FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozšíření webové aplikace pro správce systému sdílení automobilů
Student:	Jiří Gutwirth
Vedoucí:	Ing. Václav Jirovský, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Navrhněte a implementujte rozšíření webové aplikace Uniqway o podporu kontroly vozidel správci služby.

• Společně s vedoucím práce a ostatními členy týmu proveďte detailní specifikaci požadavků na rozšíření webové aplikace o dokumentaci z pravidelných kontrol vozidel, zobrazování přehledu kontrol a případně dalších identifikovaných aktivit.

• Analyzujte možnosti a technická omezení stávající webové aplikace pro uživatele systému sdílení automobilů.

• Pomocí metod softwarového inženýrství proveďte návrh rozšíření aplikace včetně návrhu uživatelského rozhraní.

• Implementujte navržené rozšíření do již existující webové aplikace.

• Navrhněte API pro serverovou část aplikace, se kterou bude daná aplikace komunikovat.

• Aplikaci vhodným způsobem otestujte a dobře zdokumentujte.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 18. listopadu 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozšíření webové aplikace pro správce systému sdílení automobilů

Jiří Gutwirth

Katedra softwarového inženýrství
Vedoucí práce: Ing. Václav Jirovský, Ph.D.

9. května 2021

Poděkování

Děkuji svému vedoucímu Ing. Václavu Jirovskému, Ph.D. za možnost podílet se v rámci práce na zajímavém projektu, za vstřícnost po celou dobu přípravy práce, ochotu kdykoli pomoci a bryskní odezvu na kladené dotazy. Dále děkuji Bc. Štěpánu Severovi, vedoucímu technického týmu Uniqway, za jeho čas, rady a dlouhodobou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, 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.

Tímto má předložená práce nemůže být zveřejněna po dobu platnosti závazku mlčenlivosti, tj. do 13. května 2024.

V Praze dne 9. května 2021 . . . . . .

Odkaz na tuto práci

Gutwirth, Jiří. *Rozšíření webové aplikace pro správce systému sdílení automobilů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá analýzou současné klientské webové aplikace systému sdílení automobilů Uniqway, návrhem a implementací rozšíření aplikace o podporu kontroly vozidel správci služby. Cílem je zrychlit a zjednodušit proces kontroly, poskytnout platformu pro správu kontrol a aktuálních poškození vozidel.

Práce začíná literární rešerší uvádějící čtenáře do problematiky vývoje webových aplikací. Teoretická část je zakončena rozborem současné webové aplikace a identifikací možností k jejímu rozšíření. Následuje návrh rozšíření a po něm již samotná implementace. K vypracování je použit Javascriptový framework Vue.js. Výsledkem je funkční a otestované uživatelské rozhraní připravené k nasazení.

Hlavním přínosem této práce je funkcionalita aplikace umožňující efektivnější provedení kontrol a zjednodušení správy aktuálního stavu vozidel.

Klíčová slova Uniqway, webová aplikace, analýza webové aplikace, frontend, návrh uživatelského rozhraní, sdílení automobilů, kontrola automobilu, Javascript, Vue.js

Abstract

This thesis deals with analysis of the current client web application of the Uniqway carsharing system and pursues design as well as implementation of an extension that provides car control management functionality. The aim is to quicken and simplify the process of car checking and to provide a control and defect management platform.

The work starts with literature review which introduces the web application development problematics to the reader. The theoretical part ends with an investigation of the current web application and identification of the ways how to expand it. Then the extension is designed, followed by its implementation. The whole project is created using the Vue.js Javascript framework. The output of the project is a tested and functional user interface ready for deployment.

The main outcome of this thesis is an application functionality that allows more effective car control process as well as easier vehicle condition management.

Keywords Uniqway, web application, web application analysis, front-end, user interface design, carsharing, car check, Javascript, Vue.js

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Sdílení automobilů, tzv. carsharing, pomalu, ale jistě nabývá na popularitě. Existuje pro to hned několik opodstatnění. Masivní technologický a ekonomický pokrok posledních desetiletí zapříčinil přehlcení větších měst auty. Vyhledání parkovacího místa pro vůz je stále problematičtější, jeho provoz je drahý a ekologické dopady současnosti není možné vyčíslit.

V současnosti je (nejen) v České republice velmi vysoký poměr počtu automobilů na počet lidí. Reálný počet aut nutných k uspokojení potřeb lidí je ale mnohem nižší. Využitím této okolnosti se zabývá právě carsharing. V poslední době strmě roste počet lidí, kteří si vozidlo raději na několik hodin týdně, kdy ho opravdu upotřebí, půjčí. Po splnění účelu (např. nákupu či přestěhování věcí) opět auto vrátí, čímž ho dají k dispozici ostatním uživatelům. Odpadnou starosti s údržbou vozu i náklady na jeho provoz. Tím se uleví městům, jejich obyvatelům i planetě jako celku.

Společností provozujících carsharing v České republice funguje několik, Uniqway je ale první čistě studentskou iniciativou v této oblasti. Služba oficiálně vznikla v říjnu 2018 spojením sil tří českých univerzit, a sice ČVUT, VŠE a ČZU. Celý projekt udržují a rozvíjejí pouze studenti, od správy a vývoje aplikací, přes marketing a komunikaci s uživateli, až po údržbu a správu aut samotných.

A právě řízení správy aut, zejména management kontrol a udržování řádného technického stavu, není v současné době bez komplikací. Správci nemají k dispozici nástroj pro evidenci kontrol, který by jim přehledně zobrazil aktuální poškození a problémy aut.

Tato práce se zabývá řešením zmíněného nedostatku. První kapitolu tvoří literární rešerše, která uvádí čtenáře do problematiky a definuje potřebné pojmy. Závěrem této kapitoly je analýza cílové webové aplikace. Následně je proveden návrh struktury aplikace a vymezen rozsah funkcionality. Posledním krokem je implementace podle designu a řádné otestování. Výsledkem je plně funkční webové rozhraní pro správu automobilů, připravené k nasazení.

Kapitola 1

Cíl práce

Hlavním cílem této práce je návrh rozšíření současné klientské webové aplikace Uniqway o funkcionalitu pro správce služeb a jeho implementace. Samotnému návrhu předchází analýza současného stavu aplikace a identifikace místa k rozšíření.

Dalším cílem je design struktury rozšíření se zaměřením na proces rutinní kontroly vozidla. V rámci toho je třeba definovat jednotný způsob lokalizace poškození vozidla, výčet vybavení, jehož přítomnost v autě se kontroluje, a také seznam ostatních aspektů kontroly týkajících se technického stavu vozidla. Při návrhu uživatelského rozhraní je potřeba klást důraz na rychlost a zároveň dbát na jednoduchost celého procesu kontroly, neboť tyto aspekty hrají spolu s kvalitou kontroly hlavní roli.

V neposlední řadě je cílem také implementované rozšíření patřičně otestovat a zdokumentovat. Posledním cílem je navrhnout možná budoucí vylepšení celého rozhraní.

Kapitola 2

Web a webové aplikace

V této kapitole bude v rámci úvodu do problematiky postupně představen pojem aplikace a rozlišeny jednotlivé její druhy. Následně budou specifikovány určité aspekty webových aplikací, které pomohou pochopit činnost webu a pojem webových aplikací samotný. V závěru budou popsány webové aplikace udržované a vyvíjené technickým týmem společnosti Uniqway.

Pod pojmem „web“ si dnes pravděpodobně většina lidí představí rychlou cestu získání informací, moderní stránky nesoucí informace z druhého konce planety a propracované aplikace, které byly donedávna dostupné, jen pokud byl zájemce ochoten zaplatit licenci, obětovat místo na disku a nainstalovat si ji na osobní počítač. Tuto podobu na sebe web vzal ale až v posledních letech.

Slovo „web“ je zkrácenou verzí „World Wide Web“, tedy jakéhosi celosvětového informačního systému poskytujícího zdroje na internetu. „Internet“ značí globální síť tvořenou propojenými komunikujícími menšími sítěmi koncových zařízení typu počítač nebo mobilní telefon. Přestože má internet kořeny již v 60. letech 20. století, oficiálním datem vytvoření internetu je 1. 1. 1983. Do té doby byl využíván především pro vojenské účely [1].

2.1 Aplikace obecně

Popularizace světa počítačů a informačních technologií s sebou přinesla mimo jiné obrovský tržní potenciál. Bezprostředně poté, co se technologie staly finančně dostupnými pro širokou veřejnost, přišla obří vlna pokroku ve světě aplikací. Pojem aplikace označuje balíček softwarových funkcí, které nějakým způsobem mění život uživatele k lepšímu. Ať už nám pomohou najít autobusové spojení, objednat jídlo či pustit oblíbenou hudbu. Na následujících řádcích budou rozebrány jednotlivé druhy aplikací.

2.1.1 Konzolové aplikace

Nejstarším druhem aplikací jsou aplikace konzolové. Konzolové proto, že těmto aplikacím chybí grafické uživatelské rozhraní (GUI - *graphical user interface*, tedy formálně řečeno prostředky, kterými uživatel ovládá zařízení a které obsahují grafické prvky jako okna, ikony či tlačítka [2], [3]). Komunikují s uživatelem pouze pomocí příkazového řádku, takzvané „konzole“. Do této konzole uživatel píše příkazy, které vykonávají jednotlivé funkce programu. Jistě si každý dovede představit, že tento typ aplikací je jen pro opravdové nadšence. V dobách vzniku konzolových aplikací byl ještě svět počítačů veřejnosti poměrně uzavřen. Pojmy jako „uživatelská přívětivost“ nebo „uživatelská zkušenost“ neexistovaly. Výhoda tohoto druhu aplikací spočívá v tom, že její vývoj je kratší o fázi programování uživatelského rozhraní. Není tedy pravda, že by se již konzolové aplikace nevyužívaly, pouze nejde o software, který je po vývoji určen zákazníkům z řad široké veřejnosti.

Protože ale již v dobách konzolových aplikací bylo zřejmé, že se počítače dříve či později dostanou i k lidem bez odborného vzdělání v oblasti informačních technologií, bylo potřeba vymyslet cestu, jak funkcionalitu aplikací vizualizovat a prolomit tak ledy mezi širokou veřejností a počítači.

2.1.2 Desktopové aplikace

Již v roce 1979 ve firmě Xerox vznikl první prototyp osobního počítače s obrazovkou a primitivním grafickým uživatelským rozhraním. Tento počítač však ještě nebyl určen pro veřejnost. Při své prohlídce společností Xerox si ale počítače s jednoduchým GUI všiml Steve Jobs, který následně ve své společnosti Apple vytvořil počítač Lisa, historicky první počítač s GUI určený veřejnosti [4].

Tímto začal vývoj takzvaných desktopových aplikací, tedy aplikací instalovaných a běžících na osobním počítači, řízených uživatelem za pomoci prvků uživatelského rozhraní. Byť je primárně tento typ aplikací určen pro osobní lokální potřebu, lze aplikace na desktop spojit se sítí a čerpat tak data z internetu. Mezi nejoblíbenější desktop software patří v poslední době různé textové, tabulkové a grafické editory, programy na práci s videozáznamy či internetové prohlížeče.

Přestože se mnoho softwaru v posledních letech přesouvá z desktopové formy do formy webové aplikace, některé aplikace budou mít i v budoucnu na lokálním počítači své místo. Jde zejména o programy, ve kterých hraje klíčovou roli výkon počítače, tedy zejména zmíněné grafické a video editory nebo například počítačové hry.

Nevýhodou desktopových aplikací je náročný vývoj. Pro multiplatformnost – tedy aby aplikace byla dostupná na více zařízení s různými druhy operačních systémů – je potřeba vyvinout aplikaci na všechny používané druhy operačních

systémů jednotlivě. Tento druh vývoje je finančně i technicky náročný a totéž platí pro následnou údržbu a rozvoj aplikace.

2.1.3 Mobilní aplikace

Jak již název napovídá, mobilní aplikace jsou aplikace instalované a provozované pomocí mobilního telefonu. S bouřlivým technologickým vývojem posledních let doznaly obřího rozvoje právě i mobilní telefony. Do telefonů se podařilo dostat výkonné procesory, dostatečný objem operační paměti (též RAM – *random access memory* – rychlé úložiště držící data aktuálně spuštěných aplikací po dobu zapnutého zařízení [5]) a hlavně operační systém. Mobilní telefony se v podstatě začaly chovat jako slabší, přenosné počítače. Protože svými možnostmi daleko převyšovaly telefony bez operačních systémů a velkých objemů RAM, začaly být označovány za „chytré“. Díky tomu se pro takováto zařízení později vžilo označení „smartphone“.

Mobilní aplikace využívají plně výhod přenositelnosti telefonu a faktu, že moderní člověk má smartphone neustále u sebe. I proto nejvíce na telefonech našly uplatnění aplikace umožňující snadnou a rychlou komunikaci s ostatními uživateli – sociální sítě [6]. Mezi dalšími oblíbenými aplikacemi jsou například aplikace využívající k nejrůznějším účelům polohu telefonu na mapě nebo aplikace usnadňující správu hudby v telefonu.

I mobilní aplikace ale mají své limity. Na předním místě jde o velikost a výkon telefonu. Aplikace musí být stavěny velmi intuitivně. Hlavní prioritou je jednoduché a plynulé ovládání. Kvůli potřebě jednoduchosti ale fakticky nelze poskytnout nějakou složitější funkcionalitu typu ovládání informačního systému či pokročilé editace audiovizuálního obsahu. Výkon telefonu, byť strmě rostoucí, stále nestačí na komplexní aplikace, jejichž potřeba výkonu se také neustále zvyšuje.

2.2 Webové aplikace

Aplikacím běžícím ve webovém prohlížeči se říká webové aplikace. Většinou jsou tvořeny klientskou a serverovou částí. Klientská část je na straně prohlížeče, který pomocí požadavků a odpovědí komunikuje s programem spuštěným na serveru, jenž pak zajišťuje aplikační logiku a správu dat ukládaných v databázi.

Tato kapitola popisuje některé technické charakteristiky webových aplikací. Seznamuje s druhy webových aplikací, zevrubně popisuje, jak spolu aplikace komunikují a následně představuje nejběžnější jazyky, ve kterých se webové aplikace klientské strany a jejich rozšíření programují.

2.2.1 Výhody a nevýhody webových aplikací

Nespornou výhodou webových aplikací ve vztahu k ostatním druhům aplikací je automatická multiplatformnost. Protože aplikace běží v prohlížeči, mohou fungovat v podstatě kdekoliv. Na mobilním telefonu i na počítači s jakýmkoliv operačním systémem. Stačí tedy vyvinout jednu verzi aplikace, která je pak kompatibilní se všemi cílovými zařízeními.

Nevýhodou tohoto druhu aplikací je limitovaný výkon. Webové aplikace nemají přímo k dispozici veškerý výkon počítače, mohou využít pouze část výkonu určenou pro webový prohlížeč. Tento problém je stále aktuální i přesto, že již vznikají technologie, které ho tlačí do pozadí. Proto mají (alespoň na klientské – uživatelově straně) webové aplikace tendenci plnit spíše úkony méně náročného charakteru. Další omezení vyplývá z faktu, že rychlost webové aplikace z velké části závisí na rychlosti internetového připojení. I když i tento problém je, díky rozvoji poskytovatelů internetu, kvality sítě a územního pokrytí, spíše na ústupu.

2.2.2 Komunikace na webu

S rozšiřováním webu a zvyšováním počtu subjektů a jejich druhů na něm figurujících vznikla potřeba komunikace mezi těmito subjekty. Kvůli vzájemné nekompatibilitě jednotlivých technologií ale jednoduchá komunikace nebyla možná, aplikace si nerozuměly a neexistovalo jednotné rozhraní, pomocí kterého by navzájem aplikace používaly své funkce. Vznikla potřeba vytvořit standardizovaná pravidla komunikace, nezávislá na platformě zdroje a cíle zpráv.

