nalézá se zde návrh datového modelu pro relační databázi, který vychází z analyzované domény. Dále se zde nalézá návrh komunikačního rozhraní aplikace spolu s návrhem architektury rozšíření.

## 5.1 Analýza a návrh domény

Tato část se zabývá analyzovanou doménou a popisem entit, které se v ní nalézají. Entita je jakýkoliv fyzický či abstraktní objekt, se kterým potřebujeme pracovat.

Hlavní myšlenkou tohoto popisu je rozložení domény na menší části, které budou lépe pochopitelné a budeme s nimi moct snáze zacházet. Výsledkem tohoto rozkladu je konceptuální doménový model vyobrazen pomocí diagramu tříd. Tento model zachycuje entity, se kterými potřebujeme pracovat. Dále vyobrazuje i vztahy, které mezi sebou entity mají, pomocí asociativních a hierarchických vztahů. Model rovněž obsahuje informace o tom, jaká data i atributy jednotlivé entity v sobě uchovávají.

Jedná se o platformě nezávislý model, je tedy odstíněn od implementačních detailů, které se můžou lišit podle použité technologie. Získané informace později budou použity při vytváření datového modelu, v průběhu modelování databáze a následné realizaci rozšíření na backendu. [7]

### 5.1.1 Lokalizace dat

Jedním z požadavků na rozšíření je umožnit ukládání některých údajů v českém a anglickém jazyce. V průběhu návrhu byly uvažovány různé způsoby uložení. V následujících sekcích budou tyto varianty blíže popsány.

#### 5.1.1.1 Samostatné atributy

Tato varianta staví na principu ukládání údajů v různých jazycích společně s každou entitou. Pro každý údaj jsou v entitě vytvořeny takové atributy, které ukládají překlad údaje pro každý potřebný jazyk.

Benefitem tohoto řešení je snadná práce s překlady, jelikož jsou uložené společně s entitou, ale nevýhodou je nutnost přidávání dalších atributů v případě, že bude potřeba přidat překlady pro další jazyk.

#### 5.1.1.2 Lokalizované záznamy

Překlady pro jednotlivé jazyky jsou v této variantě ukládány pomocí více záznamů jedné entity, kde každý záznam reprezentuje jeden překlad. Entita navíc obsahuje indikátor jazyka pro jeho identifikaci.

Výhodou takového řešení je, že není nutné modifikovat strukturu při zavedení dalšího jazyka, ale nastává problém s udržováním konzistence a hledáním všech překladů, které reprezentují jednu entitu více záznamy.

#### 5.1.1.3 Překladová tabulka

Poslední uvažovanou variantou bylo použití překladové tabulky, která by obsahovala seznam všech použitých výrazů a jejich překlady. Jedná se o vícejazyčný slovník.

Hlavní výhodou takového řešení je, že jsou všechny překlady uloženy na jednom místě a v případě jejich modifikace se změny projeví ve všech entitách, které daný překlad využívají a tedy je snížen počet duplicitních překladů. Nevýhodou je samozřejmě nutnost vytvoření další datové struktury a tedy i logiky pro získávání překladů.

#### 5.1.1.4 Zvolená varianta

Pro realizaci lokalizace dat byla zvolena varianta s použitím samostatných atributů z důvodu jejich snadné manipulace a potřeby evidovat pouze dva jazyky.

Lokalizované záznamy nebyly zvoleny z důvodu komplikované evidence a nutnosti udržovat více záznamů k jedné entitě.

Překladová tabulka by byla více komplikovaným řešením, které bohužel v tomto případě nepřináší hlavní benefit redukce duplicit, jelikož se ve velkém množství případů jedná o více slov v jednom sloupci a povaha ukládaných textů je velice různorodá.

### 5.1.2 Kontrola vozidla

V této podkapitole je ukázána část doménového modelu, která je vyobrazena na obrázku 5.1, popisující evidenci kontrol vozidla a evidenci defektů. Celý model se nalézá v příloze A.

#### User

Entita představující uživatele, který provádí kontrolu vozidla.

#### Car

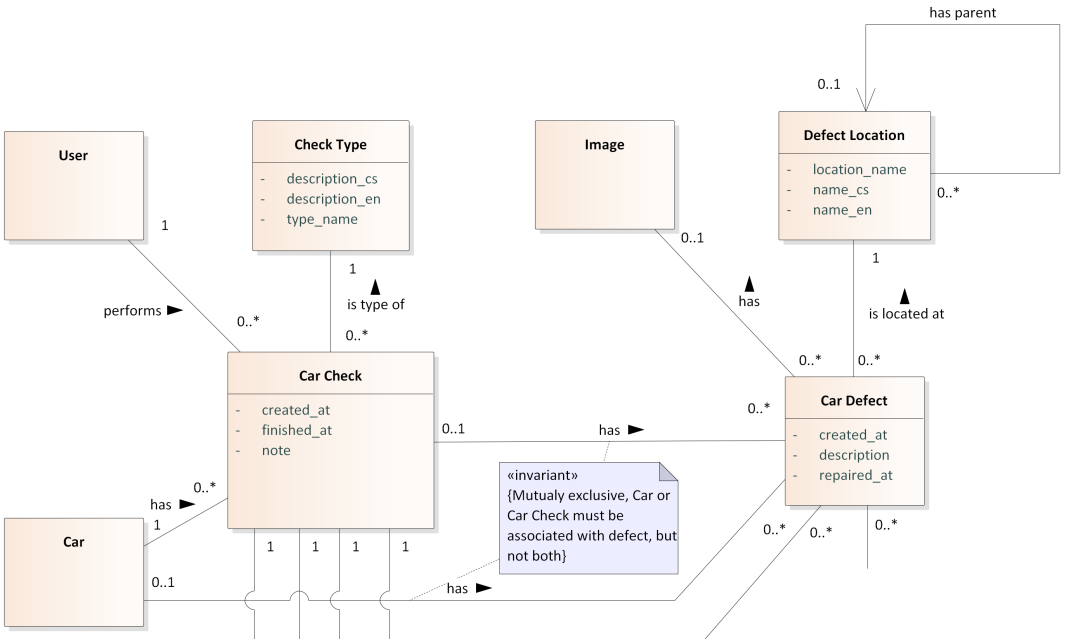
Entita představující vozidlo, na kterém je prováděna kontrola.

#### Car Check

Kontrola vozidla provedená ambasadorem.

**Tabulka 5.1** Popis atributů entity Car Check

| Název atributu | Popis |
|---|---|
| created_at | Datum a čas vytvoření kontroly |
| finished_at | Datum a čas dokončení kontroly |
| note | Poznámka ke kontrole |




**Obrázek 5.1** Část doménového modelu evidence kontroly vozidla

## Check Type

Druh provedené kontroly vozidla.

**Tabulka 5.2** Popis atributů entity Check Type

| Název atributu | Popis |
|---|---|
| type_name | Název druhu kontroly |
| description_cs | Popis druhu prohlídky v českém jazyce |
| description_en | Popis druhu prohlídky v anglickém jazyce |

## Car Defect

Entita záznamu nalezeného defektu vozidla.

**Tabulka 5.3** Popis atributů entity Car Defect

| Název atributu | Popis |
|---|---|
| created_at | Datum a čas vytvoření záznamu o defektu |
| repaired_at | Datum a čas opravy defektu |
| description | Popis defektu |

## Defect Location

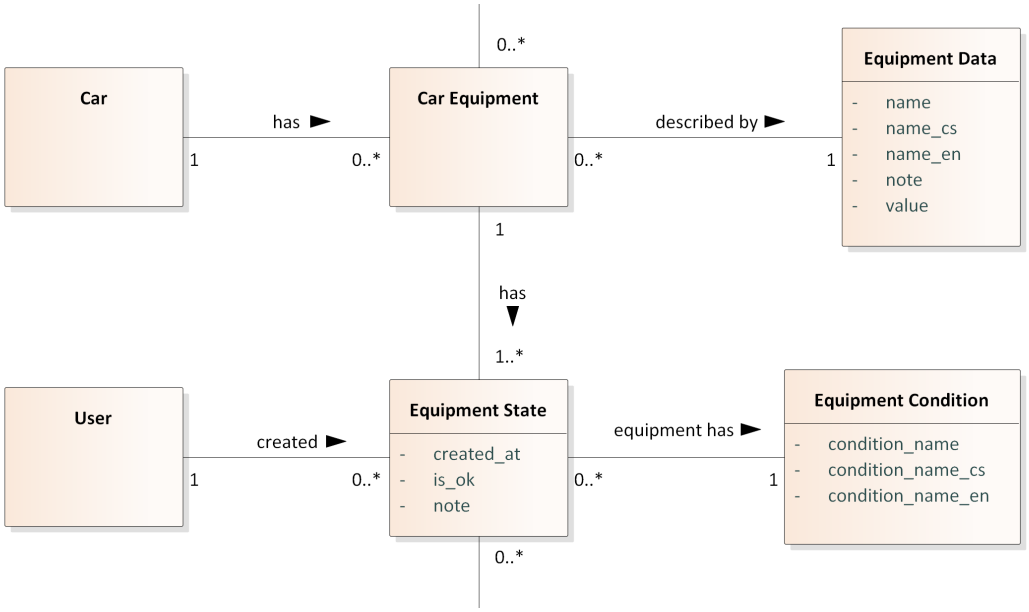
Lokace defektu určuje, kde se na vozidle defekt nalézá.

**Tabulka 5.4** Popis atributů entity Defect Location

| Název atributu | Popis |
|---|---|
| location_name | Název lokace |
| description_cs | Popis lokace v českém jazyce |
| description_en | Popis lokace v anglickém jazyce |

### 5.1.3 Vybavení

V této sekci se nalézá část doménového modelu, která je vyobrazena na obrázku 5.2, související s evidencí vybavení vozidel. Celý model se nalézá v příloze B.