Díky této myšlence vznikly tzv. webové služby. Podle [7] jsou webové služby „technologií, která umožňuje integrovat libovolné aplikace provozované na různých platformách a ovládat je prostřednictvím webového rozhraní“. Typická webová služba zahrnuje 3 základní komponenty: *kontrakt* – definici služby, vstupy a výstupy, omezení a parametry komunikace, dále *producenta*, tedy účastníka komunikace poskytujícího službu a informace a *konzumenta* – klienta žádajícího informace, uživatele poskytnuté služby [8]. V souvislosti s pojmem „web service“, tedy webová služba, vzniklo několik standardů (souborů pravidel) definujících určité obecné zásady formátu komunikace.

Prvním z nich je **SOAP** (*simple object access protocol*). Podle [9] je SOAP základem webových služeb. Jde o protokol, ve kterém jsou úkony posílání informací a volání vzdálených funkcí prováděny pomocí XML zpráv, tedy zpráv založených na *extensible markup language*, což je počítačově čitelný značkovací jazyk. Tento typ zpráv je obecně velmi bohatý na informace, pro člověka složitý na čtení a vhodný pro komunikaci, kde je exaktní definice všech parametrů nutností. Typicky je tedy používán tento typ služeb například v oblastech bankovnictví a pojišťovnictví. Samotné služby jsou zde definované pomocí takzvaného WSDL – *web services description language*. Jak zkratka napovídá,

jde o jazyk přímo vytvořený k důkladnému popisu webové služby. Tento jazyk je také založen na XML. Ukázka 2.1 znázorňuje strukturu XML kódu a byla převzata z [10].

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Zdrojový kód 2.1: Ukázka kódu v jazyce XML

Druhým z nejčastějších přístupů ke komunikaci na webu je **REST**, neboli *representational state transfer*. Dle [11] je REST sada omezení, díky kterým se u webové služby zvýší výkon, škálovatelnost a modifikovatelnost. Tento typ webových služeb se vyznačuje prací se „zdroji“, které reprezentují stav aplikace (od toho také název). Jsou určeny pomocí URI (*uniform resource identifier* – jednoznačný identifikátor zdroje na internetu [12]). Tyto zdroje mezi sebou pak interagují (pro všechny zdroje jednotně) za pomoci čtyř základních operací: create, read, update, delete. REST v praxi komunikuje nejčastěji zprávami ve formátu JSON. Zkratka JSON označuje *javascript object notation*, tedy notace objektů z jazyka Javascript. Objekty v jazyce Javascript jsou struktury slovníkového typu obsahující páry klíč-hodnota. Podrobněji budou popsány v kapitole 2.2.3 - Frontend a backend.

Důležitým protokolem pro komunikaci na webu, který nelze opomenout je **HTTP** (*hypertext transfer protocol*). Oba výše popsané standardy pracují s HTTP. Je to textový bezestavový protokol působící na aplikační vrstvě ISO/OSI modelu (referenčním modelu rozdělujícím komunikaci do 7 základních vrstev [13]) [14]. Bezestavovost protokolu znamená, že si server neuchovává mezi dvěma požadavky žádné informace (žádný stav), požadavky jsou na sobě zcela nezávislé. Základní struktura požadavku je podle [15] následující:

- HTTP metoda (tedy specifikace, zda jde o `GET`, `POST`, `PUT`, `DELETE` aj.),
- URL (*uniform resource locator*) zdroje,
- verze HTTP protokolu,
- volitelná část hlavičky,
- tělo požadavku (u některých metod, např. `GET`, prázdné),

HTTP pracuje způsobem požadavek-odpověď. Požadavky se nesou na vlnách několika metod nabízejících veškerou potřebnou funkcionalitu. Nejběžnější metodou je `GET`, mezi ostatní základní pak `POST`, `PUT`, `DELETE`. Existují i další, avšak tyto jsou nejpoužívanější. Každá z metod má jinou sémantiku, ale některé vlastnosti jsou pro více metod společné. Jednou z nich například je *idempotence*, vlastnost, kdy může být dotaz na server poslán jednou či vícekrát a pokaždé dostane stejný výsledek. Také požadavek idempotentní metody zanechá server ve stejném stavu, jako byl před vyřízením požadavku [16]. Další společnou vlastností více metod je bezpečnost. Podle [17] je bezpečná taková metoda, která nijak nezmění stav serveru, který obdržel požadavek tohoto typu. Jinými slovy jde o operaci, která je takzvaně „read-only“, tedy je určená pouze a jenom ke čtení dat, nikdy ke změně. Poslední častou vlastností je *cachability* – „cachovatelnost“ [kešovatelnost]. Podle [18] se slovem *cache* označuje úložiště, do kterého si prohlížeč schová lokální kopii některých informací o navštíveném webu. Opětovné načtení stránky je tak rychlejší tím, že se již nemusí ze serveru nahrávat celá. Mezi ukládané informace patří i dotčené HTTP metody, ty jsou pak nazývány jako „cachovatelné“ [19].

`GET` je typem požadavku, který je odesílán při žádosti o nějaké informace. Pomocí této metody by se měla pouze dostávat data (neměla by být tedy využívána k jakékoliv změně zdroje). Příkladem použití budiž přesměrování na stránku s předpovědí počasí, tento příklad je názorně předveden v ukázce 2.2. Metoda `GET` patří mezi metody bezpečné, idempotentní i cachovatelné.

```
GET https://pocasi.cz/ HTTP/1.1
```

Zdrojový kód 2.2: HTTP GET požadavek

Požadavek typu `POST` již zdroj mění, přesněji přidává do zdroje nějaký záznam či jej aktualizuje, součástí této metody jsou informace, které mají být do zdroje přidané. Nejde o metodu idempotentní, cachovatelnou ani bezpečnou. Typické použití `POST` request je při odesílání vyplněného formuláře.

`PUT` metoda je užívána ke kompletní výměně nebo tvorbě zdroje. S ní je (v těle metody) samozřejmě také posílán cílový stav zdroje (nové informace). Rozdíl mezi `PUT` a `POST` je v tom, že metoda `PUT` je idempotentní. Lze si to představit na příkladu, kdy je pomocí metody `POST` na serveru aktualizováno každý den datum tím, že se ke dnům přičte 1. Metodou `PUT` je ale datum každý den zcela vyměněno za nové, je tedy aktualizovaný celý zdroj, nikoliv pozměněn 1 jeho parametr. Odpověď serveru je tedy pokaždé stejná. Bezpečnost ani cachovatelnost ale mezi její vlastnosti nepatří.

Metoda `DELETE` slouží ke smazání zdroje. Zdroj je smazán kompletně, vždy je tedy odpověď serveru stejná. Stav serveru se mění, `DELETE` není mezi bezpečnými metodami. Tento typ metody není ani „cachován“.

2.2.3 Frontend a backend

Dalšími hojně využívanými pojmy z oblasti webového inženýrství jsou termíny *frontend* a *backend*. Pro oba termíny se dá také najít společné heslo *full-stack*. To označuje kompletní pokrytí logiky webové aplikace. Full-stack vývojář je zodpovědný za část backendu i frontendu. Podle [20] je vývoj front-end části webu definován jako proces produkce HTML, CSS a Javascriptu pro webovou stránku nebo aplikaci, aby s ní mohl uživatel přímo interagovat. Jde tedy o viditelnou část aplikace, na kterou uživatel kliká a která mu zobrazuje informace. Tyto informace pak poskytuje backend. Tak se označuje část aplikace běžící na serveru [21]. Tato část většinou řídí složitější logiku aplikace (například různé výpočty), přístup do databáze, balancování zátěže aplikace apod.

V dnešní době mají již v podstatě všechny aplikace dedikovaný backend a frontend odděleně. Některé webové programy by se teoreticky obešly bez backendu, pouze s frontendovou částí. Příkladem by mohla být jednoduchá klikací hra v prohlížeči na styl populární hry Flappy Bird, tedy aplikace s primitivním ovládáním (uživatel kliká a určuje tím rytmus skoků animovaného ptáčka) a bez potřeby složité logiky či perzistence dat (ve své podstatě). Moderní aplikace dnes již ale nezaujmou uživatele pouze možností si zahrát. Běžnou součástí her je funkce uložení výsledků, přihlášení se ke svému účtu a porovnávání průběžného skóre s ostatními hráči. K tomu všemu je potřeba server, a tedy backend. Data se ukládají do databáze, se kterou server pracuje. Server též řídí logiku autentizace na webu. Znovu je tak na místě připomenout, že rychlý posun techniky vpřed (ve vztahu k předchozímu příkladu zapříčiňující stále vyšší a vyšší očekávání uživatele) vyžaduje složitější, propracovanější aplikace a rozdělení mechanismů programu do logických celků. Základními nástroji v oblasti vývoje klientské strany webových aplikací jsou HTML, CSS a Javascript.

HTML – neboli *hypertext markup language* – je hypertextový značkovací jazyk. Hypertextový je proto, že spolu HTML stránky komunikují přes hypertextové odkazy. Značkovací je kvůli své syntaxi. HTML soubory nesou obsah webové stránky. Skládají se ze samotného obsahu – textu, obrázků a podobně, a ze značek, které tento obsah ohraničují. Obsah spolu se značkami je nazýván elementem. Element je tedy definován otevírací a zavírací značkou. Výjimkou jsou elementy, které jsou tvořeny pouze otevíracím tagem a nemají tag zavírací. Elementy poté tvoří stromovou, počítačově čitelnou strukturu [22] tím, že jsou do sebe zanořovány a větveny. Na ukázce 2.3 jsou vidět oba typy HTML elementu. Prvním z nich je nadpis, klasický typ elementu s dvěma značkami, otevírací i zavírací. Druhým je příklad „samozavíracího“ elementu `img`.

Kaskádové styly (CSS – *cascading style sheets*) představují podle [23] jednoduchý mechanismus, jak přidat styly (fonty, barvy, řádkování,...) do webových dokumentů. V podstatě jde o jazyk, pomocí kterého je definován vzhled webové stránky. Slovy „webový dokument“ je myšlen HTML soubor webové stránky.

```
<h1>Nadpis</h1>
<img src="foto.png" alt="Foto">
```

Zdrojový kód 2.3: HTML element

Tento jazyk pracuje na bázi kaskád. Kaskády způsobují vrstvení definic stylů různých aspektů stránky. Vyšší vrstvy propisují definované styly do nižších, obsahují obecnější styly týkající se větších částí stránky. V nižších vrstvách je následně možné styly upravit a přizpůsobit pro jednotlivé elementy.

CSS vybírá jednotlivé elementy či jejich skupiny z HTML souborů pomocí selektorů. Selektor specifikuje nějaký znak (jméno elementu, jméno třídy, identifikátor, atribut elementů,...), který označuje určitou skupinu elementů. Následně jsou v bloku definovány styly postihující danou skupinu. Tyto styly jsou podány formou pravidel, která se skládají z vlastnosti (například `color` – barva písma) a hodnoty (např. `red`). Příklad definice stylu v CSS je možné nahlédnout v ukázce 2.4. Selektorem je vybrán z HTML stránky nadpis první úrovně `h1`, na kterém jsou následně provedeny úpravy. Barva písma `color` je nastavena na černou, `background-color` – barva pozadí – je nastavena na bílo.

```
h1 {
    color: black;
    background-color: white;
}
```

Zdrojový kód 2.4: CSS definice stylů

Dle [24] je **Javascript** jazyk navržený pro práci na klientské straně webových aplikací. Jde o nejpopulárnější jazyk na světě, od roku 2017 se k vývojářům v tomto jazyce přidalo dalších 5 milionů lidí, celkem je jich kolem 12.5 milionu [25]. Zajímavostí je nulový vztah k populárnímu programovacímu jazyku Java. Tyto dva jazyky sdílejí pouze a jenom část názvu. Javascript je dynamicky typovaný skriptovací jazyk. Dynamické typování vychází z faktu, že datové typy proměnných ve skriptech se určují až za běhu javascriptové aplikace [26]. Slouží k přidání interakcí a chování do webových stránek. Jak již bylo řečeno, tento jazyk se používá hlavně pro klientskou část programů. Javascript je objektový jazyk, ke svému fungování využívá objekty – struktury slovníkového typu, tedy složenými závorkami ohraničený seznam dvojic typu klíč-hodnota. Klíčem může být libovolný řetězec a hodnotou instance jakéhokoliv datového typu (i jiný objekt). To umožňuje tvořit a zanořovat

složité, avšak strukturou jednotné datové bloky [27]. Příklad JSON objektu znázorňuje ukázka 2.5, ve které je vidět objekt představující datum.

```
{
    "day": 1,
    "month": 12,
    "year": 2002
}
```

Zdrojový kód 2.5: Příklad formátu JSON

K tomuto jazyku vzniklo již mnoho doplňků – knihoven a frameworků. Dle [28] je framework definován jako kolekce javascriptových knihoven – tedy sad předepsaných funkcí, které ulehčují programování rutinních vlastností aplikací. Právě tyto frameworky stojí za jeho enormní popularitou. Samotný Javascript je díky tomu obohacen o funkcionalitu specificky vyhovující (nejen) přednímu konci webových aplikací. V kapitole 2.2.5 budou popsány 3 základní, nejpopulárnější frameworky a zvláštní důraz bude kladen na popis fungování frameworku Vue.js, který byl použit pro implementační část této práce.

Javascript je jakožto technologie provázána s technologií **AJAX**, tedy *asynchronous javascript and xml*. AJAX umožňuje, jak zkratka napovídá, asynchronní komunikaci se serverem. K této komunikaci není nutné znovu načítat celou HTML stránku, lze vyměnit či obnovit jen změnou dotčené části [29]. Tímto stylem je možné webové aplikaci (aniž by to uživatel zaregistroval) posílat potřebné informace v mnoha formátech. Je to výhodné zejména pro aplikace typu SPA, kde je eliminace potřeby načtení nové HTML stránky pro změnu části dat hlavním přínosem.

2.2.4 MPA versus SPA

MPA – *multiple page applications* – je tradiční cestou k vývoji webu. Principiálně se jedná o webovou aplikaci, ve které je každá akce uživatele směřující ke změně obsahu následovaná opětovným nahráním celé webové stránky ze serveru do prohlížeče uživatele [30]. Tento přístup má své výhody. Mezi ně patří například fakt, že prvotní nahrání stránky je rychlejší a snadněji se u tohoto typu aplikací řídí SEO (*search engine optimization* – techniky umožňující růst ve výsledkových listinách vyhledávačů [31])

Díky načítání nových stránek při každé změně je možné provést optimalizaci přizpůsobenou pro každou ze stránek zvlášť. Hlavní nevýhodou je ale skutečnost, že se při jakékoliv změně na stránce načítá celá stránka kompletně znovu. To představuje výrazné zpomalení a obecně (v mnoha případech zby-

tečně velkou) zátěž pro všechny účastníky komunikace – prohlížeč, server, internetové připojení a další.

Alternativou k MPA je nový přístup k tvorbě aplikací – SPA, *single page application*. Toto jsou aplikace, u kterých si uživatel do prohlížeče prvotně načte celou stránku a jakékoliv změny jsou poté prováděny výměnou pouze dotčených částí stránky bez potřeby celou stránku načítat kompletně znovu [32]. Výhodou tohoto přístupu je zlepšení uživatelské přívětivosti (odezva je plynulejší, nikdy není totálně přerušen kontakt s uživatelem) a celkové rychlosti práce aplikace. SPA aplikace mají tendenci držet logiku aplikace na straně klienta, kam je sice nejdříve pomaleji načtena, avšak rychleji dostupná v průběhu užívání aplikace. Následně se v případě nutnosti aplikace dotáže serveru na nová data a načte novou část stránky. Dotazování probíhá za pomoci rozhraní API (*application programming interface* – sada příkazů, funkcí, protokolů a objektů, pomocí kterých program interaguje s externím systémem [33]). Jde o rozhraní poskytnuté serverem k privátnímu či veřejnému dotazování na data pomocí HTTP požadavků.

I tento přístup má ale svá negativa. Mezi ně patří například (v drtivé většině případů) nutnost zapnutého Javascriptu v prohlížeči. SPA aplikace jsou také náchylnější k útokům typu XSS (*cross-site scripting*). V neposlední řadě je u SPA aplikací složitější řízení SEO, což je způsobeno tím, že se nenačítají stránky pokaždé znovu kompletně a není tak možné například dostatečně diverzifikovat seznam klíčových slov, se kterými pak pracují vyhledávače.

2.2.5 Porovnání SPA frameworků

Tři nejoblíbenější javascriptové SPA frameworky jsou v současné době Angular, React a Vue, někdy také uváděné jako Angular.js, React.js či Vue.js. V této kapitole budou všechny 3 krátce rozebrány a kapitola 2.3 pak bude zaměřena speciálně na Vue.js, jelikož jde o framework, který byl použit v této práci.

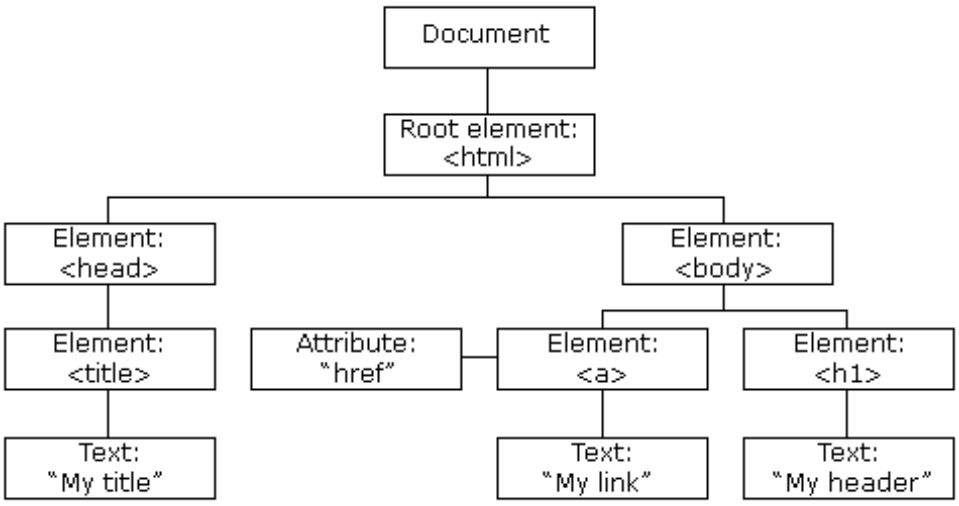
Angular je javascriptový framework založený na jazyce Typescript – javascriptové nadstavbě, která čistý Javascript rozšiřuje o statické typování. Vznikl ve firmě Google a ze zmíněných tří frameworků je nejstarší. Aplikace psané v Angular jsou členěné do komponent, tedy částí kódu sdružujících nějakou funkcionalitu. Tyto komponenty jsou následně skládány do kolekcí komponent, modulů `NGModules`. Aplikace musí mít minimálně jeden kořenový modul. Komponenty definují `views` – sadu obrazových elementů, ze kterých si může Angular vybrat podle logiky programu [34].

Ze tří zmíněných frameworků je Angular nejobtížnější k ovládnutí. To je dáno zejména faktem, že jde o kompletní MVC framework. Zkratka MVC představuje *model, view, controller* – architekturu, ve které jsou kompetence pro oblasti rozděleny mezi *model*, který se stará o data a přístup k databázi, *view*, které pracuje se zobrazováním a *controller*, který má roli organizátora komunikace mezi *view* a *model* a stará se o business logiku aplikace [35]. Díky

této vlastnosti frameworku Angular je ale také při jeho znalosti snadné porozumět ostatním dvěma a obecně fungování oblasti vývoje frontendu. Další výraznou vlastností frameworku Angular je skutečnost, že implicitně využívá obousměrné provázání dat (*two-way data binding*), tedy data proudí oběma směry mezi rodičovskými komponentami a komponentami potomků, data se tedy dají aktualizovat z obou stran [36].

React je frameworkem, který mezi třemi zmíněnými vede statistiky na Github, má nejvíce přispěvatelů i sledujících [36]. Byl vytvořen ve společnosti Facebook. Zajímavostí Reactu je cílená kombinace jazyků v jednom kusu kódu. Kombinuje se HTML a Javascript, HTML část definuje šablonu pro komponentu a Javascript jí dodá funkcionalitu, to vše v jednom bloku. Na rozdíl od Angular není React kompletním frameworkem, k velkému množství funkcionality je potřeba doinstalovat knihovny třetích stran. Díky tomu je ale jeho ovládnutí značně jednodušší. React využívá technologie virtuálního DOM.

DOM, neboli *document object model*, je podle [37] programové rozhraní pro HTML. Jedná se o stromovou strukturu, která zobrazuje HTML elementy jako objekty a poskytuje metody k jejich přístupu a úpravě. Stejnou zkratkou se také označuje standard popisující toto chování. Typicky je DOM strukturován tak, že kořenem stromu je objekt `Document` a z něj se větví HTML struktura, v čele s jediným povinným elementem v HTML souboru, a sice `html`. Příkladná struktura dokumentového objektového modelu je na obrázku 2.1 převzatém z [37].



Obrázek 2.1: Příklad struktury Document object model

Virtuální DOM je obecné označení pro programovací koncept, ve kterém je virtuální reprezentace DOM stromu držena v paměti a postupně synchronizována s reálným DOM [38]. Zrychlení tkví v tom, že reálný DOM není ak-

tualizován pokaždé, když by k tomu byla záminka ze strany uživatele. O optimalizovanou aktualizaci reálného DOM se stará interně React. Představit si to snadno lze na příkladu. Bez virtuálního DOM při aktualizaci seznamu na webové stránce by vždy, když se aktualizuje jeden prvek seznamu, byl překreslen celý DOM. V případě výměny celého seznamu toto ale zbytečně zatěžuje systém, obnova reálného DOM je pomalá. Místo toho se v případě (například) frameworku React aktualizuje nejdříve celý seznam ve virtuálním DOM (ten je uložen v paměti, čímž je jeho aktualizace i výrazně rychlejší) a poté až provede – pouze jednu – aktualizaci reálného DOM, který dostane nový celý seznam [39]. Ukázka 2.6 znázorňuje jednoduchou komponentu frameworku React. Jde o komponentu, která dostane v rámci `props` (tedy jako argument) jméno uživatele, kterého poté formou nadpisu první úrovně pozdraví. Vrací tedy HTML kód, do kterého jsou dynamicky dosazena data a který se vypíše v rodičovské komponentě.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Zdrojový kód 2.6: Příklad komponenty ve frameworku React

Vue je nejmladší z frameworků, vznikl roku 2014. Byl založen bývalým pracovníkem společnosti Google, dnes je ale vyvíjen komunitou, nikoliv konkrétní společností, na rozdíl od React a Angular, u kterých je to z velké části tento případ. Za poslední roky zaznamenal tento framework velký průnik do obecného povědomí a dnes vlastní nejvíce hvězdiček na Gitlab. Může za to především jeho intuitivnost a rychlost. Také (jako v případě React) nejde o kompletní MVC systém. Intuitivní je zejména díky principu komponent, které skládají dohromady webovou aplikaci. Jedním z dalších kladů je vysoká přívětivost ve věci integrace s technologiemi jako Bootstrap, což je nejoblíbenější CSS framework na světě, ulehčující vývoj responzivního uživatelského rozhraní [40]. Pojmem „responzivní“ je myšlen takový typ návrhu (a implementace) uživatelského rozhraní, které působí dobrým dojmem na všech typech zařízení tím, že se automaticky přizpůsobí podle velikosti displeje [41]. Framework Vue bude detailněji rozebrán v kapitole 2.3.

2.3 Vue.js

V této kapitole bude framework Vue rozebrán především z technického hlediska. Budou popsány jednotlivé stavební kameny a klíčové rysy, které vysvětlují, proč je tento framework tak populární.

2.3.1 Inicializace Vue aplikace

Funkce frameworku se zpřístupní po instalaci nebo zahrnutí odkazu na Vue do kořenového HTML webové aplikace. Inicializace první komponenty proběhne tak, že se vytvoří první instance Vue a předá se jí objekt s možnostmi. Této instanci se říká „kořenová“, protože z ní se větví všechny ostatní [42].

2.3.2 Vue komponenty

Vue je progresivní framework. Podle [43] to znamená, že ho lze zapojit jen do části již fungujícího, nasazeného systému. Jak již bylo naznačeno, tento framework (podobně jako ostatní typu SPA) využívá architekturu komponent. Celá aplikace je uvnitř rozčleněná do oddělených celků. Celý kód je pak strukturovaný, přehledný a znovu použitelný. Každá komponenta si drží své vlastní HTML, CSS a Javascript. Tyto části jsou odděleny speciálními značkami. HTML část je mezi otevírací a ukončovací značkou elementu `<template>`, CSS elementu `<style>` a Javascript je v elementu `<script>`. Jsou tak sice v jednom souboru označeném koncovkou .vue (koncovka označující, že jde o komponentu frameworku Vue.js), ale přehledně odděleny.

HTML část komponenty definuje obsah, respektive takzvanou šablonu. Šablonu proto, že značky HTML části drží obsah, který není neměnný a přizpůsobuje se situaci, HTML část není tedy pouze statická, ale spíše definuje strukturu, do které jsou dynamicky dosazována konkrétní data. Že jde o šablonu dokládá i fakt, že jsou HTML komponenty zasazovány do elementu `<template>`. Pro Vue je nutné, aby byl v těchto značkách pouze jeden kořenový element. Ten se pak může samozřejmě dále větvit podle potřeby. Je to kvůli vnitřnímu fungování frameworku, který z jednotlivých komponent sestavuje výsledný HTML soubor celé stránky, obsahující všechny HTML části příslušných komponent. Příklad HTML části Vue komponenty je vidět v ukázce 2.7, kde je vracen nadpis první úrovně dynamicky reagující na vlastnost `name`, která je součástí komponenty.

```
<template>
    <h1>{{name}}</h1>
</template>
```

Zdrojový kód 2.7: Příklad HTML části komponenty frameworku Vue.js

Část komponenty obsahující CSS slouží jako definice vzhledu komponenty. Struktura komponent frameworku tak zpřehledňuje i tuto část kódu aplikace. Díky této architektuře je možné i styly rozdělit do jednotlivých komponent, aby nemusel být vzhled celé stránky webové aplikace definován v jednom souboru. Zjednodušuje to orientaci i tvorbu selektorů. CSS lze v komponentě

psát jak pouze pro komponentu, tak i pro všechny její podkomponenty či globálně pro celou aplikaci. Nastavení tohoto se provede pouze jedním klíčovým slovem `scoped` jakožto atributu otevírací značky elementu `<style>`. Pokud je slůvko přítomno, definované styly se váží pouze a jenom ke komponentě, nikoliv k jejímu okolí.

Javascript část komponentě „vdechuje život“. Celá je tvořena jedním exportovaným objektem. Ten obsahuje přesně nazvané atributy se specifickým účelem. Mezi nejzákladnější patří například `data` nebo `computed`.

Atribut `data` je klíč, hodnotou je k němu funkce, která má vracet objekt obsahující definice atributů používaných v komponentě k držení dat a práci s nimi. Na tyto atributy je možné odkazovat v HTML šabloně. Jsou poté automaticky synchronizovány, změnou dat se tedy automaticky změní i referencovaná hodnota v šabloně a tedy stránka [44]. Zde je záhodno zmínit, že Vue.js také využívá technologii Virtual DOM, tedy při každé aktualizaci dat v šabloně je aktualizován nejdříve virtuální DOM, o aktualizaci toho reálného se již interně stará Vue.

Dalším atributem je `computed`. Zde jsou definovány atributy, jimž je hodnota dodána až poté, co jsou dodány všechny podklady k jejímu vypočítání. Typickým případem může být například atribut `fullName`, který vzejde spojením již definovaných dat `firstName` a `lastName` (viz ukázka 2.8) [45].

```
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0,
      firstName: "James",
      lastName: "Thompson"
    }
  },
  computed: {
    fullName: function() {
      return this.firstName + " " + this.lastName;
    }
  },
  template: '<button v-on:click="count++">
            You clicked me {{ count }} times.
            </button>'
})
```

Zdrojový kód 2.8: Příklad komponenty ve frameworku Vue.js

2.3.3 Životní cyklus komponenty

Každá komponenta ve Vue.js prochází svým životním cyklem. Ten započne fází *beforeCreate*, což je fáze před fyzickým vytvořením instance komponenty. Poslední fází je *destroyed*. Například součástí *created* fáze cyklu je kompilace šablon, nastavení sledování dat a připojení instance komponenty k DOM. Během celého tohoto procesu je umožněno programátorovi přidat svoji funkcionalitu do jednotlivých fází. Tomuto se říká *lifecycle hooks* [46]. Vhodné jsou například při záměru provést určitou logiku aplikace jen jednou, ihned po či těsně před vytvořením komponenty. Příkladem budiž okamžité jednorázové natažení dat ze serveru. Když je komponenta připravena, data jsou již stažena a připravena také, protože k jejich načtení došlo ještě před vystavením komponenty v prohlížeči.

2.3.4 Direktivy ve Vue

Direktivy jsou podle [47] speciální atributy s prefixem „v-“. Jejich hodnotou je jeden Javascript výraz (s výjimkou `v-for`). Tyto atributy přinášejí nějakou speciální funkcionalitu do renderování a zobrazování šablon komponent. Pomocí direktiv je možné předávat atributem nějaké hodnoty komponentě potomka. I tyto atributy jsou následně reaktivní, tedy pokud se změní v rodičovské komponentě, změní se i v komponentě potomka. Tohoto jednostranného provázání je možné docílit pomocí direktivy `v-bind`. Pro oboustranné provázání je používána direktiva `v-model`. Pomocí direktiv je také korigováno chování programu v případě prohlížečových událostí. K tomuto účelu slouží `v-on`. Tedy například odchycení kliknutí uživatelem na nějaký prvek je řešeno direktivou `v-on:click`. Odchytávání není limitováno na události z prohlížeče, lze také reagovat na události *emitované* komponentou potomka. Takto psanou direktivu lze následně zkrátit na `@click` (tedy `@` je plnohodnotnou náhradou za `v-on`). Stejně tak lze `v-bind` zkrátit na „:“. Například `v-bind:href="url"` lze tedy zkrátit na `:href="url"`. V neposlední řadě velmi používaná direktiva je `v-for`. Užívá se na průchod polem hodnot definovaným v `data` (nebo `computed`) či obdrženým v rámci `props`. Syntaxe je `v-for="item in items"`, kde `items` je ono zmíněné pole. Item pak představuje jeden z prvků pole. S ním pak lze v šabloně pracovat. Nejčastějším případem užití je potřeba vygenerovat stejnou komponentu pro různá data stejného typu, která jsou soustředěna v poli.

2.3.5 Rozšíření jádra frameworku Vue

V případě zájmu přidat na frontend webové aplikace více business logiky lze jádro Vue snadno integrovat s několika nástroji ze stejné dílny. Například to jsou Vuex a Vue-Router.

Podle [48] Vuex představuje vzor na řízení stavu aplikace a k němu příslušnou knihovnu, která se snadno integruje do Vue. Slouží jako centralizované

úložiště stavu pro všechny komponenty v aplikaci. Obsahuje pravidla, která se starají o omezení modifikací stavu tak, aby byly predikovatelné. Zároveň Vuex přináší rozšíření do prohlížeče vue-devtools. Toto rozšíření umožňuje sledovat aktuální stav aplikace, všech komponent a slouží k debuggingu (procesu identifikace problému, izolace jeho zdroje a jeho opravy nebo definice jeho obcházení [49]). Základním principem Vuex je tzv. single state tree. Jde o jeden jediný objekt držící všechna data o stavu aplikace na všech úrovních. Tento objekt slouží jako „single source of truth“. Díky tomuto přístupu je ulehčeno hledání informací o specifickém stavu aplikace [50].

Vuex je vhodný k použití u středních či velkých SPA aplikací. Pro malé a jednoduché aplikace jde o zbytečně složité řešení sdíleného přístupu ke stavu aplikace.

2.4 Webové aplikace v Uniqway

V této kapitole budou rozebrány webové aplikace projektu Uniqway. Uniqway spravuje celkem 3 webové aplikace, které jsou průběžně udržovány a rozvíjeny. Jde o administrátorskou aplikaci, statické zákaznické stránky a klientskou aplikaci, jejímž rozšířením se zabývá tato práce. Každá z aplikací bude krátce popsána a následovat bude podrobnější rozbor činnosti klientské aplikace.