**Obrázek 5.2** Část doménového modelu evidence vybavení

#### Car Equipment

Entita reprezentující vybavení ve vozidle.

#### Equipment State

Stav vybavení vozidla.

**Tabulka 5.5** Popis atributů entity Equipment State

| Název atributu | Popis |
|---|---|
| created_at | Datum a čas vytvoření záznamu o stavu |
| is_ok | Indikuje, zdali je vybavení v pořádku |
| note | Poznámka ke stavu vybavení |

#### Equipment Data

Tato entita uchovává data k danému vybavení vozidla.

**Tabulka 5.6** Popis atributů entity Equipment Data

| Název atributu | Popis |
|---|---|
| name | Název vybavení |
| name_cs | Název vybavení v českém jazyce |
| name_en | Název vybavení v anglickém jazyce |
| note | Poznámka k vybavení |
| value | Dodatečná unikátní hodnota/kód vybavení |

### Equipment Condition

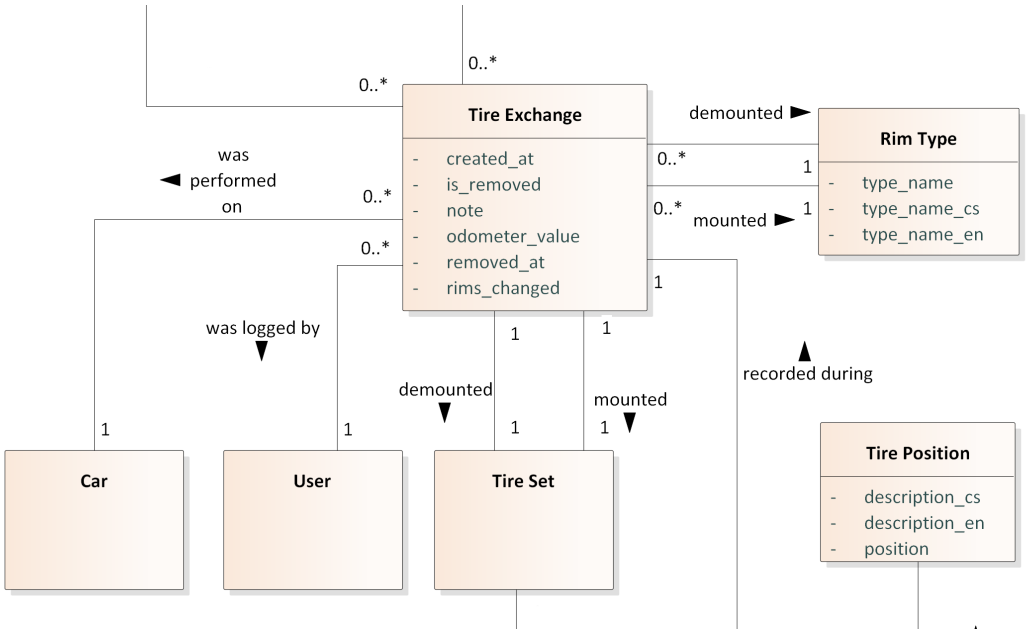
Kondice vybavení určuje, jaký je fyzický stav vybavení.

**Tabulka 5.7** Popis atributů entity Equipment Condition

| Název atributu | Popis |
|---|---|
| condition_name | Název stavu vybavení |
| condition_name_cs | Název stavu vybavení v českém jazyce |
| condition_name_en | Název stavu vybavení v anglickém jazyce |

## 5.1.4 Výměna pneumatik

Tato podkapitola obsahuje částečný doménový model entit, který je vyobrazen na obrázku 5.3, v rámci evidence pneumatik a evidencí jejich výměn. Celý model se nalézá v příloze C.




**Obrázek 5.3** Část doménového modelu evidence výměny pneumatik

### Tire Exchange

Výměna pneumatik provedená na vozidle.

**Tabulka 5.8** Popis atributů entity Tire Exchange

| Název atributu | Popis |
|---|---|
| created_at | Datum a čas provedení výměny |
| is_removed | Indikátor smazaného stavu |
| note | Poznámka k provedené výměně |
| odometer_value | Záznam stavu tachometru v čase výměny |
| removed_at | Záznam času smazání |
| rims_changed | Indikuje stav, zdali byly změněny ráfky během výměny |

### Tire Set

Entita reprezentující sadu pneumatik, která může být na vozidlo namontována.

### Rim Type

Entita zachycující typ ráfku.

**Tabulka 5.9** Popis atributů entity Rim Type

| Název atributu | Popis |
|---|---|
| type_name | Název typu ráfku |
| type_name_cs | Název typu ráfku v českém jazyce |
| type_name_en | Název typu ráfku v anglickém jazyce |

### Exchange Record

Entita záznamu vytvořeného při sundání pneumatiky v průběhu výměny.

**Tabulka 5.10** Popis atributů entity Exchange Record

| Název atributu | Popis |
|---|---|
| pattern_depth | Hloubka dezénu naměřená při sundání pneumatiky |
| weight | Váha pneumatiky změřená při sundání pneumatiky |

### Tire Position

Entita zachycující pozici pneumatiky na vozidle.

**Tabulka 5.11** Popis atributů entity Tire Position

| Název atributu | Popis |
|---|---|
| description_cs | Popis pozice pneumatiky v českém jazyce |
| description_en | Popis pozice pneumatiky v anglickém jazyce |
| position | Název pozice pneumatiky |

## 5.2 Datový model

Datové modely vychází z doménových modelů, které byly vytvořeny v rámci předchozí analýzy a návrhu. Jedná se o modely, které nám již popisují jednotlivé entity tak, jak budou ukládány do relační databáze. Datové modely tvoří dva druhy modelů. První model, který nazveme konceptuální, nám popisuje vazby a atributy jednotlivých entit a jejich vlastnosti. Druhým modelem

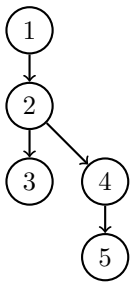
je model relační, který již detailně popisuje konkrétní pojmenování klíčů a vazeb, tak jak budou realizovány v databázi a jednotlivé datové typy atributů. Jelikož tyto modely se velice podobají modelům doménovým, nebudou zde uváděny a budou k nahlédnutí v příloze. Model kontroly vozidel se nalézá v příloze D, model vybavení v příloze E a model výměny pneumatik v příloze F. Dále zde budou popsány pouze vybrané principy použité při řešení.

### 5.2.1 Návrh ukládání hierarchické struktury do relační databáze

V návrhu rozšíření se vyskytuje potřeba ukládat hierarchická data pro určení lokace defektu na vozidle. Ukládání hierarchických dat do relační databáze není triviální proces, protože relační databáze v základu nemají podporu pro hierarchické typy. Je tedy nutné navrhnout datovou strukturu, která nám umožní hierarchii ukládat, ale také procházet a vyhledávat. Následující sekce se budou konkrétně zabývat možnými způsoby uložení stromové hierarchické struktury pomocí relační databáze. Dále budou porovnány výhody a nevýhody jednotlivých způsobů.

Hierarchie bude v následujících příkladech reprezentována za pomocí zakořeněného stromu a tedy všechny hrany mezi uzly vedou směrem od kořene. Graf stromu, který budeme v následujících příkladech reprezentovat v databázi, je ukázán na obrázku 5.4.

Příklad používá dva základní vztahy mezi uzly, které budou podrobněji popsány. První vztah je předchůdce-následník, kdy libovolný uzel na cestě od kořene do uzlu A se nazývá předchůdce uzlu A a libovolný uzel ležící na cestě od uzlu A do listu stromu se nazývá následník uzlu A. Druhý vztah je rodič-syn, kdy syn uzlu A je přímý následník uzlu A a přímý předchůdce uzlu A je rodič pro uzel A.




**Obrázek 5.4** Příklad ukládáného stromu

#### 5.2.1.1 Adjacency list

První a nejvíce triviální možností řešení hierarchie v relační databázi je metoda adjacency list. Princip uložení hierarchie je takový, že každý záznam má v sobě uložený klíč svého otce (přímý záznam o jednu úroveň výše na cestě ke kořenu stromu). Příklad takto uloženého stromu je uvedený v tabulce 5.12. Výhoda takového řešení je jednoduchá implementace a zachování referenční integrity klíčů. Nevýhoda je špatný pohyb po uzlech stromu a komplikované prohledávání podstromů. Tento problém se dá řešit pomocí rekurzivních SQL dotazů, ale ne všechny relační databáze je podporují. [21]

**Tabulka 5.12** Příklad tabulky s řešením hierarchie pomocí adjancency list

| Id | Id rodiče | Popis záznamu |
|---|---|---|
| 1 | NULL | kořen |
| 2 | 1 | syn pro id 1 |
| 3 | 2 | syn pro id 2 |
| 4 | 2 | další syn pro id 2 |
| 5 | 4 | syn pro id 4 |

#### 5.2.1.2 Path enumeration

Další možnost, jak uložit informace o hierarchii, je pomocí path enumeration. Data udržující hierarchii jsou uložena u každého záznamu ve speciálním sloupci ve formě textového řetězce, který tvoří posloupnost identifikátorů všech uzlů na cestě od kořene k danému záznamu včetně. Tyto identifikátory jsou odděleny vždy stejným libovolným oddělovačem, nejčastěji lomítkem nebo pomlčkou. Příklad takové tabulky je uvedený v tabulce 5.13. Výhoda řešení je, že je velice snadné získat celou cestu od kořene stromu k poslednímu uzlu. Nevýhoda takového řešení je, že ztrácíme referenční integritu, jelikož už neprovádíme reference pomocí cizích klíčů. Další nevýhodou je, že získání konkrétního syna je náročnější. Dále je nutné udržovat řetězce cest aktuální po každé úpravě a ve správném formátu. [21]