2.4.1 Admin aplikace

Admin aplikace je pracovní název pro webovou aplikaci, která usnadňuje správu vozového parku společnosti Uniqway. Vznikla jako součást bakalářské práce Ing. Marka Šidlovského (FEL ČVUT) „Webová aplikace správy vozového parku pro systém sdílení automobilů více uživateli“. Původně byla tato aplikace napsána v Javascript frameworku Angular [51], později byla však přepsána kvůli jednotnosti do frameworku Vue.js.

Tato aplikace je centralizovaným zdrojem informací o všech uživatelích, historii jízd a automobilech. Soustřeďuje polohy vozidel, aktuální dostupnost a další údaje. Pomocí této webové aplikace lze také kontrolovat platby za službu a posílat notifikace uživatelům.

2.4.2 Statické webové stránky

Statické webové stránky slouží jako hlavní informační zdroj pro potenciální i současné uživatele. Tyto stránky jsou vyvíjeny a udržovány také ve Vue.js. Poskytují informace o týmu a jeho fungování. Také jsou zde k nalezení aktuální ceníky, zodpovězené nejčastěji kladené otázky a samozřejmě možnost registrace do služby.

2.4.3 Klientská webová aplikace

Klientská webová aplikace je původem projekt, který vznikl jako součást bakalářské práce Bc. Štěpána Severy „Webová aplikace pro uživatele systému sdílení automobilů“. Původně měla tato webová aplikace sloužit zákazníkům jako alternativa Android aplikace poskytující tutéž službu [52]. Nicméně po analýze uživatelů bylo rozhodnuto, že do produkce nasazena nebude. Závěr vychází zejména z faktu, že drtivá většina uživatelů by stejně používala pouze nativní Android aplikaci k objednávání služby a nikoliv webovou verzi. Navíc od té doby vznikla nativní aplikace pro systém iOS pro uživatele telefonů Apple iPhone. Podrobnější analýze klientské aplikace se věnuje následující kapitola.

2.5 Analýza klientské webové aplikace Uniqway

Na následujících řádcích bude provedena analýza klientské webové aplikace Uniqway. Analýza bude podrobnější proto, že tato práce se zabývá rozšířením právě této webové aplikace. Klientská webová aplikace obsahuje funkcionalitu potřebnou pro rezervaci a využívání sdílení automobilů [52]. Vzhledem k tomu, že tato aplikace nebyla na produkci nikdy využívána, neprošla od svého vzniku zásadními změnami. Tento stav nicméně poskytuje nejlepší prostor pro rozvoj a rozšíření.

Původní aplikace splňovala funkční požadavky tehdejší verze mobilní aplikace na Android, vzniklé také jako součást bakalářské práce Bc. Filipa Ravase (FEL ČVUT) „Mobilná aplikácia pre užívateľov systému zdieľania automobilov viac užívateľmi“. Při vstupu do aplikace se uživatel ocitne na domovské stránce s mapkou aktuálně volných vozidel. Tato mapa je interaktivní, jsou na ní vyznačeny pozice aktuálně volných vozidel v parkovacích zónách. Na vozidla lze kliknout a dostat se tak na detail vozu. Na levé straně obrazovky se nachází postranní panel zobrazující možnosti akcí. Umožňuje zobrazit jmenný seznam vozidel (každé vozidlo flotily má své unikátní jmenné označení) s jejich typy, anebo se lze do aplikace přihlásit. Pro zobrazení seznamu vozidel ani dalších akcí uživatel nemusí být přihlášen.

Výběrem možnosti Vozidla se uživatel dostane na seznam aktuálně volných vozidel, který je zobrazen formou dlaždic ve dvourozměrné tabulce. Z těch již vedou odkazy na detaily jednotlivých vozů. Seznam aut je vidět na obrázku 2.2.

Obrázek 2.2: Seznam aut klientské webové aplikace

Po výběru vozidla se zobrazí jeho detail. V případě nepřihlášeného uživatele se zobrazí výchozí forma obrazovky. V této formě jsou vidět technické parametry vozidla či informace o stavu jeho nádrže. Mezi technické údaje se řadí například objem a výkon motoru, typ převodovky nebo objem zavazadlového prostoru a počet míst k sezení. Zobrazí se také možnost přihlášení do aplikace kvůli potenciální rezervaci. Kromě těchto informací je k dispozici náhled vozidla a současně i mapka lokality, kde je vozidlo zaparkováno.

Po kliknutí na možnost Přihlášení má uživatel zadat údaje, pod kterými se registroval do systému na webu Uniqway nebo pod kterými se přihlásil do mobilní aplikace. Po jejich vyplnění je přenesen na Hlavní stránku. Z té je možné se opět pomocí mapky či postranního panelu proklikat k detailu vozidla, kde se místo možnosti Přihlášení zobrazí možnost Rezervace. Po kliknutí na Rezervaci se aplikace zeptá uživatele na typ tarifu a také na potvrzení, zda opravdu chce automobil rezervovat. Pokud uživatel vše potvrdí, je mu na jeho platební kartě vytvořena blokace 200 Kč. Po ukončení rezervace je přeplatek podle ceny služby vrácen uživateli na bankovní účet. [52]

Po rezervaci se v Detailu vozu zobrazí možnosti odemknutí vozidla a vrácení vozidla z rezervace. Zároveň je zobrazen PIN k CCS kartě, která se u vozů Uniqway používá k čerpání paliva. Odemknutí vozidla se provádí přiložením karty k čidlu vozidla ve lhůtě několika sekund od zvolení možnosti v aplikaci. Pokud se vůz ohlásí a odemknutí se připojí, zobrazí se možnost zamknutí. V opačném případě aplikace nabídne možnost opakování pokusu o odemknutí.

Pro uzamčení vozu stačí pouze kliknout na možnost zamknutí a auto se po obdržení požadavku ze serveru zamkne samo, bez nutnosti přiložení karty. Podmínkou pro vrácení automobilu je jeho uzamčení a zaparkování v jedné z vyhrazených parkovacích zón. Pakliže jsou tyto podmínky splněny, uživatel je aplikací vyzván k potvrzení ukončení rezervace a po úspěšném potvrzení jsou mu zobrazeny parametry jeho rezervace – délka trvání, počet ujetých kilometrů a celková výsledná cena rezervace.

2.5.1 Výhody a nevýhody klientské webové aplikace

Klientská webová aplikace trpí zejména faktem, že není produkčně udržována. Proto také její funkcionalita zaostává za nativními aplikacemi pro Android a iOS. Aplikace není propojena se systémem obchodu s odměnami. Neobsahuje kontaktní informace, nápovědu pro krizové situace ani místa, kam se může uživatel dojít zaregistrovat osobně. Další nevýhodou je absence profesionálně navrženého designu grafického uživatelského rozhraní. Tato nevýhoda je ale vzhledem k situaci pochopitelná.

Ve vztahu k této práci má aktuální stav klientské webové aplikace nevýhodu v tom, že nikterak nepodporuje možnost správy kontrol automobilu ani všeobecného stavu vozidel. O některých aspektech stavu vozidel informuje, avšak jejich správa umožněna není. Uživatelská interakce je limitována na základní potřeby užívání automobilů.

Na druhou stranu má klientská webová aplikace i mnoho výhod. Paradoxní výhodou je například fakt, že nefunguje produkčně. Jde tím pádem o projekt malého až středního rozsahu, ve kterém se programátor snadno a rychle zorientuje. Další výhodou je skutečnost, že je aplikace napsána ve Vue.js, shoduje se tedy s ostatními webovými aplikacemi Uniqway. Aplikace je napsána přehledně a jednoduše. Jednou z dalších výhod je například fakt, že původní aplikace již implementuje technologie potřebné pro realizaci projektu rozšíření o funkcionalitu pro správce vozidel. Tuto technologii tedy při implementaci stačí rozšířit o potřebnou další funkcionalitu, není třeba ji do projektu integrovat nově. Jednoduchý intuitivní design a vícejazyčnost, to jsou jen další nefunkční požadavky, které aplikace splňuje.

Klientská webová aplikace také naplno využívá výhod SPA (a obecně webových) aplikací oproti nativním Android a iOS aplikacím. V čele těchto výhod je jednoznačně multiplatformnost a snadnější aktualizace. Multiplatformnost vychází z povahy aplikace, která běží v prohlížeči a ten je již vyvinut na všechny operační systémy mobilních telefonů, lze tedy tutéž aplikaci spustit na všech zařízeních bez nutnosti implementovat speciální verzi. To výrazně ulehčuje vývoj a posun aplikace. Proces aktualizace je usnadněný o činnost ze strany uživatele. Po nahrání nové verze na server je již automaticky dostupná všem uživatelům v prohlížeči. U mobilních verzí je nutné si samostatně u každého uživatele stáhnout aktualizaci a nainstalovat si ji.

Kapitola 3

Návrh rozšíření o podporu kontroly vozidel

V této kapitole bude popsán průběh návrhu rozšíření klientské webové aplikace Uniqway o podporu kontroly vozidel. V první řadě bude definována kontrola jako proces, co obnáší a jak probíhá. Následuje popis funkčních a nefunkčních požadavky na rozšíření, spolu s nimi pak jednotlivé případy užití výsledku této práce. V další části bude popsán návrh uživatelského rozhraní. Tato kapitola bude zakončena návrhem komunikace aplikace se serverem.

3.1 Struktura kontroly

Standardní kontrola probíhá v několika fázích, při kterých se postupně hledí na exteriér, interiér a technický stav automobilů. Souslednost kroků má ve svém režimu každý ambassador (správce vozidel) individuálně, avšak celková sada úkonů kontroly je stejná.

Největší částí vozu ke kontrole je exteriér. Ten je nutné zkontrolovat ze všech stran. Důraz je kladen i na drobné škrábance kdekoliv na karoserii, dále jsou zkontrolována skla, světla a pneumatiky. Při nálezu defektu je nejdříve zkontrolován jeho záznam na některé z minulých kontrol a pokud jde o nový defekt, je zaznamenáno datum nálezu, slovní popis poškození a jsou přiloženy fotografie.

Následuje vnitřek vozu. Zde je nutné zkontrolovat obě řady sedadel a obě strany, řidiče i spolujezdce. Opět evidenci neuniknou ani drobné oděrky a škrábance. Kontroluje se funkčnost vnitřního osvětlení, portů k zapojení mobilního telefonu, stav řadicí páky a ruční brzdy. Následuje kontrola veškerého nutného vybavení vozidla. Mezi to patří například lékárnička, náhradní žárovky, reflexní vesty, apod.

Po kontrole vybavení nastává kontrola technického stavu vozidla. V něm jde především o funkčnost světel a motoru a kontrolu hladiny provozních ka-

palin. Vůz je při této části kontroly nastartován a zaznamenána je i aktivita varovných kontrolek. Posledním krokem je potvrzení celkové způsobilosti vozu k dalšímu používání.

3.2 Strom lokalizace defektu

Při nálezu poškození vozu je potřeba rychle, ale zároveň přesně definovat jeho umístění. Se zvyšující se přesností určení místa roste počet možností, a tedy prostý seznam takovýchto míst není vhodným řešením. Ideálu bližší je stromovitá struktura, ve které se s postupným zanořováním do hladin dál od kořene objevují specifičtější lokace. Při správném rozdělení částí vozu tento způsob představuje nejlepší poměr přesnost lokalizace/rychlost lokalizace. Takovýto strom byl v rámci této práce vyvinut. Uživatel při značení místa defektu začíná rozhodnutím, zda je poškození v interiéru či exteriéru. Na nejnižší hladinu stromu lze dojít po maximálně pěti krocích. Jeho kompletní podoba je součástí přílohy E této práce.

3.3 Požadavky na rozšíření

Jednou z cest, jak definovat rozsah a hloubku návrhu a později i implementace řešení je specifikace požadavků, funkčních a nefunkčních, které má řešení splňovat.

Požadavky musejí plnit určitá kritéria, aby byly validní a v konečném důsledku hlavně užitečné a směrodatné. Společnou zkratkou pro tato kritéria je FURPS, představující pojmy *functionality*, *usability*, *reliability*, *performance* a *supportability*. Pod *functionality* si lze představit schopnost systému, kompatibilitu a přenositelnost. Pojem *usability* vyjadřuje použitelnost ve smyslu uživatelské zkušenosti (UX – *user experience* – tedy pole zabývající se aspekty interakce uživatele a produktu [53]). *Reliability* označuje spolehlivost, dostupnost a s ní související četnost poruch. Pod rouškou *performance* se skrývá hlavně zkoumání efektivity, náročnosti na hardware a škálovatelnosti. Termín *supportability* značí podporovatelnost a rozšiřitelnost, rychlost oprav a obecnou flexibilitu řešení. [54]

3.3.1 Funkční požadavky

Dle [54] jsou funkční požadavky definovány jako požadavky určující funkce aplikace – tedy co má aplikace umět. Tyto požadavky musí být měřitelné a testovatelné, na konci vývoje musí být jednoznačně určitelné, které funkční požadavky byly splněny a které nikoliv.

Kompletní výpis funkčních požadavků na rozšíření klientské webové aplikace je následující:

F1 Označení poškozené části automobilu
Při zaznamenávání defektu na automobilu je nutné co nejpřesněji a nejsnadněji zaznamenat lokaci defektu, ať již jde o interiér či exteriér. Podrobnosti lze najít v podkapitole 3.2 - Strom lokalizace defektu.

F2 Možnost popisu defektu
Popis poškození vozu bude systémem umožněn jak v písemné, tak obrazové podobě. K defektu tedy bude možné vložit slovní popis situace a přiložit i fotografický materiál.

F3 Seznam aktuálně platných defektů vozidla
Součástí aplikace bude možnost nahlédnout na celkový aktuální seznam platných defektů vozidla. K nim budou k dispozici detaily nesoucí informace o lokaci, slovní popis a přiložené fotografie poškození.

F4 Označení chybějícího vybavení ve voze
Uživatel bude mít možnost zaznamenat i chybějící vybavení vozu. Mezi to patří například kontrola přítomnosti lékárničky, rezervních žárovek a podobně.

F5 Záznam technického stavu vozidla
Systém bude umožňovat zaznamenat vybrané aspekty technického stavu vozidla, mezi něž patří například svícení varovných kontrolek či stav kapalin vozidla.

F6 Záznam okolností kontroly vozidla
V záznamu o kontrole vozidla musí být kromě informací o vozidle samotném také určitá metadata, zaznamenávající okolnosti kontroly. Mezi tato data patří například ambassador, který kontrolu prováděl, datum a čas samotné kontroly a označení kontrolovaného vozu.

F7 Náhled výsledků předchozích kontrol
V aplikaci bude k dispozici uživateli kompletní seznam kontrol daného vozidla spolu s možností náhledu na její výsledky a záznam.

F8 Editace poslední kontroly
Systém bude poskytovat možnost editovat poslední kontrolu. Při odeslání chyby je tedy možné kontrolu opravit. Zároveň tato vlastnost plní funkci přerušení procesu kontroly.

F9 Záznam výměny pneumatik
Stejně jako záznam kontroly vozidla je možné zaznamenat výměnu pneumatik. V rámci toho se zaznamenávají údaje typu hloubky vzorku zouvaných pneumatik i značka či typ výměny (letní za zimní či naopak). Zároveň bude k dispozici seznam výměn pneumatik daného vozidla. Bude tedy možné nahlédnout na výsledky předchozích výměn.

3.3.2 Nefunkční požadavky

Nefunkční požadavky určují kvalitu systému, zkoumají jeho bezpečnost a dostupnost. Do té patří i míra odolnosti vůči výpadkům. Plnění nefunkčních požadavků nelze exaktně změřit, posouzení je pouze subjektivní záležitost [54]. Mezi nefunkční požadavky rozšíření klientské webové aplikace patří:

N1 Responzivita
Systém bude mít navržené a implementované uživatelské rozhraní responzivní. Uživatelská přívětivost by tedy měla být zajištěna pro všechny druhy zařízení, grafické uživatelské rozhraní se automaticky přizpůsobí velikosti zařízení. Zvláštní důraz musí být kladen na design pro mobilní telefony, správci vozidel provádějí kontroly v terénu a do systému se bude přistupovat většinově právě přes telefon.