**Tabulka 5.13** Příklad tabulky s řešením hierarchie pomocí path enumeration

| Id | Cesta | Popis záznamu |
|---|---|---|
| 1 | 1/ | kořen |
| 2 | 1/2/ | syn pro id 1 |
| 3 | 1/2/3/ | syn pro id 2 |
| 4 | 1/2/4/ | další syn pro id 2 |
| 5 | 1/2/4/5/ | syn pro id 4 |

#### 5.2.1.3 Nested sets

Princip tohoto uložení je takový, že každý záznam tabulky reprezentující uzel ve stromě zakódovává své následníky pomocí dvou čísel (levé a pravé). Levé číslo je menší než levá a pravá čísla všech jeho následníků. Pravé číslo je vetší než levá a pravá čísla všech jeho následníků. Hodnoty čísel mezi levým a pravým číslem jsou tedy čísla jeho následníků. Hodnoty levého a pravého čísla jsou vždy mezi hodnotami jeho předchůdce. Příklad takto zakódované hierarchie je v tabulce 5.14. Levá ani pravá hodnota čísla není cizím klíčem. To je také jedna z nevýhod tohoto řešení, jelikož nezachovává referenční integritu. Další nevýhodou je nutnost přepočítání čísel při úpravě stromu a náročné získávání synů. Výhodou je snadné získávání podstromů. [21]

**Tabulka 5.14** Příklad tabulky s řešením hierarchie pomocí nested set

| Id | Levé číslo | Pravé číslo | Popis záznamu |
|---|---|---|---|
| 1 | 1 | 10 | kořen |
| 2 | 2 | 9 | syn pro id 1 |
| 3 | 3 | 4 | syn pro id 2 |
| 4 | 5 | 8 | další syn pro id 2 |
| 5 | 6 | 7 | syn pro id 4 |

#### 5.2.1.4 Closure table

Tato metoda ukládání hierarchie využívá druhou pomocnou tabulku, do které ukládá všechny vazby předchůdce-následník. Každý uzel je navázán i na sebe sama. Dále se do pomocné tabulky ukládá vzdálenost mezi předchůdcem a následníkem, to usnadňuje hledání synů. Výhodou tohoto řešení je zachování referenční integrity, snadné získání podstromů a synů i snadné provádění úprav stromu. Nevýhodou je nutnost vytvoření druhé pomocné tabulky jejíž velikost roste $O(n^2)$, kde $n$ je počet uzlů stromu. Dále je nutné tuto tabulku udržovat při změně stromu. [21] Příklad datové tabulky 5.15 a pomocné tabulky 5.16.

**Tabulka 5.15** Příklad tabulky s daty pro closure table

| Id | Popis záznamu |
|---|---|
| 1 | kořen |
| 2 | syn pro id 1 |
| 3 | syn pro id 2 |
| 4 | další syn pro id 2 |
| 5 | syn pro id 4 |

**Tabulka 5.16** Příklad tabulky s vazbami pro closure table

| Id | Id předchůdce | Id následníka | Vzdálenost |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 2 | 1 | 2 | 1 |
| 3 | 1 | 3 | 2 |
| 4 | 1 | 4 | 2 |
| 5 | 1 | 5 | 3 |
| 6 | 2 | 2 | 0 |
| 7 | 2 | 3 | 1 |
| 8 | 2 | 4 | 1 |
| 9 | 2 | 5 | 2 |
| 10 | 4 | 4 | 0 |
| 11 | 4 | 5 | 1 |
| 12 | 3 | 3 | 0 |
| 13 | 5 | 5 | 0 |

#### 5.2.1.5 Zvolené řešení

Po prozkoumání možných řešení bylo zvoleno ukládání hierarchického stromu pomocí metody closure table, jelikož zachovává referenční integritu, bude se s ním tedy snáze pracovat pomocí ORM frameworku používaného v Play. Dále má dobrou strukturu k získávání synů a podstromů. Velikost pomocné tabulky nebude omezující, jelikož předpokládané množství ukládaných dat nebude rozsáhlý v řádech desítek a nižších stovek záznamů.

## 5.3 Návrh komunikačního rozhraní

Výsledné API bude tvořeno se snahou co nejvíce dodržet principy a pravidla návrhu webového rozhraní podle REST (Representational State Transfer). Jedná se o architektonický styl a nejedná se tedy o protokol či standard. REST se vyznačuje zejména architekturou klient-server, jednotným komunikačním rozhraním a bezstavovostí, která vyžaduje, aby každý dotaz poslaný serveru obsahoval všechny informace k jeho dokončení a nemůže se tedy spoléhat na dotazy

předchozí. Dalšími vlastnostmi, které u REST API můžeme nalézt, jsou možnost využití cache pro ukládání požadavků pro vybrané požadavky, rozšíření funkcionality klienta kódem zaslaným od serveru (Code on Demand) a rozdělení architektury do vrstev.

Informace, data a funkcionality, které můžeme od serveru získat, se nazývají zdroje a ke každému zdroji je přiřazen jeho unikátní identifikátor URI (Uniform Resource Identifier). Pro modifikaci zdrojů na serveru se používají metody zdrojů, které vyvolávají požadované operace se zdroji a převádí je do nového stavu. [22]

Pro potřeby vytvoření API je nutné použít vhodný protokol a formát posílaných dat, který nám umožní snadno provádět operace čtení, ukládání, aktualizace a mazání dat. Toto rozhodnutí je za mě již učiněno na základě požadavků *NFR1 Dostupnost přes webové rozhraní*, *NFR3 Volba technologie* a *NFR5 Nezavádění nových technologii*, jelikož současné řešení aktuálně využívá ke komunikaci mezi klienty a serverem protokol HTTP a data jsou posílána ve formátu JSON. Existují i další formáty, které se běžně v kombinaci s REST používají. Jsou to například HTML, XML či prostý text.

### 5.3.1 Protokol HTTP a REST

Jako komunikační protokol bude, jak bylo zmíněno v předchozí sekci, použit protokol HTTP. Jedná se o jeden z nejvíce využívaných protokolů použitých ve spojení s REST architekturou. Při použití HTTP společně s REST můžeme jednotlivé zdroje identifikovat a také k nim přistupovat pomocí adres URL (příklad: http://www.domain.net/cars/123) a metody zdrojů volat pomocí HTTP metod. Návratové kódy používané v HTTP nám dále poskytnou další informace o zdroji. [23, 24]

#### 5.3.1.1 HTTP Metody

Dále bude popsáno využití jednotlivých metod protokolu HTTP pro práci se zdroji. [23]

**Metoda GET** Požadavek s metodou GET se používá pouze k získání informací ze serveru. Mělo by se jednat vždy o bezpečné metody, to znamená, že jejich volání nesmí jakkoliv změnit stav libovolného zdroje.

Dále by měla metoda GET být idempotentní a tedy vícenásobné volání stejného požadavku by mělo vrátit vždy stejný výsledek, dokud jiný požadavek nezmění stav zdroje.

**Metoda POST** Metoda POST je použita ve chvíli, kdy je potřeba vytvořit nový záznam zdroje. Ze své podstaty se nejedná o bezpečnou ani idempotentní metodu, jelikož mění stav zdroje na serveru.

**Metoda PUT** Primární účel metody PUT je úplná aktualizace již existujícího zdroje. Jedná se ve svém důsledku o nahrazení zdroje novými údaji. Server se může rozhodnout, jestli má dojít k vytvoření nového zdroje v případě, kdy vyžádaný zdroj neexistuje.

Nejedná se o bezpečnou metodu, ale její opakované volání by mělo mít vždy stejný výsledek a je tedy idempotentní.

**Metoda DELETE** Metoda DELETE slouží ke smazání volaného zdroje na volané adrese. tato operace není bezpečná, ale můžeme ji brát jako idempotentní, kdy opakované volání již nevyvolává další změny zdrojů.

**Metoda PATCH** Na rozdíl od metody PUT se metoda PATCH používá pro částečnou aktualizaci zdroje a dochází tedy ke změně pouze vybrané části zdroje.

#### 5.3.1.2 Návratové stavové kódy

Každý požadavek v HTTP, ať už byl úspěšně dokončen, nebo nemohl být z nějakého důvodu zpracovaný, by měl vrátit správný HTTP stavový kód, který informuje klienta o celkovém výsledku požadavku. Tyto kódy jsou rozděleny do pěti kategorii: [25]

- 1xx (100–199) : Informativní odpověď na úrovní protokolu.
- 2xx (200–299) : Indikují úspěšné zpracování požadavku.
- 3xx (300–399) : Kódy přesměrování oznamují klientovi, že musí provést nějaké další úkony pro dokončení požadavku.
- 4xx (400–499) : Indikují problém na straně klienta, který brání zpracování požadavku.
- 5xx (500–599) : Oznamují chybu na straně serveru, která brání úspěšnému dokončení zaslaného požadavku.