N2 Vícejazyčnost
Kompletní rozhraní rozšíření systému bude poskytnuto ve dvou jazykových verzích. K dispozici bude verze v anglickém jazyce a českém jazyce.

N3 Multiplatformnost
Multiplatformnost je nefunkčním požadavkem původního systému, stejně jako jeho rozšíření. Systém bude dostupný ve většině operačních systémů (Linux, Windows, MacOS, Android, iOS,...) a na různých druzích zařízení od mobilních telefonů přes tablety až po PC.

N4 Vypracování ve Vue.js
Celá původní klientská aplikace je napsána v javascriptovém frameworku Vue.js, v tomto frameworku bude vypracováno i její rozšíření o podporu kontrol vozidel.

N5 Omezení přístupu
Rozšíření webové aplikace bude dostupné pouze ověřeným uživatelům, kteří mají v systému přidělena příslušná uživatelská práva, a to po ověření totožnosti přihlášením. Nepřihlášeným uživatelům není poskytnuta žádná funkcionalita rozšíření.

3.4 Případy užití

V rámci návrhu rozšíření aplikace je součástí definice rozsahu a náročnosti práce také návrh případů užití (UC – *use case*) výsledného rozšíření. Případy užití následně zjednodušují orientaci v požadavcích při vývoji. Seznam případů užití pro rozšíření o podporu správy vozidel je následující:

UC01 – Evidovat viditelné poškození vozu
Rozšíření umožňuje ambassadorovi zaznamenat nový defekt na voze. Záznam o defektu sestává z lokace defektu na automobilu, slovního popisu a případně přiložené fotografie. Před samotným zaznamenáním je možné nahlédnout do aktuálně platných defektů daného vozu, aby nebyly do aplikace zadávány defekty duplicitně. Označení místa poškození probíhá v několika úrovních počínaje rozdělením exteriér / interiér, které se řídí stromem lokalizace defektu. Tento případ užití využívá funkční požadavky F1, F2 a F3.

UC02 – Zkontrolovat přítomnost veškerého vybavení vozu
Systém umožňuje uživateli průběžně při kontrole označovat věci, které ve voze mají být, proti přiloženému seznamu pro daný vůz. Jde například o chybějící karty k vozu, lékárničku, náhradní žárovky apod. Opět je k dispozici seznam

již chybějících předmětů, vycházející ze záznamu předchozí kontroly vozu. Funkční požadavek F4 je tímto use casem využit.

UC03 – Evidovat stav a funkčnost vozu po technické stránce
Uživatel zaznamenává aktuální stav motorového oleje, chladící kapaliny a kapaliny do ostřikovačů. Kontroluje funkčnost světel vozidla a případné rozsvícení některé z varovných kontrolek vozu po nastartování. Také v této části ambassador zadá do aplikace, zda proběhlo čištění exteriéru či interiéru vozidla. Nakonec finálně potvrdí, zda je vozidlo způsobilé pro další jízdu. Tento případ využije funkční požadavek F5.

UC04 – Nahlédnout na výsledky předchozích kontrol
Rozšíření umožňuje zobrazení výsledků historických kontrol včetně metadat typu ambassadora, který kontrolu prováděl. Tímto jsou využity funkční požadavky F6 a F7. Historická data o kontrolách umožňují analyzovat nejčastější poškození vozu.

UC05 – Upravit předchozí kontrolu
Ambassador je schopen v aplikaci upravit poslední kontrolu, přidat či odebrat poškození nebo poupravit jakékoliv jiné evidované údaje. Tato funkce zároveň reprezentuje možnost přerušení kontroly a jejího dokončení později. Díky tomuto případu užití je využit požadavek F8.

UC06 – Zaznamenat výměnu pneumatik
Aplikace umožňuje uživateli vytvořit záznam o speciálním typu kontroly – výměně pneumatik. Mezi zaznamenávané údaje patří hloubka dezénu zouvaných pneumatik nebo například jejich značka. Pro kontrolu může uživatel po záznamu nahlédnout na seznam historických výměn pneumatik. Využit je tedy i funkční požadavek F9.

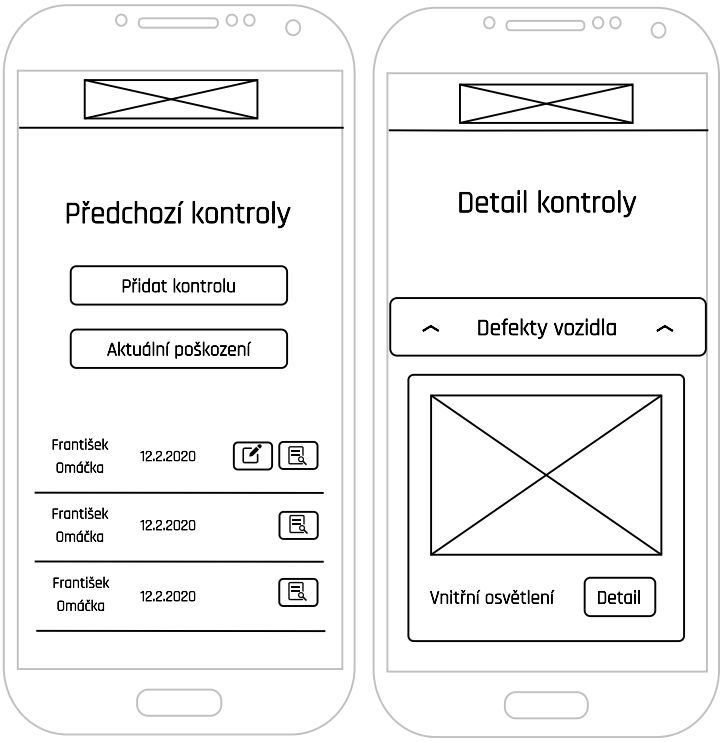
3.5 Návrh uživatelského rozhraní

Nedílnou součástí vývoje webových aplikací je návrh uživatelského rozhraní. U klientské webové aplikace (respektive jejího rozšíření) tomu není jinak. Grafické uživatelské rozhraní je vždy nejprve navrženo, poté až programováno, aby se chyby rozhraní projevily a opravily ještě před samotnou implementací. V rámci této práce byly navrženy dvě verze všech obrazovek. Počítačová verze ukazuje vzhled aplikace v případě užívání z PC, tedy na relativně velkém monitoru. Druhá verzí je mobilní, tedy rozhraní přizpůsobené pro procházení z prohlížeče na mobilním telefonu.

3.5.1 Seznam kontrol vozidla

První obrazovkou po zvolení možnosti „Kontrola vozidla“ z postranního menu klientské webové aplikace je výběr vozidel. K dispozici jsou všechna vozidla a datum jejich poslední kontroly. Návrh byl převzat z původní aplikace, kde uživatel obdobným stylem vybírá vůz k rezervaci, a následně drobně upraven, aby obsahoval informaci o poslední kontrole.

Po zvolení vozidla se uživateli už zobrazí seznam dosavadních kontrol daného vozu. V rámci tohoto seznamu je u každé položky vidět jméno správce, který kontrolu provedl, a datum kontroly. Následuje odkaz na detail výsledku kontroly se všemi údaji. Wireframe (dvoudimenzionální obrys kostry aplikace [55]) s návrhem je vidět na obrázku 3.1.



Obrázek 3.1: Seznam kontrol vozidla, detail kontroly vozidla – část „Defekty vozidla“ (mobilní verze)

3.5.2 Detail kontroly vozidla

Další obrazovkou je detail konkrétní kontroly obsahující všechny informace zaznamenané ambassadorem při kontrole. Tato obrazovka je rozdělena na 3 sekce podle pomyslného členění kontroly - záznamy defektů vozidla, kontrola vybavení a kontrola technického stavu vozidla. V počítačové verzi návrhu jsou tyto sekce umístěny vedle sebe. Je tak využit potenciál široké obrazovky ke zrychlení editace kontroly, možnost „přeskakování“ mezi sekcemi je rychlejší než v mobilní verzi. V té jsou kvůli šířce obrazovky telefonu sekce umístěny pod sebe. Aby ale uživatel mohl rovnou editovat například poslední část, zobrazí se v této verzi ke každé části tlačítko sloužící k jejímu zakrytí. Uživatel pak

může jednoduše dvěma kliky zakrýt první dvě části kontroly a rychle se tak přesunout k editaci části třetí.

První část – defekty vozidla – ukáže defekty zaznamenané v rámci dané kontroly. Každý ze záznamu obsahuje fotografii defektu (v případě, že při vytváření záznamu defektu bylo přiloženo více fotografií, ukáže se první z nich), krátký popisek a odkaz na detail. Krátkým popiskem je myšlena poslední zvolená hladina ve stromu lokalizace defektu, tedy nejpřesnější zvolené umístění. Tento popisek spolu s fotkou slouží pouze pro rychlé zorientování, detail defektu s plným popisem i kompletním určením místa se dá zobrazit po kliku na tlačítko „Detail“. Obrázek 3.1 ukazuje mobilní verzi této části.

Druhá část obsahuje seznam vybavení k danému vozu. Rozdělené je na samotné vybavení (například žárovky, lékárnička,...), zvláštní kategorii pak mají čipy a dokumenty (například technický průkaz či tankovací CCS karta). Kompletní seznam vybavení lze vidět v příloze D.

U každé položky je pak ikona indikující jeho přítomnost či absenci, kromě této ikony se zde nachází také ikona poznámky. V případě potřeby má při kontrole ambassador možnost ke kterémukoliv vybavení dopsat poznámku. Tato funkce je zde proto, že například při chybějícím obvazu v lékárničce není vhodné za chybějící označit lékárničku celou, v tomto případě se pouze k záznamu přiloží poznámka. Wireframe této části je vidět na obrázku 3.2.

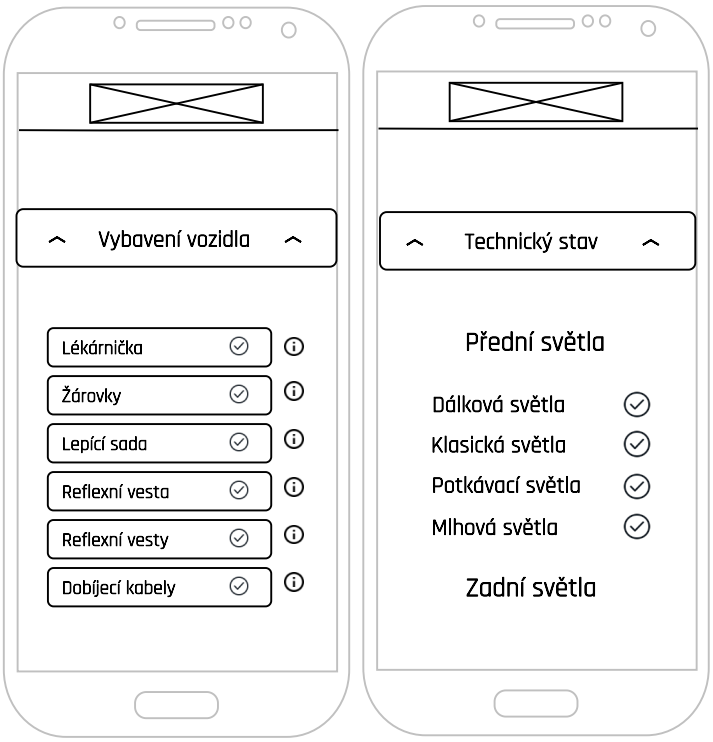
Ve třetí části se kontroluje technický stav vozidla. Do něj patří funkčnost světel, stav provozních kapalin a do sekce „Ostatní“ patří například stav tachometru, provedení čištění či ostatní defekty vozu. Nakonec je označeno, zda je vozidlo způsobilé pro další provoz. Celý seznam faktorů kontrolovaných v této části lze nalézt v příloze C. Na obrázcích 3.2 a 3.3 jsou vidět ukázky návrhu.

3.5.3 Tvorba a úprava záznamu o kontrole

Aplikace umožňuje tvorbu nových kontrol a úpravu historických. Pro přístup k této možnosti bude sloužit tlačítko „Přidat kontrolu“ v seznamu všech kontrol pro dané vozidlo (po výběru vozidla). Úprava poslední kontroly je možná z tlačítka „Upravit“, umístěného u příslušné kontroly. Po zvolení jedné z těchto možností se uživatel dostane na modifikovanou obrazovku detailu kontroly. Kostra obrazovky je stejná, rozdíl je pouze v několika detailech.

Při tvorbě či úpravě kontroly je v první části kromě možnosti otevřít si detail defektu také možnost záznam defektu upravit. Další změnou je tlačítko umožňující přidat nový defekt. Při tvorbě kontroly jsou v této sekci zobrazené všechny aktuální defekty automobilu – na rozdíl od klasického detailu kontroly, ve kterém se zobrazí pouze defekty zaznamenané při dané kontrole.

Střední část je modifikována tak, aby obrázky indikující přítomnost či absenci nějakého vybavení byly „klikací“. Po kliku na toto tlačítko (zde je již samotný termín „ikona“ nedostatečně přesný) se změní na tlačítko opačné. Tedy po stisku tlačítka, které indikovalo správný stav daného kusu vybavení



Obrázek 3.2: Detail kontroly vozidla – části „Vybavení vozidla“ a „Technický stav“ (mobilní verze)

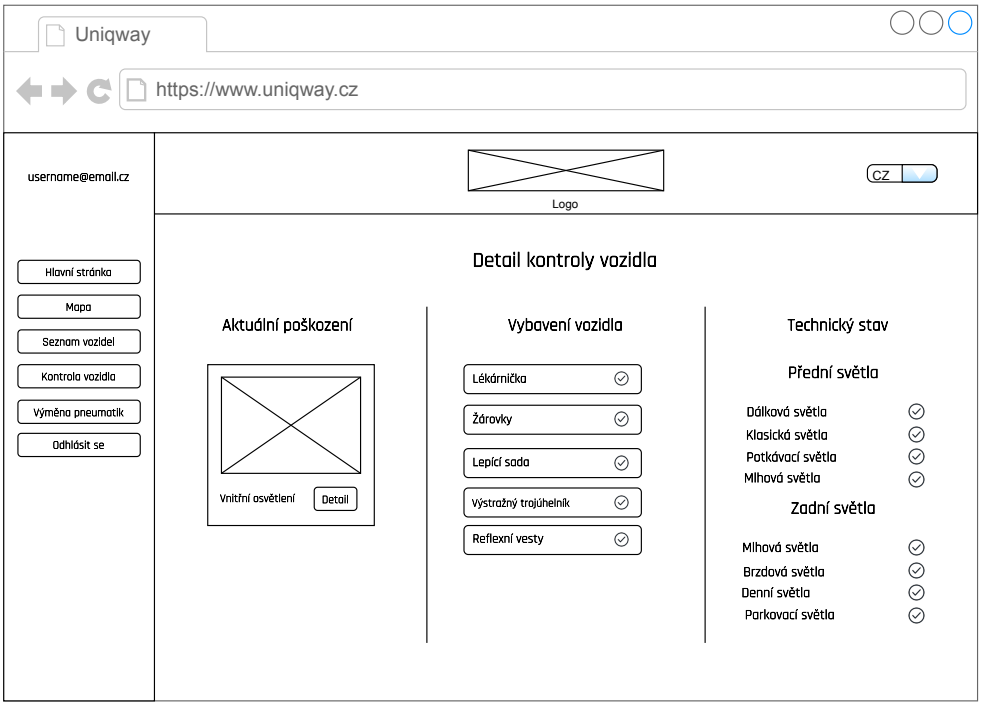
se z něj stane tlačítko poukazující na chybu. Pří editaci kontroly (pro tvorbu platí také) je možnost přidat poznámku k jakémukoliv vybavení. Tato možnost je realizována tlačítkem, po kterém se zobrazí tzv. *popup* s textovým polem pro vložení poznámky.

Modifikace poslední části jsou obdobného rázu. Z pouhých ikon se stala tlačítka s možností přepnout stav (například světel, která již nefungují). Ze statických informací o doplnění kapalin jsou pole s možnostmi, uživatel pouze volí, zda doplňoval kapalinu nebo stav stačil. Indikace celkové způsobilosti vozu k dalšímu provozu je ve stejné režii.