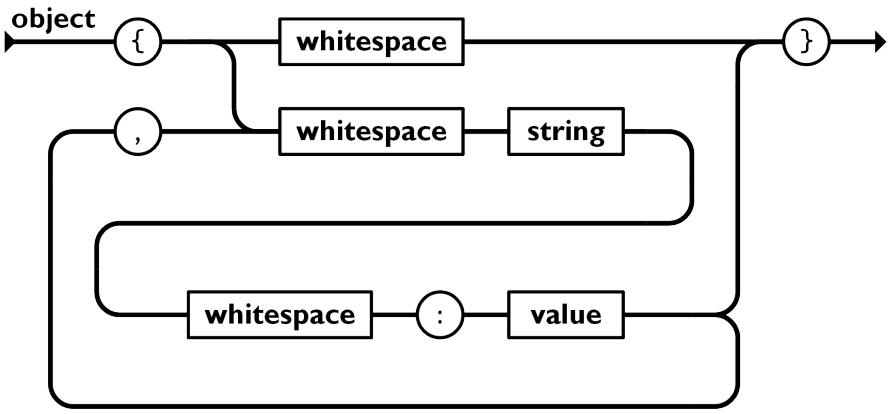
**Tabulka 5.17** Příklad HTTP metod a operací [23]

| HTTP metoda | Operace | Příklad návratového kódu |
|---|---|---|
| GET | Získání záznamu | 200 - Ok<br>404 - Not found<br>400 - Bad request |
| POST | Uložení nového záznamu | 201 - Created<br>404 - Not found<br>400 - Bad request |
| PUT | Aktualizace stávajícího záznamu | 200 - Ok<br>404 - Not found<br>400 - Bad request |
| DELETE | Smazání záznamu | 200 - Ok<br>204 - No content |

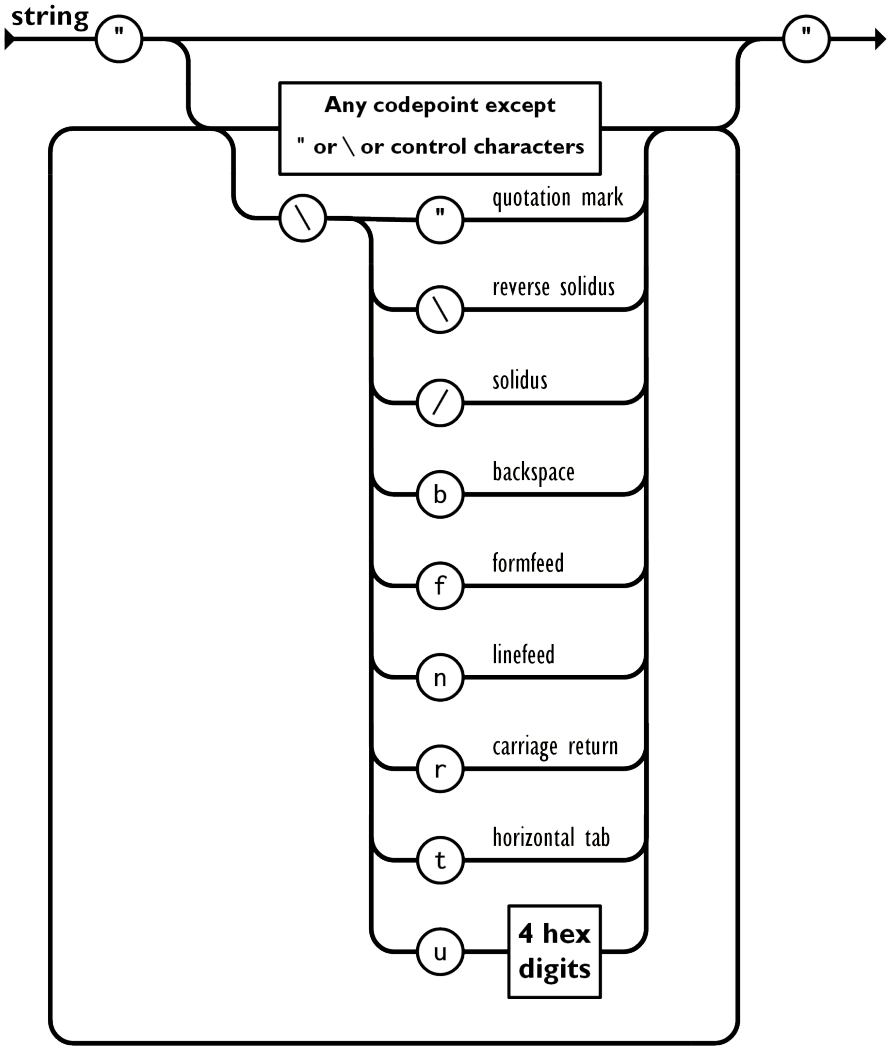
### 5.3.2 Formát posílaných dat JSON

Data budou posílána ve formátu JSON. Jedná se nenáročnou, textově založenou a jazykově nezávislou syntaxi pro definování strukturovaných dat. JSON se obecně skládá z šesti strukturálních znaků ([ ] { } , :), řetězců, čísel, třech literálů (true, false, null) a nevýznamných mezer.

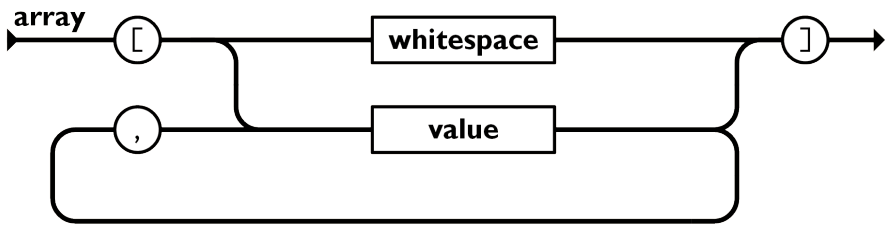
Celý princip JSON je postaven na dvou strukturách, kde první je kolekce párů název/hodnota, které dohromady tvoří objekt. Druhou strukturou je kolekce hodnot. Hodnotou může být objekt, kolekce, číslo, řetězec a literál. [26] Syntaxe objektu JSON je znázorněna pomocí diagramů na obrázcích 5.5, 5.6, 5.7 a 5.8.



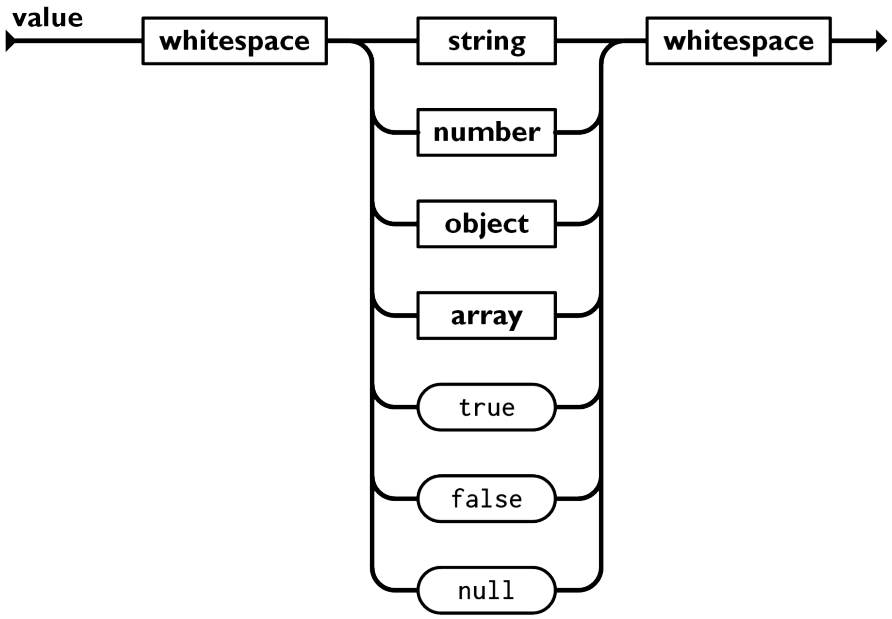

**Obrázek 5.5** Schéma struktury JSON objektu[27]




**Obrázek 5.6** Schéma struktury JSON řetězce[27]




**Obrázek 5.7** Schéma struktury JSON kolekce[27]




**Obrázek 5.8** Schéma struktury JSON hodnoty[27]

### 5.3.3 Výsledný návrh rozhraní

Jelikož se nové funkcionality opírají hlavně o operace ukládání, čtení, aktualizace a mazání dat, zkráceně CRUD (Create, Read, Update, Delete) a na čtení výčtových typů, bude celé komunikační rozhraní pro jednotlivé entity vytvořeno pomocí následujícího vzoru, který bude uveden na příkladu rozhraní pro práci s kontrolou vozidla 5.18 a na typu kontroly vozidla, který reprezentuje výčtový typ 5.19. Celý návrh je k nahlédnutí v příloze G. Všechna data budou posílána ve formátu JSON.

**Tabulka 5.18** Návrh API pro kontrolu vozidla

| HTTP metoda | URL | Popis |
|---|---|---|
| GET | /car-checks/{id} | Získání jednoho záznamu kontroly s parametrem {id} |
| GET | /car-checks | Získání všech kontrol vozidel |
| POST | /car-checks | Uložení nové kontroly s daty poslaných těle požadavku |
| PUT | /car-checks/{id} | Aktualizace kontroly vozidla s parametrem {id} pomocí dat poslaných těle požadavku |
| DELETE | /car-checks/{id} | Smazání kontroly vozidla s parametrem {id} |

**Tabulka 5.19** Návrh API pro typ kontroly vozidla

| HTTP metoda | URL | Popis |
|---|---|---|
| GET | /car-checks/types/{id} | Získání jednoho záznamu typu kontroly s parametrem {id} |
| GET | /car-checks/types | Získání všech typů kontrol vozidel |

## 5.4 Použité technologie a architektura

Z nefunkčních požadavku *NFR3 Volba technologie* a *NFR5 Nezavádění nových technologii* vychází volba technologie pro vytvoření samotné webové aplikace na framework Play společně s programovacím jazykem Java a pro realizaci evidence dat je zvolena relační databáze PostgreSQL.

Architektura rozšíření bude následovat stejné principy a vzory jako stávající aplikace, které jsou popsány v sekci 4.1.2 *Architektura backendu* a nebude zavádět nové architektonické vzory.

## 5.5 Použité nástroje při návrhu

Tato sekce obsahuje představení nástrojů, který byly použity v průběhu návrhu pro tvorbu doménových modelů a následných datových modelů.

### 5.5.1 Enterprise Architect

Pro vytvoření doménových modelů byl zvolen nástroj Enterprise Architect, který je vyvíjen společností Sparx Systems Pty Ltd. Obsahuje velké množství nástrojů pro analýzu, modelování podnikových procesů, modelování architektury programů a systémů včetně vizualizace výsledků.