Pod všemi třemi sekcemi se u úprav objeví možnost odeslat záznam stiskem tlačítka „Upravit kontrolu“ (v případě úpravy) nebo „Ukončit kontrolu“ (v případě tvorby nové kontroly).

3.5.4 Detail defektu

Wireframe obrazovky obsahující detaily defektu je vidět na obrázku 3.4. Každý defekt obsahuje fotografie, přesnou lokaci a slovní popis. Kromě tohoto se ukládá například také datum zadání do systému.

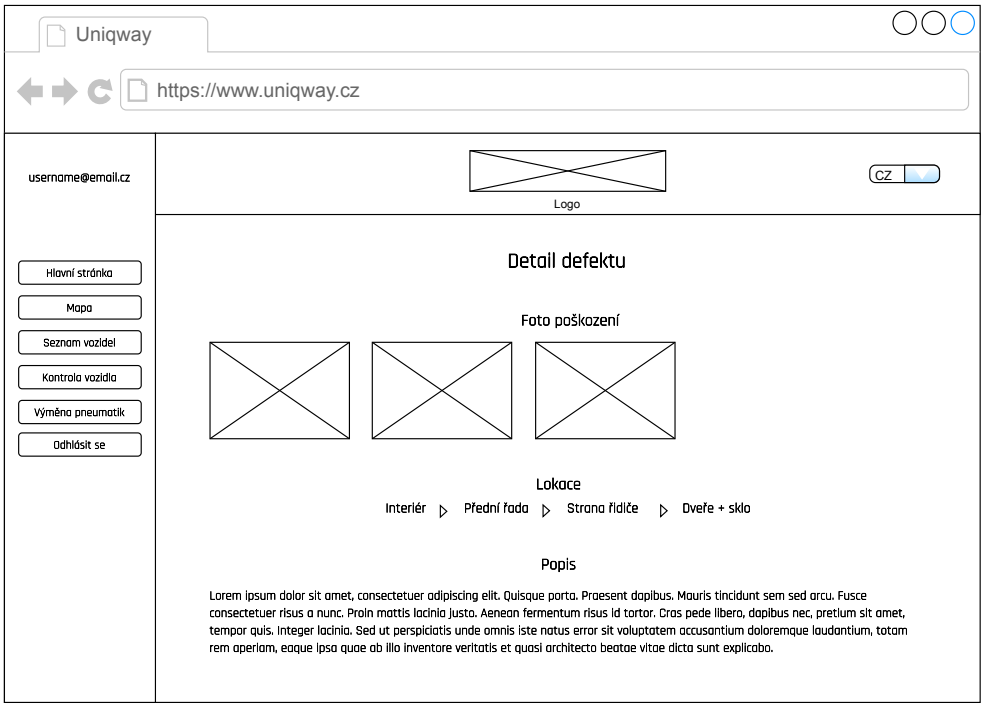


Obrázek 3.3: Detail kontroly vozidla (PC verze)

3.5.5 Tvorba a úprava záznamu o defektu

Tvorba nového záznamu o defektu zahrnuje lokalizaci poškození - vybráním vhodné cesty ve stromu lokalizace defektu. Cesta je vybírána tak, že nejdříve je uživateli nabídnuta první úroveň stromu, která obnáší položky „Interiér“ a „Exteriér“. Kliknutím na jednu z nich se pod ní otevře další úroveň podle toho, kterou uživatel zvolil, zároveň se zvolená možnost viditelně označí. Jednotlivé úrovně jsou skládány pod sebe, pro rychlejší přesun mezi jednotlivými částmi vytváření defektu je nahoře nad prvním výběrem lokace opět tlačítko, které na kliknutí skryje či odkryje kompletní cestu ve stromu. Takto ji lze vždy zkontrolovat, případně upravit, zároveň však uživateli nijak „nepřekáží“. Následně je uživateli poskytnuta možnost přidat do záznamu fotografie poškození, kliknutím na tlačítko „Přidat foto“ se otevře prohlížeč souborů na telefonu či PC. Následuje zadání slovního popisu a odeslání defektu.

Úprava záznamu o defektu je rozdílná od tvorby nového pouze ve skutečnosti, že jsou informace předvyplněné podle upravovaného záznamu. Na obrázku 3.5 jsou vyobrazeny návrhy lokalizace defektu a ostatních částí tvorby záznamu.



Obrázek 3.4: Detail poškození vozidla (PC verze)

3.5.6 Seznam aktuálních poškození vozu

Kompletní seznam aktuálních poškození vozu je možné dostat dvěma způsoby. Prvním z nich je vytvoření nové kontroly. V rámci něj ambassador reviduje aktuální poškození a jejich platnost, tam jsou tedy zobrazena všechna. Druhým způsobem je kliknutím na tlačítko „Aktuální poškození“ na obrazovce seznamu kontrol daného vozidla. Tímto tlačítkem se uživatel však dostane na samostatnou stránku zobrazující informace. Návrh této stránky je vidět na obrázku 3.6. Ke každému poškození je zde vidět jedna položka obsahující krátký popis (poslední zvolená úroveň ze stromu lokalizace defektu) a datum založení záznamu. Po kliknutí na položku seznamu vyjede detail ke konkrétnímu záznamu. Tento detail je stejného rázu jako po kliku na detail defektu z obrazovky „Detail kontroly vozidla“, jsou zde tedy zobrazené všechny informace včetně fotografií. Po opětovném kliknutí na „hlavičku“ záznamu se detail skryje. Takto lze záznamy pohodlně procházet i na mobilním zařízení.

3.5.7 Výměna pneumatik

V aplikaci je možné evidovat i výměnu pneumatik, jakožto speciální typ kontroly. Při této výměně se evidují různé informace o původních i nově nasazovaných pneumatikách. Patří mezi ně typ výměny – zda jsou zouvány letní

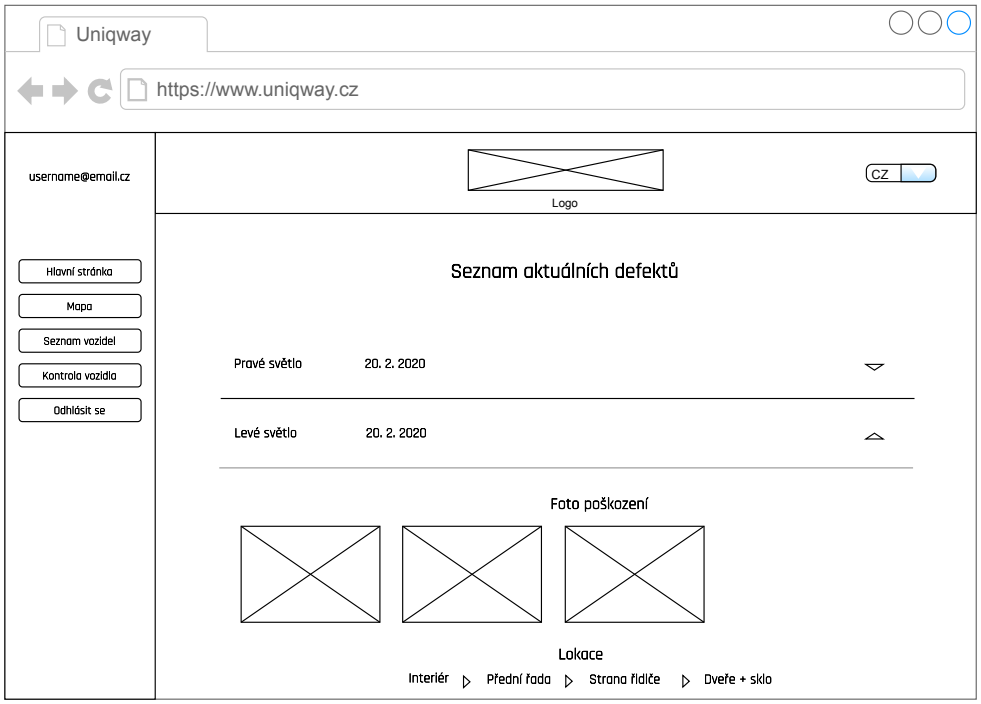
Obrázek 3.5: Tvorba defektu – lokalizace a ostatní informace (mobilní verze)

či zimní pneumatiky (a nasazovány opačné). U zouvaných pneumatik se eviduje hloubka dezénu. Doplňkovými informacemi jsou název výrobce zouvaných/nazoufaných pneumatik a typy disků (zda jde o ocelové či hliníkové).

O celý tento proces se v aplikaci starají dvě, resp. tři obrazovky. První z nich je výběr automobilu, na kterém je výměna prováděna. Ten je identický s výběrem auta pro kontrolu či jeho detail (funkcionalita původní klientské webové aplikace). Druhou obrazovkou je seznam výměn pneumatik daného vozu, ten se nese ve stejném duchu jako seznam všech defektů automobilu. Jde tedy o výčet všech výměn pneu, každá položka obsahuje jméno ambassadora, který se o výměnu postaral, a datum výměny. Po kliknutí na položku se zobrazí detail výměny. Třetí obrazovka slouží k přidávání nové výměny pneu do evidence. Má podobu formuláře, jehož odeslání způsobí, že nový záznam následně přibude do seznamu všech kontrol. Na obrázku 3.7 lze vidět wireframe obrazovky sloužící pro přidání záznamu do evidence výměny pneumatik.

3.6 Návrh komunikace se serverem

V této části bude popsán návrh API pro komunikaci serveru Uniqway s rozšířením klientské webové aplikace. Všechna posílaná data jsou ve formátu JSON,



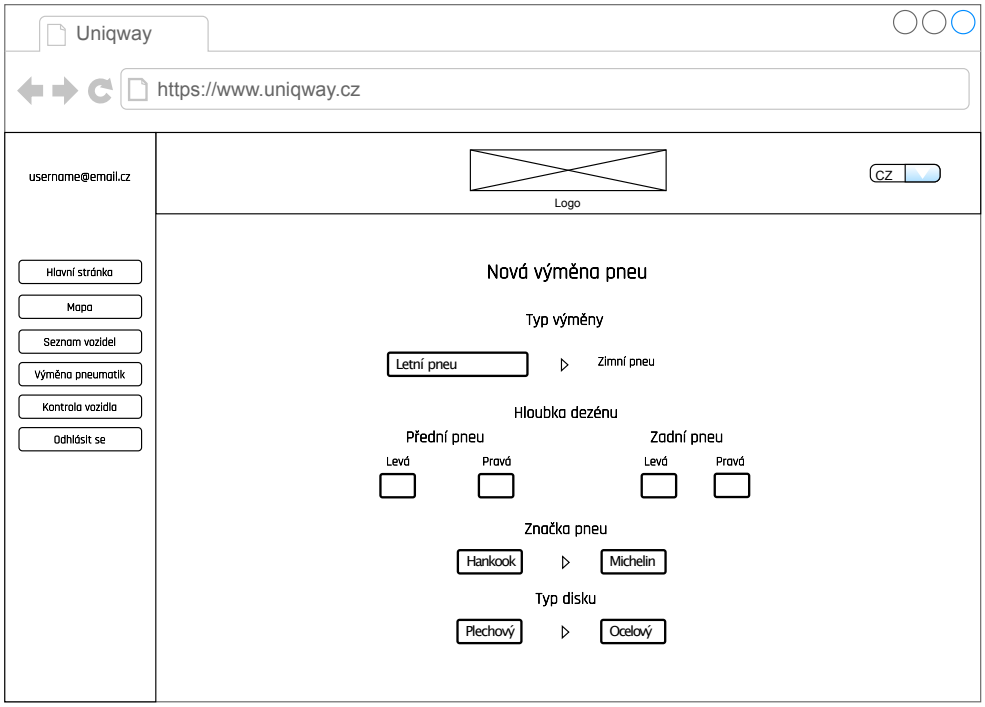
Obrázek 3.6: Seznam aktuálních defektů vozu (PC verze)

v této kapitole se vyskytují ukázky komunikace, většina je pro účely příkladu ve zkrácené verzi. Protože serverová strana zatím není implementovaná (a její implementace není předmětem této práce), funkcionalita serveru je simulována pomocí falešného API. K tomuto účelu byl využit nástroj Mockoon. Tento program zvládne vytvořit lokální server na nastavitelném portu, který poté reaguje na HTTP požadavky. Vybrán byl zejména kvůli své jednoduchosti. Mezi ostatní možnosti se řadí například technologie JSONPlaceholder (konkrétně My JSON Server) od společnosti Typicode. Její nevýhodou je ale velikostní omezení těl odpovědí na HTTP požadavky a také skutečnost, že jde o beta verzi s občasnými výpadky. Po nainstalování programu Mockoon se uživatel rychle zorientuje díky předepsaným příkladům nejpoužívanějších metod. Mockoon funguje na běžných počítačových operačních systémech, tedy na Windows, MacOS i Linux [56].

3.6.1 Kontroly vozů

Následuje seznam metod, které poskytují funkcionalitu komunikace aplikace se serverem ohledně dat z kontrol vozů.

- **Všechny kontroly daného vozidla**
  Tato metoda zajišťuje žádost serveru o seznam všech kontrol ke specifi-



Obrázek 3.7: Tvorba záznamu o výměně pneumatik (PC verze)

kovanému vozidlu. Vozidlo je označeno svým unikátním identifikátorem poslaným v URL adrese metody. V odpovědi metody se vracejí data ve formátu JSON. Jde o pole objektů, kde každý jednotlivý objekt představuje odkaz na jednu kontrolu. Obsahují identifikátor kontroly, ID ambassadora, který kontrolu vykonal, a datum kontroly.
Metoda: `GET`
URL: `admin/car-check/car/{{carId}}`
Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
carId	ano	číslo

Tabulka 3.1: Parametry HTTP GET požadavku: všechny kontroly daného auta kontroly

Chybová odpověď: 400 Bad Request (Car not found)
Tělo odpovědi:

```
[
    {
      "checkId": 12,
```

```
      "ambassadorId": 89,
      "createdAt": "2020-12-30 14:46:00"
    }
]
```

- **Detail specifikované kontroly**
  Tato metoda se stará o požadavek na server o detailní informace ke specifikované kontrole. Kontrolu identifikuje její ID uvedené v URL. V odpovědi dostane zpět všechny evidované informace k identifikátoru kontroly, které byly při vytváření kontroly odeslány na server.
  Metoda: `GET`
  URL: `admin/car-check/{{checkId}}`
  Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
checkId	ano	číslo

Tabulka 3.2: Parametry HTTP GET požadavku: detail specifikované kontroly

Chybová odpověď: 400 Bad Request (Check not found)
Tělo odpovědi:

```
{
  "id": 22,
  "type": "normal",
  "carId": 1,
  "isFinished": true,
  "createdAt": "2020-12-30 14:46:00",
  "ambassadorId": 89,
  "defects": [
    1,
    2
  ],
  "technicalStatus": {
    "liquids": {
      "coolant": "aboveMin",
    }
  }
}
```

- **Vytvoření nové kontroly**
  Vytvořením nového záznamu o kontrole se zabývá tento požadavek.

V těle požadavku je specifikovaný identifikátor automobilu, kterému kontrola náleží. Odesílají se také informace nasbírané od uživatele - jejich struktura odpovídá struktuře odpovědi na požadavek o detail defektu (kromě atributu ID). Při správné odpovědi se poté vrací ze serveru ID přidělené databází záznamu dané kontroly.
Metoda: `POST`
URL: `admin/car-check`
Očekávaná odpověď: 200 OK, 201 Created

- **Úprava některé z předešlých kontrol**
  Pro upravení některé z předešlých kontrol je potřeba v těle požadavku odeslat upravené informace v původním formátu (struktura těla požadavku je tedy shodná s tělem požadavku o vytvoření kontroly). Kromě tohoto je v URL metody specifikován přímo zdroj, který je upravován – v tomto případě identifikátor kontroly, která je upravována.
  Metoda: `PUT`
  URL: `admin/car-check/{{checkId}}`

Parametr	Povinný	Datový typ
checkId	ano	číslo

Tabulka 3.3: Parametry HTTP PUT požadavku: úprava specifikované kontroly

Očekávaná odpověď: 200 OK
Chybová odpověď: 400 Bad Request (Check not found)

3.6.2 Defekty vozů

Pro práci s defekty jsou vyhrazené speciální metody. Starají se o přidávání defektů, jejich mazání a úpravu, lokalizaci defektů. Jejich soupis je následující:

- **Všechny aktuální defekty daného vozidla**
  Tato metoda se zabývá žádostí o soupis všech neopravených defektů k danému vozidlu. V URL metody je specifikované ID vozidla. Defekty přicházejí ve formě pole JSON objektů se základními informacemi o defektech. Mezi ty patří krátký popis - poslední zvolená úroveň stromu lokalizace defektu, která je v aplikaci překládána ze serializované podoby na český ekvivalent. Dále ID defektu a fotografie (první z fotografií přidaných při tvorbě záznamu o defektu). Ukázka těla odpovědi je zjednodušena na pouze jeden defekt.
  Metoda: `GET`
  URL: `admin/car-defects/car/{{carId}}/active`
  Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
carId	ano	číslo

Tabulka 3.4: Parametry HTTP GET požadavku: seznam všech aktivních defektů vozidla

Chybová odpověď: 400 Bad Request (Car not found)
Tělo odpovědi:

```
[
    {
      "defectId": 1,
      "shortDescription": "rightLight",
      "photo": "https://defect-photos.com/defect01.jpeg"
    }
]
```

- **Detail konkrétního defektu**
  O nahrání detailních informací o defektu se stará tato metoda. V URL dostane identifikátor defektu a jako odpověď přijde objekt se všemi informacemi evidovanými o defektech včetně kompletní cesty ve stromu lokalizace defektu.
  Metoda: `GET`
  URL: `admin/car-defects/{{defectId}}`
  Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
defectId	ano	číslo

Tabulka 3.5: Parametry HTTP GET požadavku: detail konkrétního defektu vozidla

Chybová odpověď: 400 Bad Request (Defect not found)
Tělo odpovědi:

```
{
  "id": 1,
  "carId": 1,
  "createdAt": "2020-01-01 14:58:34",
  "repairedAt": null,
  "location": [
    "exterior"
  ],
```

```
  "photos": [
     "https://defect-photos.com/defect01.jpeg"
  ],
  "description": "Pravé světlo proražené"
}
```

- **Přidávání nového defektu**
  Tato metoda se stará o přidání nově vytvořeného záznamu o defektu. Tělo požadavku má obdobnou strukturu jako tělo odpovědi na požadavek o detail defektu, samozřejmě vyjma atributu ID. Důležitý je atribut `carId`, ten určuje, ke kterému automobilu nový defekt patří. Identifikátor samotného záznamu defektu ve chvíli odesílání na server ještě není znám, automaticky ho vygeneruje a přidělí databáze a tento identifikátor následně přijde v odpovědi na tento požadavek.
  Metoda: `POST`
  URL: `admin/car-defects/`
  Očekávaná odpověď: 200 OK, 201 Created

- **Úprava defektu**
  Aktuální defekty lze upravovat podle potřeby, stará se o to tato metoda. Struktura posílaného těla požadavku je stejná, jako při vytváření defektu, zdroj se záznamem defektu je totiž kompletně vyměněn za jeho aktualizovanou verzi.
  Metoda: `PUT`
  URL: `admin/car-defects/{{defectId}}`
  Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
defectId	ano	číslo

Tabulka 3.6: Parametry HTTP PUT požadavku: editace konkrétního defektu vozidla

Chybová odpověď: 400 Bad Request (Defect not found)

- **Stáhnutí stromu lokalizace defektu**
  Poslední metodou starající se o hladký průběh evidence defektů vozidla je metoda požadavku o kompletní podobu stromu lokalizace defektu. Tento strom přichází ve formátu JSON. Struktura příchozího objektu se větví podle úrovní stromu. Parametr `name` značí uzel stromu, parametr `data` pak jeho potomky ve stromu. Zde je tělo odpovědi na požadavek zkráceno pouze na první a druhou úroveň stromu. Kompletní podoba stromu je v příloze E.
  Metoda: `GET`

URL: `admin/defect-location-tree`
Očekávaná odpověď: 200 OK
Tělo odpovědi:

```
{
    "name": "defectLocationTree",
    "data": [
        {
            "name": "interior",
            "data": [
                {
                    "name": "frontRow",
                    "data": [
                    ]
                },
            ]
        },
    ]
}
```

3.6.3 Vybavení vozů

Následující metody zajišťují obsluhu zdrojů týkajících se vybavení vozu. Obsluha zahrnuje dvě základní činnosti – přidávání nového stavu vybavení specifikovaného vozu (respektive aktualizace aktuálního stavu vybavení) a získání tohoto stavu.

- **Aktuální stav vybavení vozidla**
  Tato metoda zajišťuje požadavek o aktuální stav vybavení specifikovaného vozidla. Aktuální stav je vždy měněn při kontrole, po této žádosti tedy přijde výsledek poslední kontroly, konkrétně její sekce o vybavení vozu. V těle odpovědi se nachází data ve formátu JSON odpovídající záznamům podle návrhu obrazovky detailu kontroly. Každý jednotlivý kus vybavení u sebe má informaci o tom, zda je v pořádku, případně poznámku. Tělo odpovědi je zde zkráceno, kompletní výpis lze vidět v kapitole 3.5.2. Metoda: `GET`
  URL: `admin/car-equipment/{{carId}}`
  Očekávaná odpověď: 200 OK
  Chybová odpověď: 400 Bad Request (Car not found)
  Tělo odpovědi:

```
{
    "carId": 1,
```

Parametr	Povinný	Datový typ
carId	ano	číslo

Tabulka 3.7: Parametry HTTP GET požadavku: aktuální stav vybavení vozidla

```
    "equipment": [
        {
            "id": 1,
            "name": "medicalKit",
            "isOkay": true,
            "note": ""
        },
    ],
    "chips": [
        {
            "id": 1,
            "name": "PRE",
            "isOkay": false,
            "note": "Nefunkční"
        },
    ],
    "cardsAndDocuments": [
        {
            "id": 1,
            "name": "CCS",
            "isOkay": true,
            "note": ""
        },
    ]
}
```

- **Aktualizace stavu vybavení vozidla**
  Tato metoda se stará o poslání aktuálního stavu vybavení vozidla na server, a tak zajišťuje jeho perzistenci. Struktura požadavku přesně kopíruje strukturu těla odpovědi požadavku o aktuální stav vybavení vozidla. Rozdílem jsou samozřejmě uživatelem změněné položky či přidané poznámky. Položka `carId` jednoznačně určuje vozidlo, kterého se změna týká.
  Metoda: `POST`
  URL: `admin/car-equipment`
  Očekávaná odpověď: 200 OK

3.6.4 Výměny pneumatik

Tyto metody zajišťují funkcionalitu výměny dat aplikace se serverem. Data se týkají zouvaných a nazouvaných pneumatik jakožto speciálního typu kontroly.

- **Všechny výměny pneumatik**
  Tato metoda se stará o nahrání všech výměn pneumatik ze serveru. Výměny se vztahují k jednomu konkrétnímu autu, jehož identifikátor je poskytnut v URL. Tělo odpovědi obsahuje všechny informace zadávané při tvorbě nového záznamu výměny pneumatik. Kromě těchto informací je také v odpovědi přijat čas výměn a identifikátor ambassadorů, kteří výměnu prováděli. Zde je tělo zkráceno na jednu kontrolu. Metoda: `POST`
  URL: `admin/tire-exchanges/{{carId}}`
  Očekávaná odpověď: 200 OK

Parametr	Povinný	Datový typ
carId	ano	číslo

Tabulka 3.8: Parametry HTTP GET požadavku: seznam všech záznamů výměn pneumatik daného vozidla

Chybová odpověď: 400 Bad Request (Car not found)
Tělo odpovědi:

```
[
    {
      "carId": 1,
      "ambassadorId": 89,
      "createdAt": "2020-12-30 14:46:00",
      "type": "tireExchange",
      "newTires": "summerTires",
      "oldTirePatternDepth": {
        "frontTires": {
          "rightTire": 4.7,
          "leftTire": 4.9
        }
      },
      "tireBrand": {
        "oldTireBrand": "Hankook",
        "newTireBrand": "Michelin"
      },
      "wheelType": {
        "oldWheelType": "steel",
        "newWheelType": "steel"
      }
```

```
    }
]
```

- **Nová výměna pneumatik**
  Tato HTTP metoda posílá na server ve svém těle informace o nové výměně pneumatik. Její tělo je strukturálně shodné se záznamem o výměně získaném v odpovědi na předchozí metodu. Automobil je opět jednoznačně určen atributem `carId`.
  Metoda: `POST`
  URL: `admin/tire-exchanges`
  Očekávaná odpověď: 200 OK

Kapitola 4

Implementace rozšíření

V této kapitole bude popsána struktura implementace návrhů a konceptů z předchozích kapitol. Budou představeny komponenty, jejich funkce a vzájemná komunikace, následovat bude implementace komunikace se serverem. Rozšíření je programováno v jazyce Javascript, frameworku Vue.js, ve kterém je napsána i celá aplikace. Doplňky frameworku, například Vue-Router, Vuex či knihovna pro testování Jest jsou již v projektu integrované, rozšíření je pouze využívá, výčet a krátký popis takto využitých rozšíření frameworku bude následovat po představení implementace komunikace se serverem.

4.1 Komponenty a struktura rozšíření

Komponenty jsou (jak již bylo zmíněno) základním stavebním kamenem jakékoliv Vue.js aplikace. Na následujících řádcích bude popsána struktura rozšíření po stránce komponent. Každá komponenta má své kompetence a funkci. Vzhled komponent je dán návrhy (wireframe) obrazovek z kapitoly 3. Pro přehlednost jsou všechny kontroly rozděleny do tří kategorií podle funkcionality, kterou zajišťují.

4.1.1 Kontrola vozidla

- **CarCheckDetail**
  Rozložení prvků této komponenty je dáno návrhem obrazovky detailu kontroly (obrázek 3.3). Obsahuje 3 části, první část – část evidovaných defektů – obsahuje seznam komponent typu „CarDefectListItem“, které reprezentují jednotlivé aktuální defekty vozu. Druhá část zaobaluje seznam prvků typu „CarEquipmentListItem“, což jsou komponenty, z nichž každá představuje jeden kus vybavení a s ním spjaté informace. Třetí část obsahuje komponentu „CarTechnicalStatus“, zobrazující technický stav vozidla. Tato komponenta také slouží jako hlavní zdroj informací o kontrole. Je zodpovědná za načtení dat ze serveru a předání do

odpovědných komponent. Ty si samy stahují ze serveru pouze pomocná data, hlavní informace jsou jim předány při vytvoření.

Všechny části jsou schopné pracovat ve dvou základních režimech. Jedním je režim „prohlížení“, sloužící pouze k zobrazení detailu kontroly. Pokud ale jde o detail poslední kontroly a uživatel zvolil možnost úpravy kontroly, pak se tato komponenta nachází v režimu „editace“, ve kterém po ukončení všech úprav řídí odeslání výsledků na server.

Funkční požadavek F6 – Záznam okolností kontroly vozidla, zejména tedy záznam informací o správci, který kontrolu vykonal, data a času kontroly apod. je díky této komponentě splněn, všechny tyto informace jsou komponentou shromažďovány a následně odesílány k persistenci v databázi. Komponenta detailu záznamu kontroly se také spolupodílí na splnění funkčního požadavku F7 – Náhled výsledků předchozích kontrol. Pomáhá i ke splnění funkčního požadavku F8 – Editace poslední kontroly. O faktickou editaci se sice starají zmíněné podkomponenty, ale tato komponenta je zodpovědná za sběr upravených informací a následné odeslání na server.

- **CarCheckList**
  Tato komponenta je zodpovědná za vyobrazení seznamu kontrol daného vozidla. Ke každé kontrole je zobrazeno jméno správce, který kontrolu provedl, a datum kontroly. Následuje u každého prvku odkaz na detail. U poslední zaznamenané kontroly je nabídnuta i možnost kontrolu upravit. V rámci této komponenty jsou i vedena tlačítka poskytující možnost vytvořit novou kontrolu či zobrazit seznam aktuálně platných defektů vozidla. Vzhled komponenty odpovídá návrhu z obrázku 3.1. Tato komponenta spolu s „CarCheckDetail“ splňují kompletně funkční požadavky F7 a F8 (nabízí možnost editace poslední kontroly, která je provedena v komponentě detailu kontroly vozidla).

- **CarCheckListItem**
  Seznam aut ke kontrole a k výměně pneumatik obsahuje položky představující jednotlivá auta. Tato komponenta se stará o zobrazení informací k položce seznamu. Obsahuje informace o jménu auta a jeho typu, v případě seznamu aut ke kontrole také datum poslední kontroly. Její vzhled vychází z původní aplikace, kde je obdobně navržený prvek seznamu aut k rezervaci.

- **CarEquipmentListItem**
  Tato komponenta představuje jeden prvek seznamu vybavení vozidla. Kromě označení vybavení také nese informaci o jeho stavu a případnou písemnou poznámku. Stará se tedy (spolu s rodičovskou komponentou, která informace shromažďuje) o splnění čtvrtého funkčního požadavku – Označení chybějícího vybavení ve voze. Pokud je rodičovská komponenta

„CarCheckDetail“ v režimu editace, je v režimu editace i každý z prvků seznamu vybavení vozu. V tu chvíli lze klikem na tlačítko indikující stav vybavení tento stav změnit. Komponenta následně informaci o změně odesílá rodičovské komponentě.

- **CarTechnicalStatus**
  Všechny evidované informace o technickém stavu vozidla jsou spravovány pomocí této komponenty. Vzhled vychází z návrhu v obrázku 3.3. V editačním režimu rodičovské komponenty lze i prvky této komponenty stisknutím příslušného tlačítka či vyplněním textového pole měnit. Tato komponenta se stará o to, aby byl splněn funkční požadavek F5 – Záznam technického stavu vozidla.

4.1.2 Defekt vozidla

Celkem šest komponent se podílí na evidenci defektů vozidla a společně splňují všechny příslušné funkční požadavky, tedy požadavky F1, F2 a F3.

- **CarDefectCreation**
  Komponenta zaštiťující tvorbu nového záznamu o defektu obsahuje několik částí. Především je to část, ve které probíhá definice umístění poškození. Tu tvoří seznam komponent „CarDefectLocation“, každá z nich představuje jednu hladinu ve stromu lokalizace defektu. Tato sekce se dá tlačítkem skrýt, pod ní se nachází možnost přidat k záznamu fotografie a vlastní komentář. Částečně díky této komponentě je splněn funkční požadavek F1 – Označení poškozené části. Komponenty typu „CarDefectLocation“ totiž reagují na volbu uživatele a vracejí této komponentě zvolenou možnost, ta poté určí další hladinu stromu lokalizace defektu a postará se o její zobrazení vytvořením další komponenty téhož typu. Komponenta pak kompletně splňuje funkční požadavek F2 – Možnost popisu defektu. Po uzavření tvorby záznamu o defektu lze informace tlačítkem „Hotovo“ odeslat. Vzhled komponenty vychází z návrhu v kapitole 3.5.5.

  Tato komponenta se kromě tvorby záznamů o defektech stará i o editaci. Úprava záznamu defektu je obdobná jeho tvorbě, rozdílem je skutečnost, že při zvolení možnosti editace komponenta požádá server o podrobnosti o upravovaném záznamu. Uživatel tedy při příchodu zobrazení komponenty vidí již předvyplněné informace.

- **CarDefectDetail**
  Tato komponenta má na starost zobrazení všech informací o defektu vozidla. Samotné zobrazení informací probíhá pomocí znovu použité komponenty „CarDefectExpandableListItem“, tato komponenta slouží pouze jako obal a dodává jí potřebná data, o která požádá server.