Podporuje vytváření modelů ve formátu UML, SysML, BPMN a v dalších otevřených standardech. Mezi další poskytované funkcionality patří testování vytvořených modelů, generování kódu a dokumentace z a do modelů. Obsahuje také podporu pro týmy vývojářů, jejich kolaboraci na jednom projektu a další nástroje. [28]

### 5.5.2 Oracle Sql Developer

Nástroj Sql Developer vytvořený společností Oracle je bezplatné vývojové prostředí pro vytváření a správu databází Oracle. Nabízí kompletní řešení pro vývoj PL/SQL aplikací, spouštění dotazů a skriptů, nástroje pro správu databází a pro nás důležité modelovací nástroje. [29]

V tomto nástroji byly vytvořeny konceptuální a relační datové modely databáze. Později byly tyto modely využity jako podklad pro vygenerování základního SQL kódu.

# Kapitola 6
# Implementace

Tato kapitola se zabývá implementací samotného rozšíření backendu na základě předchozí analýzy a návrhu. Nalézá se zde postup realizace databáze a informace k její dokumentaci. Dále se zde nalézá postup implementace kódu a zhodnocení míry pokrytí funkčních a nefunkčních požadavků z kapitoly 4.3 *Požadavky na rozšíření*. V rámci této kapitoly se také nalézá část, která se zabývá principy použitých k dokumentaci rozšíření a součástí je i sekce věnovaná dokumentaci výsledného API, které implementované rozšíření poskytuje.

## 6.1 Vývojové nástroje použité při implementaci

Pro vytváření aplikací dnes existuje velké množství vývojových prostředí nazývané jako IDE (Integrated Development Environment), které může programátor použít. Liší se svým účelem a specializací na určité technologie. Existují vývojová prostředí určená pro tvorbu webových, mobilních či terminálových aplikací. Jejich hlavním cílem je usnadnit a urychlit vývoj softwaru.

Další kategorie jsou podporující nástroje, které neslouží přímo pro vývoj softwaru, ale velice usnadňují ostatní procesy, které se vývoje týkají. Může se jednat o nástroje pro administraci databází, vedení dokumentace či správu a verzování souborů v projektu.

### 6.1.1 IntelliJ IDEA

Jakožto vývojové prostředí bylo zvoleno IDE vytvářené společností JetBrains s. r. o., které je k dispozici jak v bezplatné Community verzi při dodržení licenčních podmínek, tak i v placené verzi Ultimate. Prostředí podporuje práci s jazykem Java a dalšími JVM (Java Virtual Machine) jazyky jako Kotlin, Scala a Groovy. Dále poskytuje integrované nástroje pro práci s mnoha populárními frameworky jako je Spring, Jakarta EE a další. Funkcionality prostředí je možné rozšířit pomocí dalších pluginů, kde se nalézá i plugin pro podporu frameworku Play. [30]

### 6.1.2 DBeaver

Pro efektivní administraci a práci s databází PostrgeSQL byl zvolen nástroj DBeaver Community. Jedná se o univerzální multiplatformní grafický databázový administrační nástroj. Podporuje širokou škálu databází, jako jsou Microsoft SQL Server, MySQL, MariaDB, Postgres, Azure SQL Server a mnoho dalších.

Obsahuje grafický editor a správce dat v připojené databázi, což usnadňuje kontrolu a případnou úpravu uložených dat. Dále obsahuje integrovaný SQL editor pro vykonávání skriptů nad databází. [31]

### 6.1.3 Git

Pro správu souborů projektu v Uniqway je použit bezplatný open-source distribuovaný verzovací systém Git. Umožňuje zaznamenávat změny souborů v čase, vedení historie a udržovat různé vývojové větve. [32]

## 6.2 Postup implementace

Implementace rozšíření byla provedena ve třech celcích. Jedná se o celky kontroly vozidel, vybavení vozidla a evidence pneumatik, které nám vzešly z předchozích kapitol. Realizace mohla být takto snadno rozdělena, jelikož jsou tyto celky navzájem takřka nezávislé a pro jejich fungování nejsou ostatní části potřeba.

Postup implementace každého celku je takřka volitelný, jelikož rozšíření není implementováno více vývojáři a záleží tedy jen na programátorovi, jaký postup zvolí. Začít vytvářet náhodně třídy z řešení není ideální a pravděpodobně ani efektivní, a proto byl zvolen následující postup řešení.

Implementace začínala od nejnižší vytvářené vrstvy, kdy nejprve docházelo k vytvoření rozšíření databáze následované datovou vrstvou s objekty DAO včetně tříd entit pro potřeby ORM. Dalším krokem bylo vytvoření objektů DTO z komunikační vrstvy, po kterých následovala realizace vrstvy služeb, jelikož až v tu chvíli byly definované vstupní a výstupní parametry jednotlivých služeb. Poslední dokončenou vrstvou byla vždy komunikační vrstva, kdy se dokončila implementace kontrolérů, akcí a bylo sepsáno mapování URL na jednotlivé akce.

### 6.2.1 Vytvoření databáze a datové vrstvy

Implementace každého celku začínala vytvořením schématu databáze a sepsáním evoluce, která obsahuje přidané změny. Poté co byla databáze realizována, bylo následujícím úkonem sepsání tříd enitit, které mapují objekty na záznamy ve vytvořené databázi.

#### 6.2.1.1 Realizace databáze

Pro realizaci změn v databázi byla použita vestavěná funkce frameworku Play, která pomocí evolucí převádí databázi do nového stavu. Samotný framework kontroluje, zdali je schéma připojené databáze v aktuálním a konzistentním stavu a aplikuje evoluce podle potřeby.

Informace o aplikovaných evolucích si framework ukládá do samotné databáze v samostatné tabulce a pouze porovnává aktuální dostupné evoluce s těmi dříve uloženými. Pokud je nalezen rozdíl v nějaké evoluci, nebo je přidána nová, jsou aplikovány potřebné změny.

Evoluce je skript založený na jazyce SQL, který obsahuje dvě části [33] a to:

**Ups** : Tato část evoluce obsahuje SQL kód, který zavádí nové struktury do schématu databáze a může provádět další operace s daty v databázi.

**Downs** : Tato část evoluce obsahuje SQL kód, který vrací provedené změny databáze definované v části Ups do stavu před jejich aplikováním.

**Výpis kódu 6.1** Příklad evoluce

```
# --- !Ups
CREATE TABLE users (
    id          bigserial PRIMARY KEY,
    email       varchar NOT NULL UNIQUE,
    first_name  varchar NOT NULL,
    last_name   varchar NOT NULL
);
```

```
    # --- !Downs
    DROP TABLE users;
```

Při vytváření evoluce databáze bylo využito předchozí práce na datových modelech v nástroji Sql Developer, který umožňuje vygenerování SQL kódu přímo z vytvořených datových modelů. Tento SQL kód posloužil jako základ pro napsání evoluce databáze. Stejně jako implementace kódu, byl i SQL kód evoluce vytvářen ve třech hlavních celcích, kdy se postupně přidávali struktury pro kontrolu vozidel následované strukturami pro evidenci vybavení a na závěr byly přidány tabulky potřebné pro evidenci výměn pneumatik.

Do vytvořené evoluce byly rovněž do sekce Ups napsány SQL příkazy pro vložení startovacích dat pro tabulky, které obsahují výčtové typy, které jsou nutné při vytváření nových záznamů. Tyto data byla získána na základě komunikace a analýzy dodaných podkladů od Uniqway.

#### 6.2.1.2 Třídy entit a Java Persistence

Třídy entit nám poskytují mapování mezi vytvořenými objekty a záznamy v relační databázi pomocí JPA (Java Persistence API) a jeho konkrétní implementace Hibernate. Jedná se o realizace entit pro potřeby ORM, viz sekce 4.1.1.3 *Object Relational Mapping*.

Každá entitní třída obsahuje privátní atributy, které odpovídají jejich protějšku v databázi, navíc může obsahovat takové atributy, které nám můžou poskytovat mapování vazeb a tím zjednodušit přístup k ostatním entitám reprezentující namapované záznamy z ostatních tabulek. Lze tedy jednoduše mapovat také záznamy, mezi kterými existuje v databázi vytvořená vazba (1:1, 1:M, M:N).

Atributy jsou ve třídě zapouzdřeny a přistupuje se k nim s pomocí přístupových metod. Konkrétní mapování atributů do databáze je popsáno za použití anotací, které nám specifikují například informace o názvu sloupce či o konkrétní vazbě na jinou entitu. Další možností popsání mapování je pomocí externího XML souboru. [34] Příklad mapované entity pomocí anotací je uveden ve výpisu kódu 6.2.

Atributy jsou v Hibernate v základu mapovány na sloupce s názvem, který je stejný jako název atributu, nebo následuje jmennou konvenci.[35]

**Výpis kódu 6.2** Příklad třídy entity

```
@Entity
/* Název mapované tabulky v databázi */
@Table(name = checks")
public class Check extends Model {

    /* Anotace specifikuje, že atribut mapuje navázanou
    entitu Car pomocí sloupce car_id v databázi. */
    @ManyToOne
    @JoinColumn(name = "car_id")
    private Car car;

    /* Anotace specifikuje, že atribut obsahuje navázané
    entity CarDefect pomocí atributu check v entitě Defect. */
    @OneToMany(mappedBy = "check")
    private List<Defect> carDefects;

    /* Přístupové metody */
    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
```

```
        this.car = car;
    }

    public List<CarDefect> getCarDefects() {
        return carDefects;
    }

    public void setCarDefects(List<CarDefect> carDefects) {
        this.carDefects = carDefects;
    }
}
```

#### 6.2.1.3 Data Access Object

Pro přístup k datům je využit DAO vzor, viz sekce 4.1.2 *Architektura backendu*, a tedy ke každé entitě byla vytvořena odpovídající třída DAO, která dědí základní implementaci pro práci s daty a přístup k databázi z již existující generické třídy BaseDao a byla rozšířena o další funkce podle konkrétní potřeby dané entity.

### 6.2.2 Vytvoření vrstvy služeb

Pro zpracování hlavní logiky byla vytvořena vrstva služeb, která využívá funkcí z datové vrstvy pro práci z daty. Tato vrstva existuje mezi komunikační vrstvou a datovou vrstvou a je zodpovědná například za zpracovávání a vytváření objektů DTO z mapovaných entit, zpracování dat, ověřování zdali jsou zpracovávané objekty ve validním stavu a předávání objektů ke zpracování datovou vrstvou.

### 6.2.3 Vytvoření komunikační vrstvy

Za zpracování HTTP požadavků jsou zodpovědné kontroléry a jejich metody, které jsou nazvány akcemi. Pro každý koncový bod navrženého webového API existuje jedna akce, která obsahuje kód pro validaci, získání vstupních dat z požadavku, volání vrstvy služeb a vytvoření odpovědi.

Obsah posílaných dat je definován pomocí objektů DTO. Jedná se o běžné Java třídy s atributy, které jsou stejné jako v posílaných objektech JSON. Dále jsou doplněny o omezující anotace s informacemi, které jsou používány při validaci. Tyto objekty jsou serializované na objekt JSON a následně posílány jako text v těle odpovědi. Při přijímání požadavku od klientů jsou data ve formátu JSON z těla požadavku de-serializované do DTO objektu.

Vytvořené webové komunikační rozhraní, včetně formátu posílaných dat z pohledu ze strany klienta je popsáno ve Swagger dokumentaci, více k API dokumentaci v sekci 6.5.3 *Dokumentace vytvořeného API*.

## 6.3 Pokrytí funkčních požadavků

Následující sekce je zaměřena na úroveň pokrytí funkčních požadavků, které byly popsány během předchozí analýzy v části 4.3.1 *Funkční požadavky*.

První implementovaná část se zabývala evidencí technické kontroly vozidel v rámci které byly plně pokryty funkční požadavky *FR2 Poskytnutí informací ke kontrole vozidla*, *FR3 Poskytnutí detailu konkrétní kontroly vozidla*, *FR4 Poskytnutí seznamu kontrol vozidla*. Požadavek *FR5 Modifikace neukončené kontroly vozidla* byl pokryt se změnou, kdy v případě smazání kontroly nedojde k odstranění evidovaných defektů, ale k jejich přiřazení ke kontrolovanému vozidlu. Částečně byl pokryt i funkční požadavek *FR1 Evidence kontrol vozidel*, který byl plně pokryt

až v druhé části implementace v rámci evidence vybavení vozidla. Dále se tato část zabývala evidencí defektů na vozidle, ve které byly pokryty funkční požadavky *FR6 Evidence defektů*, *FR7 Poskytnutí informací k defektu*, *FR8 Poskytnutí detailu konkrétního defektu*, *FR9 Poskytnutí seznamu všech platných defektů*, *FR10 Poskytnutí seznamu všech defektů* a *FR11 Modifikace evidovaného defektu*.

Druhá část implementace byla zaměřena na vytvoření evidence vybavení vozidla, v rámci které byly realizovány funkční požadavky *FR12 Evidence vybavení vozidla*, *FR13 Poskytnutí seznamu vybavení vozidla* a *FR14 Modifikace stavu vybavení vozidla*. Společně s první částí poskytují potřebné funkcionality pro provádění celkových kontrol vozidel ambasadorem.

Třetí část implementace byla věnována evidenci pneumatik a jejich výměn a pokrývá tedy funkční požadavky *FR15 Evidence výměny kol*, *FR16 Poskytnutí seznamu předchozích výměn kol* a *FR17 Evidence pneumatik*. Ve třetí části implementace tedy byly pokryty všechny zbývající funkční požadavky.

## 6.4 Pokrytí nefunkčních požadavků

Následující sekce se věnuje pokrytí nefunkčních požadavků, které byly vytvořeny v rámci analýzy v části 4.3.2 *Nefunkční požadavky*.

Nově vytvořené rozšíření komunikuje za pomocí HTTP přes webové rozhraní a tím je splněn *NFR1 Dostupnost přes webové rozhraní*.

Databáze obsahuje u výčtových typů sloupce pro uložení českého i anglického popisu. Logika rozšíření umožňuje na základě poslané hlavičky *Accept-Language* v požadavku získat a poskytnout obsah těchto sloupců. Tímto je splněn *NFR2 Lokalizace*.

Pro realizaci databáze byla využita a rozšířena stávající databáze PostgreSQL. Rozšíření je napsáno v jazyce Java za použití frameworku Play a tím je splněn *NFR3 Volba technologie*.

Rozšíření využilo již existujícího systému ověřování uživatelů u všech nově vytvořených koncových bodů a byl tedy splněn *NFR4 Ověření uživatelé*.

Během implementace nebyly zavedeny žádné nové technologie a byly použity pouze aktuálně využívané technologie v Uniqway a tím je splněn *NFR5 Nezavádění nových technologii*.

## 6.5 Dokumentace

### 6.5.1 Dokumentace kódu

Pro každý kód je kromě jeho samotného účelu neméně důležité jeho snadné porozumění dalšími vývojáři za účelem jeho znovupoužití, modifikace a údržby. Za tímto účelem vzniká dokumentace kódu, kterou lze vytvořit například pomocí komentářů v kódu, diagramy či pomocí různých nástrojů. [36]

Při tvoření dokumentace se ovšem může velice snadno stát, že pouze přepíšeme funkci kódu pomocí přirozeného jazyka bez další přidané informace. Takový přepis je ve své podstatě duplicita, která jde proti principu DRY (Don't Repeat Yourself) psaní kódu. Pokud se takto zdokumentovaný kód změní a dokumentace nebude aktualizována, může dojít naopak k zmatení čtenářů. [37]

V realizované implementaci byly výše popsané problémy vyřešeny za pomocí dokumentace, která je založená na sebe-popisném kódu a staví na principu, že kód samotný je svou vlastní dokumentací a komentáře jsou použity opravdu v nejnutnějších případech. Každý kód ovšem není samo-popisný a je nutné dodržet přehledné strukturování kódu, dobré pojmenovávání metod, proměnných a tříd, které názorně určuje jejich funkci a účel v kódu. [36, 38]

### 6.5.2 Dokumentace databáze

Dokumentace databáze je tvořena z dříve vytvořených datových modelů, které jsou k nahlédnutí v přílohách D, E a F. Dále je výsledný SQL kód evoluce doplněný o podrobné komentáře k tabulkám, sloupcům a dalším strukturám. Tyto komentáře poskytují detailní popis jejich účelu.

### 6.5.3 Dokumentace vytvořeného API

Pro využití implementovaného rozšíření a jeho výsledného API dalšími vývojáři bylo potřeba vytvořit dokumentaci k jednotlivým koncovým bodům.

Dokumentace výsledného API, která byla předána týmu v Uniqway, byla vytvořena v nástroji Swagger Editor dostupný na adrese `https://editor.swagger.io`. Tento nástroj již v Uniqway používají a umožňuje vývojářům procházet zdokumentované API v grafické podobě. Výsledný soubor ve formátu yaml, který lze otevřít v nástroji Swagger Editor, včetně offline verze výstupu editoru Swagger byl předán společně s touto prací. Příklad této dokumentace je ukázán na obrázku 6.1.

Soubor dokumentace obsahuje informace k jednotlivým koncovým bodům, jako je jeho adresa, HTTP metoda, parametry a formát posílaných dat včetně formátu odpovědi. Dokumentace obsahuje rovněž obecné příklady komunikace pomocí těchto koncových bodů. V následujícím výpisu kódu 6.3 je uveden příklad takto zdokumentovaného koncového bodu.

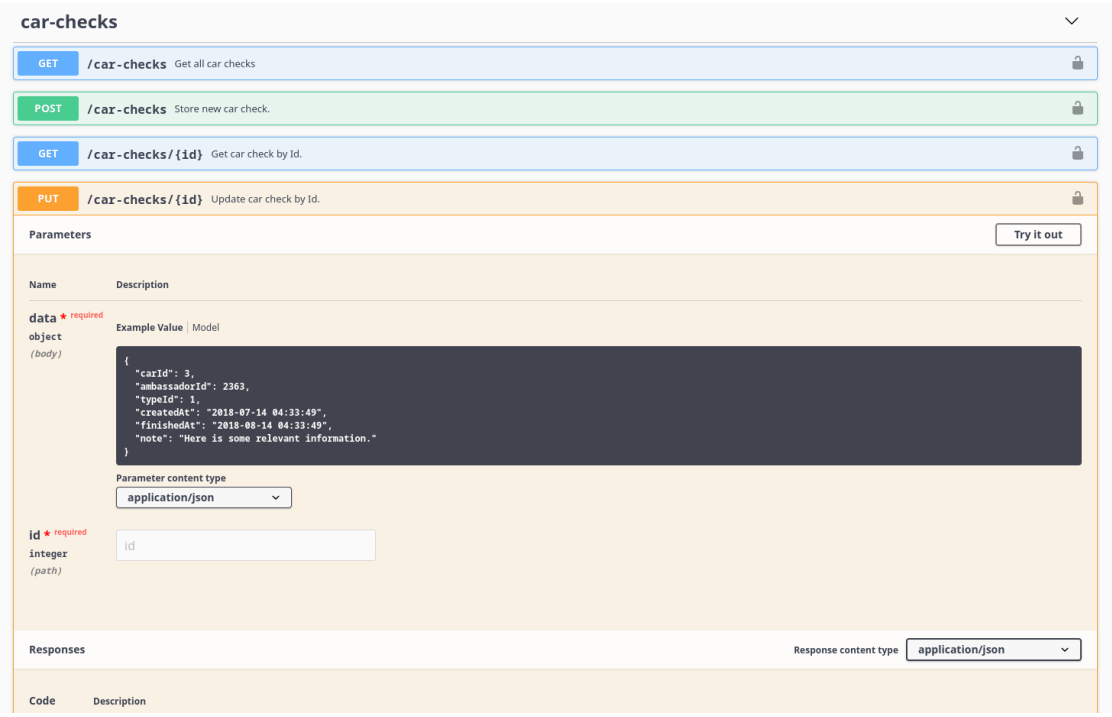
**Výpis kódu 6.3** Příklad yaml dokumentace koncového bodu

```
swagger: '2.0'
info:
  description:
    This is exapmle.
  version: 1.0.0
  title: Example API
security:
  - jwt_token: []

# schemes:
# - http
paths:          # Zde se nalezají adresy ke koncovým bodům
  /user:        # Adresa
    get:        # HTTP metoda
      tags:
      - users
      summary: Get all users
      operationId: get
      produces:
        - application/json          # Formát dat v odpovědi
      parameters:                   # Seznam query parametrů
        - in: query
          name: sorts
          description: Sort by ("id", "name")
          type: string
      responses:                    # Druhy odpovědí a jejich popis
        200:
          description: Success
          headers:
            JWT-Token:
              type: string
              description: JWT token
          schema:
```

```
            $ref: "#/definitions/GetUser"    # Schéma vráceného objektu

definitions:              # Definice objektu
  GetUser:                # Příklad objektu GetUser
    type: object
    properties:
      id:
        type: integer
        example: 1
      name:
        type: string
        description: Username.
        example: user123
```




**Obrázek 6.1** Náhled vygenerované HTML dokumentace

# Kapitola 7
# Testování

Testování kódu je důležitou součástí vývoje jakéhokoliv spolehlivého softwaru, jelikož pomáhá předcházet chybám ve finálním produktu a následným nákladům spojených s odstraněním těchto chyb a jejich dopadů. [39]

Obsahem této kapitoly je úvod do testování kódu a běžně používaných principů. Dále budou uvedeny druhy testů, se kterými se můžeme setkat v průběhu vývoje softwaru a blíže se seznámíme s použitými testy v této práci.

## 7.1 Primární úkony při testování

Testování softwaru, ale i testování obecně, lze rozdělit do několika základních úkonů a cílů. Jedná se zejména o:

1. Identifikování zdroje a velikosti rizika, které se testováním sníží. Snažíme se nejdříve zaměřit na ta rizika s největší závažností a nejčastějším výskytem.

2. Provedení testů za účelem snížení rizika. Takové testy obsahují jak pozitivní případy, kdy se ověřuje správné fungování podle požadavků, tak negativní případy, které testují situace ve kterých testovaný případ podle předpokladů selže.

3. Ukončení dalšího testování v případě, že cena testování je vyšší než dopady rizika. Testování by však stále mělo obsahovat všechny pozitivní případy a co nejvíce negativních případů.

4. Testování by mělo být chápáno jako standardní projekt v rámci vývoje.

Výše uvedené úkony by měly být zařazeny a provedeny v průběhu životního cyklu softwaru.[39] Existuje mnoho způsobů, jakými lze tento postup zavést. Může se například jednat o samostatnou fázi vývoje, jak je uvedeno ve volbě metodiky této práce, viz sekce 3.3 *Volba metodiky pro vývoj rozšíření*, nebo můžeme například použít metodiku zvanou jako Test Driven Development (zkráceně TDD), která se opírá o základní princip psaní testů před testovaným kódem. Vývojář až po napsání testů vytváří samotný kód nové funkcionality dokud všechny dříve napsané testy neskončí úspěšně. Testování je tedy součástí celého vývojového cyklu. [39, 40, 41]

## 7.2 Limitace testování

Testování tedy samo o sobě nemůže zaručit naprosto bezchybný software, ale výrazně přispívá k odstranění chyb. Úspěšné testování nám pouze potvrzuje, zdali testované funkce splňují předepsaná kriteria.

Jen velice málo projektů může být testováno na všechny možné případy vzhledem k limitovaným zdrojům či zkušenostem vývojářů, a je proto nutné zavádět takové metody a postupy vývoje, které vedou k vytváření kvalitního softwaru a minimalizují výskyt chyb. [39, 41]

## 7.3 Kategorie testů

Samotné testy lze zařadit do různých kategorii podle toho, jak přistupují k testovanému softwaru. Mezi hlavní kategorie, které jsou zmíněné v [39, 42], patří:

**Statické testování** : Tato kategorie testů se zabývá správností samotných dokumentů a kódu po formální stránce. Rovněž do této kategorie lze zařadit ověřování správnosti vytvořených schémat či dokumentace. Probíhá bez spuštění vyvíjené aplikace a provádí se manuálně nebo za pomocí automatizovaných nástrojů.

**White box testování** : Jedná se o takové testování softwaru, kdy máme přístup ke zdrojovému i spustitelnému kódu. To nám umožňuje otestovat každou řádku kódu a vytvořenou logiku pro všechny možné případy.

**Black box testování** : Testy spadají do této kategorie v případě, pokud nemáme přístup ke zdrojovému, ale máme k dispozici spustitelný kód. Dochází tedy zejména k ověřování funkčnosti celku na základě porovnání očekávaných a získaných výstupů pro dané vstupy.

**Výkonnostní testování** : Zde dochází k zaměření na samotný běh softwaru. Může se jednat například o ověření doby výpočtu, rychlosti odpovědí, paměťové náročnosti a dalších parametrů.

## 7.4 Testování nového rozšíření

Nové funkcionality rozšíření byly otestovány pomocí různých druhů automatických testů po dokončení implementace. Tyto automatické testy byly vytvořeny a spouštěny během hlavního testování. Jediným druhem testů, které nebyly prováděny během hlavního testování, byly testy z kategorie manuálního API testování, které byly použity pro ověření fungování nových funkcionalit během vývoje rozšíření.

### 7.4.1 Použité typy testů

V následující části budou přiblíženy jednotlivé druhy použitých testů a za jakým účelem byly vytvořeny.

#### 7.4.1.1 Statické testy

Statické testování kódu bylo zajištěno přímo použitým vývojovým prostředím IntelliJ Idea, které automaticky provádí inspekci psaného kódu a ohlašuje nalezené chyby. [43]

Ostatní dokumenty, zejména grafického charakteru, byly ověřovány manuálně. Jedná se například o vytvořené diagramy, schémata a dokumentaci.

#### 7.4.1.2 API testy

Tyto testy byly použity k otestování funkčnosti výsledného API, které rozšíření poskytuje. Během testování se k jednotlivým koncovým bodům přistupovalo jako by nebyl znát zdrojový kód a byly pouze ověřovány získané výstupy. Jedná se tedy o kategorii black box testů.

V průběhu vývoje byly vytvářeny manuální testy pro ověření základní funkčnosti nových koncových bodů. Toto manuální testování bylo provedeno pomocí volání jednotlivých koncových bodů z nástroje Postman a kontroly vrácené odpovědi.

Následně v hlavní testovací fázi byly tyto testy přepsány a rozšířeny do spustitelných skriptů v jazyce Python. Tyto již automatické testy ověřují funkčnost jednotlivých koncových bodů, formát vrácených dat a dále ověřují správné chování rozšíření aplikace a jeho funkcionality.

#### 7.4.1.3 Jednotkové testy

Jednotkové testy byly využity pro testování samotného zdrojového kódu a tedy se jedná o kategorii white box testů. Jejich cílem je otestovat malé izolované části kódu a ověřit jejich správné fungování. V našem případě se jedná o jednotlivé metody tříd. [44]

Jednotkové testy byly použity pro otestování celé vrstvy služeb, jelikož se zde nalézá hlavní logika rozšíření a je zde tedy největší pravděpodobnost výskytu chyb. Jednotkovými testy nebyly pokryty objekty DAO a DTO, jelikož neobsahují takřka žádnou přidanou logiku a nebo ji dědí z již existujících tříd. Dále nebyly jednotkovými testy pokryty kontroléry, jelikož jejich správné fungování je ověřeno pomocí API testů.

Každý test byl strukturován podle vzoru Arrange, Act, Assert, který je popsán v [44], a rozděluje test do tří části:

**Arrange** : V této části se připravují vstupní proměnné testu.

**Act** : Tato část je zodpovědná za provedení samotného testu. Dochází zde k volání testované metody a získání výstupu.

**Assert** : Jedná se o poslední část ve které se ověřují získané výsledky a dochází k vyhodnocení úspěšnosti testu.

# Kapitola 8
# Závěr

Tato práce se zabývala návrhem a vytvořením nového rozšíření serverové části webové aplikace používané ve společnosti Uniqway. Byly pokryty jednotlivé fáze vývojového cyklu softwaru, které byly započaté volbou vhodné metodiky, následované analýzou stávajícího stavu a specifikací požadovaných funkcionalit rozšíření na základě informací a potřeb získaných od týmu v Uniqway. Byl proveden návrh řešení na jehož základě byla vytvořena implementace nových funkcionalit, které jsou poskytovány přes webové rozhraní. Výsledná implementace byla pokryta jednotkovými testy a komunikační rozhraní je testováno pomocí API testů. Součástí řešení dodaného Uniqway je i dokumentace vytvořeného databázového schématu a komunikačního rozhraní.

Vytvořené rozšíření umožňuje uživatelům skrze webové rozhraní provozovat evidenci kontrol vozidel, provedených úklidů vozidla a vést detailní záznamy nalezených defektů, včetně jejich lokace na vozidle. Dále jsou k dispozici funkcionality pro vedení evidence vybavení jednotlivých vozidel a kategorizace vybavení. Uživatelům je dále poskytnuta možnost evidence výměn pneumatik i pneumatik samotných s možností ukládání průběžného opotřebení vzorku.

V budoucnu by bylo možné některé funkcionality rozšířit a upravit tak, aby mohly být poskytnuty nejen samotným správcům a zaměstnancům v Uniqway, ale i jejich klientům a umožnit jim například ohlašování závad na zapůjčeném vozidle.

# Bibliografie


1. GUTWIRTH, Jiří. *Rozšíření webové aplikace pro správce systému sdílení automobilů*. 2021. Dostupné také z: `https://dspace.cvut.cz/bitstream/handle/10467/95122/F8-BP-2021-Gutwirth-Jiri-BP-Gutwirth-Jiri-2020.pdf?sequence=-1%5C&isAllowed=y`.
2. EVERETT, Gerald D; MCLEOD JR, Raymond. Software Testing. In: John Wiley & Sons, Inc., Hoboken, New Jersey, 2007, s. 29–31. ISBN 978-0-471-79371-7.
3. BRUCKNER, Tomáš; BUCHALCEVOVÁ, Alena; CHLAPEK, Dušan; ŘEPA, Václav; STANOVSKÁ, Iva; VOŘÍŠEK, Jiří. Tvorba informačních systémů: Principy, metodiky, architektury. In: U Průhonu 22, Praha 7: Grada Publishing, a. s., 2012, s. 102–112. ISBN 978-80-247-4153-6.
4. BRUCKNER, Tomáš; BUCHALCEVOVÁ, Alena; CHLAPEK, Dušan; ŘEPA, Václav; STANOVSKÁ, Iva; VOŘÍŠEK, Jiří. Tvorba informačních systémů: Principy, metodiky, architektury. In: U Průhonu 22, Praha 7: Grada Publishing, a. s., 2012, s. 115–119. ISBN 978-80-247-4153-6.
5. FOWLER, Martin; HIGHSMITH, Jim et al. The agile manifesto. *Software development*. 2001, roč. 9, č. 8, s. 28–35. Dostupné také z: `http://www.awslad.com/wp-content/uploads/2010/01/The_Agile_Manifesto_SDMagazine1.pdf`.
6. KOĎOUSKOVÁ, Barbora. *Co je agilní vývoj aplikací a kdy ho využívat* [online]. [B.r.]. Dostupné také z: `https://www.rascasone.com/cs/blog/co-je-agilni-vyvoj`.
7. BRUCKNER, Tomáš; BUCHALCEVOVÁ, Alena; CHLAPEK, Dušan; ŘEPA, Václav; STANOVSKÁ, Iva; VOŘÍŠEK, Jiří. *Tvorba informačních systémů: Principy, metodiky, architektury*. U Průhonu 22, Praha 7: Grada Publishing, a. s., 2012. ISBN 978-80-247-4153-6.
8. ŠTRÁFELDA, Jan. *Backend* [online]. [B.r.]. Dostupné také z: `https://www.strafelda.cz/backend`.
9. BEAL, Vangie. *API (Application Program Interface)* [online]. [B.r.]. Dostupné také z: `https://www.webopedia.com/definitions/api`.
10. RICHARD-FOY, Julien. Play Framework Essentials. In: 35 Livery Street, Birmingham, UK: Packt Publishing, Limited, 2014, s. 23–24. ISBN 9781783982400.
11. *Java Play - What is Play?* [Online]. [B.r.]. Dostupné také z: `https://www.playframework.com/documentation/2.8.x/Introduction`.
12. O'REGAN, Gerard. Java Programming Language. In: *The Innovation in Computing Companion: A Compendium of Select, Pivotal Inventions*. Cham: Springer International Publishing, 2018, s. 171–174. ISBN 978-3-030-02619-6. Dostupné z DOI: `10.1007/978-3-030-02619-6_35`.

13. JAVATPOINT. *JPA Object Relational Mapping* [online]. [B.r.]. Dostupné také z: https://www.javatpoint.com/jpa-object-relational-mapping.
14. JAVATPOINT. *JPA Introduction* [online]. [B.r.]. Dostupné také z: https://www.javatpoint.com/jpa-introduction.
15. POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL - About* [online]. [B.r.]. Dostupné také z: https://www.postgresql.org/about.
16. *PostGis - Features* [online]. [B.r.]. Dostupné také z: https://postgis.net/features%C2%B4.
17. MAYR, Christine; ZDUN, Uwe; DUSTDAR, Schahram. View-based model-driven architecture for enhancing maintainability of data access services. *Data & Knowledge Engineering*. 2011, roč. 70, č. 9, s. 794–819. ISSN 0169-023X. Dostupné z DOI: https://doi.org/10.1016/j.datak.2011.05.004.
18. BAELDUNG. *The DTO Pattern (Data Transfer Object)* [online]. 2021. Dostupné také z: https://www.baeldung.com/java-dto-pattern.
19. ŠTRÁFELDA, Jan. *Frontend* [online]. [B.r.]. Dostupné také z: https://www.strafelda.cz/frontend.
20. ŠTRÁFELDA, Jan. *Functional and Nonfunctional Requirements: Specification and Types* [online]. 2021. Dostupné také z: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.
21. KARWIN SOFTWARE SOLUTIONS LLC. *Models for hierarchical data* [online]. 2010. Dostupné také z: https://www.slideshare.net/billkarwin/models-for-hierarchical-data.
22. GUPTA, Lokesh. *What is REST* [online]. 2022. Dostupné také z: https://restfulapi.net.
23. GUPTA, Lokesh. *HTTP Methods* [online]. 2021. Dostupné také z: https://restfulapi.net/http-methods.
24. BERNERS-LEE, Tim. *URL* [online]. 1211 Geneva 23, Switzerland: World-Wide Web project CERN, [b.r.]. Dostupné také z: https://www.w3.org/Addressing/URL/uri-spec.html.
25. GUPTA, Lokesh. *HTTP Status Codes* [online]. 2021. Dostupné také z: https://restfulapi.net/http-status-codes/.
26. *Standard ECMA-404: The JSON data interchange syntax* [online]. 2. vyd. Ecma International, 2017. Dostupné také z: https://www.ecma-international.org/publications-and-standards/standards/ecma-404.
27. *Introducing JSON* [online]. [B.r.]. Dostupné také z: https://www.json.org/json-en.html.
28. SPARX SYSTEMS PTY LTD. *ENTERPRISE ARCHITECT* [online]. [B.r.]. Dostupné také z: https://sparxsystems.com/products/ea/index.html.
29. ORACLE CORPORATION. *SQL Developer* [online]. [B.r.]. Dostupné také z: https://www.oracle.com/cz/database/sqldeveloper.
30. JETBRAINS S. R. O. *What is IntelliJ IDEA* [online]. [B.r.]. Dostupné také z: https://www.jetbrains.com/idea/features.
31. *About DBeaver* [online]. [B.r.]. Dostupné také z: https://github.com/dbeaver/dbeaver/wiki.
32. SCOTT CHACON, Ben Straub. Pro Git. In: 1 New York Plaza, New York, NY 10004: Apress Media, LLC, 2012, s. 10–25. ISBN 978-1-4842-0077-3.

33. *Java Play - Managing database evolutions* [online]. [B.r.]. Dostupné také z: https://www.playframework.com/documentation/2.8.x/Evolutions.

34. *Package javax.persistence* [online]. Oracle, [b.r.]. Dostupné také z: https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html.

35. JANSSEN, Thorben. *Naming Strategies in Hibernate 5* [online]. [B.r.]. Dostupné také z: https://thorben-janssen.com/naming-strategies-in-hibernate-5/%20#CamelCaseToUnderscoresNamingStrategy_in_Hibernate_554.

36. ANGUSWAMY, Reghu; FRAKES, William B. Reuse design principles. *International Journal of Software Engineering and Knowledge Engineering, IJSEKE* [online]. 2012. Dostupné také z: https://www.researchgate.net/profile/William-Frakes/publication/236868092_Reuse_Design_Principles/links/02e7e519c0306adcc2000000/Reuse-Design-Principles.pdf.

37. SPINELLIS, Diomidis. Code Documentation. *IEEE Software*. 2010, roč. 27, č. 4, s. 18–19. Dostupné z DOI: 10.1109/MS.2010.95.

38. STUEBEN, Michael. Self-Documenting Code. In: *Good Habits for Great Coding: Improving Programming Skills with Examples in Python*. Berkeley, CA: Apress, 2018, s. 67–90. ISBN 978-1-4842-3459-4. Dostupné z DOI: 10.1007/978-1-4842-3459-4_6.

39. EVERETT, Gerald D; MCLEOD JR, Raymond. Software Testing. In: John Wiley & Sons, Inc., Hoboken, New Jersey, 2007, s. 29–31. ISBN 978-0-471-79371-7.

40. KEET, C. Maria; ŁAWRYNOWICZ, Agnieszka. Test-Driven Development of Ontologies. In: SACK, Harald; BLOMQVIST, Eva; D'AQUIN, Mathieu; GHIDINI, Chiara; PONZETTO, Simone Paolo; LANGE, Christoph (ed.). *The Semantic Web. Latest Advances and New Domains*. Cham: Springer International Publishing, 2016, s. 642–657. ISBN 978-3-319-34129-3.

41. SPINELLIS, Diomidis. State-of-the-Art Software Testing. *IEEE Software*. 2017, roč. 34, č. 5, s. 4–6. Dostupné z DOI: 10.1109/MS.2017.3571564.

42. HAMILTON, Thomas. *What is Static Testing? What is a Testing Review?* [Online]. [B.r.]. Dostupné také z: https://www.guru99.com/testing-review.html.

43. JETBRAINS S.R.O. *Code inspections* [online]. [B.r.]. Dostupné také z: https://www.jetbrains.com/help/idea/code-inspection.html.

44. FREEMAN, Adam. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. In: 1 New York Plaza, New York, NY 10004: Apress Media, LLC, 2012, s. 121–136. ISBN 978-1-4842-7956-4.

# Obsah přiloženého média

```
readme.txt ..... stručný popis obsahu média
src
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    BP_Jan_Pospíšil_2022.pdf ..... text práce ve formátu PDF
```