- **CarDefectExpandableListItem**
  Jak již název napovídá, tato komponenta vznikla za účelem informací v seznamu defektů. Slovo „expandable“ nese proto, že je seznam defektů koncipován tak, že po kliku na prvek seznamu se pod prvkem zobrazí příslušný detail defektu, prvek se tedy „roztáhne“. Tato funkcionalita je přístupná pouze při přístupu přes seznam aktuálních defektů vozu (komponenta „CarDefectList“). Komponenta je ovšem využívána i při rozkliku detailu defektu v kontrole vozidla. V tomto případě ale má k dispozici celou stránku a tak seznamové „roztahování“ není potřeba.

  Tato komponenta se stará o vykreslení detailních informací o defektu. Informace jí přijdou z rodičovské komponenty, sama o žádné server nežádá. Vzhled stránky zobrazující informace je dán návrhem na obrázku 3.4.

- **CarDefectList**
  Tato komponenta se stará pouze a jenom o zobrazení seznamu komponent typu „CarDefectExpandableListItem“. Slouží tedy jako obal pro tento seznam, poskytuje podkomponentám informace, o které žádá server. Díky této komponentě je splněn funkční požadavek F3 – Seznam aktuálně platných defektů vozidla.

- **CarDefectListItem**
  Funkci položky v seznamu defektů plní i tato komponenta. Rozdíl oproti komponentě „CarDefectExpandableListItem“ je ten, že zde jde o prvek seznamu defektů v detailu kontroly vozidla (tedy jeho první části). Tyto prvky obsahují jednu fotografii defektu, krátký popis a odkaz na detail, případně úpravu záznamu o defektu (pokud komponenta detailu kontroly je v režimu „editace“). Všechny potřebné informace jsou komponentě dodány při její tvorbě, sama komponenta se nestará o žádné požadavky na server.

- **CarDefectLocation**
  Komponenta „CarDefectLocation“ má na starosti několik činností. Sama zobrazuje seznam položek jedné hladiny stromu lokalizace defektu. Například pro první hladinu zobrazuje položky „Interiér“ a „Exteriér“. Po zvolení jedné z nich uživatelem je do rodičovské komponenty volba zaslána a v kompletním zobrazení při tvorbě defektu přibyde další instance této komponenty, s další hladinou stromu podle odeslané volby. Ukázka návrhu je vidět na obrázku 3.5. Díky této a rodičovské komponentě je splněn funkční požadavek F1 – Označení poškozené části automobilu.

4.1.3 Výměna pneumatik

Následuje popis komponent starajících se o záznamy výměn pneumatik. Díky nim je pokryto splnění funkčního požadavku F9 – Záznam výměny pneumatik,

jehož součástí je jak seznam historických výměn pneu, tak samotný záznam tohoto typu kontroly.

- **CarTireExchangeCreation**
  O vykreslení formuláře, kam ambassador vyplní při výměně pneumatik všechny potřebné informace, se stará tato komponenta. Dostane k dispozici pouze identifikátor vozu, strukturu kontroly si drží ve vlastní proměnné. Po vyplnění se postará o odeslání dat na server a přesměrování zpět na stránku se všemi kontrolami daného vozu. Její vzhled je odvíjen od návrhu z obrázku 3.7.
- **CarTireExchangeExpandableListItem**
  Ve stejném stylu jako komponenta „CarDefectExpandableListItem“ je i tato komponenta. Tvoří ji tedy jeden prvek seznamu, který uživatel „roztáhne“ tím, že na prvek seznamu klikne. Zobrazují se poté informace zadané při tvorbě kontroly typu výměna pneumatik. Zobrazované informace jsou jí předány rodičovskou komponentou, ona sama se serverem nijak nekomunikuje.
- **CarTireExchangeList**
  Tato komponenta je pouze obal pro seznam komponent typu „CarTireExchangeExpandableListItem“. Kromě tohoto seznamu obsahuje tlačítko pro vytvoření nového záznamu o výměně pneumatik. Stará se také o zaslání požadavku o záznamy kontrol se všemi informacemi, tyto záznamy poté posílá do komponent seznamu, kde jsou jednotlivě zobrazovány.

4.2 Implementace komunikace se serverem

Celá komunikace se serverem probíhá v komponentě „restApi“. Ta používá technologii *axios*, což je HTTP klient (nejen) pro prohlížeč, založený na slibech (*promises*) [57]. Slib je typ javascriptového objektu, který reprezentuje úspěch či neúspěch asynchronní operace - v tomto případě HTTP (REST) požadavku, například o změnu dat na serveru [58].

Komponenta „restApi“ obsahuje definice metod pro odesílání požadavků na server. Tato komponenta již byla založena v původní verzi klientské webové aplikace, v rámci rozšíření však byla doplněna o metody pro práci s informacemi o kontrolách, defektech či výměnách pneumatik.

Příkladem takové metody je ukázka 4.1. Ta implementuje návrh metody podle 3.6.1. Na příslušné URL pošle *request* (požadavek) o detail kontroly. Pokud nenastane problém, provede se funkce předaná jako argument funkci `then` - ta poté do atributu `check` předá data, která dostala ze serveru (detail kontroly ve formátu JSON). Pakliže nastane problém, zobrazí se uživateli v prohlížeči dialogové okno s chybovou hláškou. Toto se provede pomocí rozšíření Vuejs Dialog, viz kapitola 4.4.

```
loadCarCheck(checkId, th) {
    /*
    let realUrl = hostname[hostname.current].ROOT_API
                + "/admin/car-check/"
                + checkId;
    */
    let fakeUrl = "http://localhost:3000/admin/car-check/"
                + checkId

    th.$http.get(fakeUrl)
        .then(response => {
            th.check = response.data;
        },
            error => {
                Dialoger.showErrorDialog(th, error);
            })
}
```

Zdrojový kód 4.1: Ukázka metody pro práci s REST API serveru Uniqway – získání detailu kontroly

V ukázce lze vidět použití parametru `fakeUrl`. Tato URL vede na lokální server poskytovaný na portu 3000. Server je poskytován technologií Mockoon (viz kapitola 3.6) a 3000 je její výchozí číslo portu. Pro účely testování funkčnosti aplikace je na této URL nastavena odpověď – zejména tělo odpovědi s informacemi – stejného formátu, jako tomu bude u produkční verze backendu aplikace. Parametr `realUrl` pak značí předpokládanou reálnou URL serveru. Tato funkcionalita ale prozatím není na straně serveru Uniqway implementována.

4.3 Ošetření uživatelských vstupů

V rámci bezpečnosti aplikace je vždy první zásadou ošetření uživatelských vstupů, tedy akce nějakým způsobem omezující (například textový) vstup. V klientské webové aplikaci jsou takto omezeny všechny vstupy, kam může uživatel volně psát text. Tuto funkcionalitu zprostředkovává rozšíření Vuelidate.

Vuelidate umožňuje definovat ve struktuře objektu typu JSON validační pravidla, která musí kontrolované části aplikace splňovat. Objekt s definicemi pravidel je ve Vue aplikaci nazván `validations`. Splnění těchto pravidel je poté automaticky či manuálně kontrolováno. Kontrolovaná jsou většinou data,

která do aplikace zadal uživatel (lze ale takto validovat i jakákoliv jiná data aplikace). V případě projektu rozšíření webové aplikace Uniqway jsou data kontrolována například na maximální délku nebo povinnost zadání. Příklad je vidět v ukázce 4.2.

```
validations: {
    technicalStatus: {
      odometer: {
        required,
        maxLength: maxLength(7)
      }
    }
}
```

Zdrojový kód 4.2: Ukázka validace dat ve Vuelidate

4.4 Využívaná rozšíření frameworku Vue.js

V této sekci budou představena rozšíření využitá při implementaci správy automobilů v klientské webové aplikaci Uniqway. Všechna tato rozšíření byla při implementaci využita, avšak pouze jedno z nich v aplikaci přibylo přímo kvůli správě automobilů, zbytek byl již integrován v původní verzi. Zmiňované technologie usnadňují mnoho procesů v aplikaci, například přesměrovávání na komponenty podle URL, překládání textového obsahu aplikace do různých jazyků nebo validaci vstupu od uživatele.

4.4.1 Vue Router

Vue Router je oficiálním směrovačem pro framework Vue. Je integrován s jádrem frameworku a slouží k mapování komponent na URL adresy [59]. Aby se předešlo nedefinovanému chování, vždy bývá definována také „obecná“ cesta, značená pouze symbolem „*“. Hvězdička odpovídá regulárnímu výrazu vyjadřujícímu „jakkoliv dlouhý řetězec symbolů“. Pokud tedy Vue Router nenalezne shodu s předchozími cestami, s touto cestou shodu najde vždy.

Vue Router se ale nestará pouze o samotné mapování komponent, řídí také přístup k nim, tedy autorizaci, pomocí definice seznamu povolených uživatelských rolí. V projektu klientské webové aplikace Uniqway je definice cest v souboru „routes.js“. Příklad mapování komponent na URL včetně některých z výše zmíněných funkcionalit je možné vidět v ukázce 4.3.

```
export const routes = [
    {
        path: '/settings',
        component: Settings,
        meta: {
            requiresAuth: true,
            allowedRoles: ['admin', 'user']
        }
    },
    {
        path: "*",
        component: PageNotFound
    },

];
```

Zdrojový kód 4.3: Ukázka mapování komponent na URL – Vue Router

4.4.2 Vuex

Vuex je knihovna pro Vue.js aplikace umožňující jednoduchou správu stavu aplikace. Funguje centralizovaně pro celou aplikaci (v celé aplikaci je instance stavu pouze jedna) a zajišťuje, aby byly veškeré změny stavu předvídatelné [60]. Pracuje také s „Vue devtools“, což je rozšíření do prohlížeče usnadňující debugging Vue.js aplikace. V projektu rozšíření klientské webové aplikace je Vuex, respektive jeho centralizované úložiště využíváno například pro zjištění údajů o aktuálně přihlášeném uživateli.

4.4.3 Vuejs Dialog

Vuejs Dialog je plugin do frameworku Vue.js usnadňující tvorbu dialogových oken. Dialogová okna jsou jednou z cest, jak informovat uživatele o nějaké akci či změně stavu. Například v původní verze aplikace se po zadání neplatných přihlašovacích údajů zobrazí dialogové okno, které uživatele na nedostatek upozorní. Rozšíření klientské aplikace využívá plugin zejména ke zpětné vazbě po provedení asynchronních metod – volání na server.

4.4.4 Vue I18n

Další z již integrovaných pluginů v aplikaci je Vue I18n. Tento plugin slouží k překládání obsahu do různých jazyků. V aplikaci je vícejazyčnost řešena pomocí několika souborů – pro každý jazyk jeden soubor. V nich jsou v da-

ném jazyce uloženy všechny překládané termíny, vždy formou klíče a hodnoty. Klíčem je identifikátor pojmu, tento identifikátor je jednotný přes všechny jazykové soubory (za předpokladu, že je do daných jazyků překládán). Hodnotou je překlad pojmu do daného jazyka.

4.4.5 Vuelidate

Vuelidate je využíváno k ošetření vstupů od uživatele. Integrováno bylo již v původní verzi aplikace. Podrobněji k Vuelidate i kontrole uživatelských vstupů v sekci 4.3.

4.4.6 Jest

Jest je jednoduché rozšíření frameworku Vue.js využívané k validaci dat na klientské straně webové aplikace. Pracuje jednoduše a rychle. Všechny testy si udržují unikátní globální stav, díky tomu je možné testování paralelizovat (tedy spustit více testů ve stejný čas). Další výhodou je, že Jest v pořadí spouštění testů preferuje testy, které v minulosti selhaly. Pořadí je také ovlivněno dobou běhu jednotlivých testů. Jest umožňuje generovat pokrytí kódu testy, vhodné pro zhodnocení kvality otestování aplikace. V neposlední řadě tento framework podporuje *mockování* – simulaci – objektů, které jsou mimo dosah testů a jsou pro test potřeba. [61] Více o průběhu testování v kapitole 5.

4.4.7 Vue Expandable Image

Tento plugin do Vue.js jako jediný nebyl integrován již v původní verzi projektu. Jde o rozšíření frameworku, které implementuje možnost zvětšení obrázku po jeho zvolení uživatelem. Tato funkce je využita v detailu záznamu o defektu vozidla, kde jsou všechny pořízené fotografie. Tyto fotografie si uživatel může jedním kliknutím zvětšit na velikost displeje zařízení.

Kapitola 5

Testování

Na následujících řádcích bude rozebráno téma testování aplikací obecně, následovat bude popis testů rozšíření klientské webové aplikace. K jednotkovému testování aplikace byl využit javascriptový framework Jest, který je podrobněji popsán v kapitole 4.4.6. V závěru budou představeny testovací scénáře pro fyzického uživatele.

5.1 Testování obecně

Nedílnou součástí vývoje jakékoliv aplikace je její testování. Jeho cílem je změření a zajištění kvality programu a následná úspora zdrojů [62]. Jednotlivé druhy testů lze dělit podle několika kategorií. Jednou z nich jsou například druhy testů podle rozsahu. Mezi ty patří testy jednotkové, testy komponent, dále integrační a systémové testy.

Jednotkové testy se zabývají testováním jednotlivých tříd a konkrétních funkcí, testy tohoto druhu jsou rozsahově nejmenší. Testování komponent je prováděno stejným stylem jako jednotkové testování, rozdíl je v tom, že jsou testovány celé komponenty – skupiny tříd (termín „komponenta“ zde není ve významu frontendové komponenty). Tímto druhem testů je kontrolována větší funkcionalita programu. Integrační testy se zabývají kontrolou validity komunikace mezi jednotlivými komponentami. Funkčnost aplikace jako celku je kontrolována pomocí systémových testů. U tohoto druhu nemá tester o vnitřním fungování aplikace žádné znalosti a kontroluje pouze výstupy na dané vstupy. [62]

5.2 Testování rozšíření

Rozšíření webové aplikace je testováno dvěma způsoby. Jedním z nich jsou jednotkové testy popsané výše. Takto otestovány jsou jednotlivé komponenty na přítomnost všech částí a jejich správné chování v případě změny stavu

aplikace. Kontroluje se také například správné přepínání jazyka, pokud to uživatel zvolí. Následuje testování systémové, v podobě scénářů, které bude plnit fyzický tester, čímž zkontroluje jak funkčnost aplikace, tak celkovou použitelnost.

5.2.1 Jednotkové testy

Jednotkové testy, nebo také *unit* testy, jsou v projektu umístěny ve složce `/tests/unit`. Jsou rozčleněny do souborů podle komponenty, kterou mají za úkol prověřit. V ukázce 5.1 je možné vidět ukázku testu komponenty detailu kontroly, kontrolující přítomnost některých částí a jejich počtu. Další testy kontrolují například správný informační obsah nebo korektní přijetí předaných `props` komponentou.

```
it('car defects present', () => {
    expect(wrapper.find('.defects')
                    .exists()).toBe(true)
    expect(wrapper.find('.car-defects-wrapper')
                    .exists()).toBe(true)
    expect(wrapper.find('.defect-snippet')
                    .exists()).toBe(true)
    expect(wrapper.findAll(CarDefectListItem)
                    .length).toBe(3);
    expect(wrapper.find('.add-defect-button-wrapper')
                    .exists()).toBe(true)
});
```

Zdrojový kód 5.1: Ukázka jednotkových testů komponenty CarCheckDetail

5.2.2 Scénáře pro fyzického testera

Součástí systémového testování aplikace jsou i scénáře pro fyzického testera, které mají za úkol obecnou funkčnost a použitelnost aplikace v reálném světě. Uživatelské scénáře bohužel nemohly být (vzhledem k absenci serverové části rozšíření) provedeny v plném rozsahu. Soupis testovacích scénářů je následující:

- **Seznámení se stavem automobilu**

  1. Zvolení automobilu ke kontrole
     Prvnímu kroku předchází přihlášení do webové aplikace a zvolení možnosti „Kontrola vozidla“ z postranního menu. Následně si uživatel vybere ze seznamu vůz, s jehož stavem se chce seznámit. Po

kliknutí na příslušnou položku seznamu je očekávaným výsledkem přesměrování na seznam kontrol daného vozidla.

2. Zvolení detailu konkrétní kontroly
   Ze seznamu kontrol si uživatel nejdříve vybere konkrétní kontrolu a zvolí možnost detailu kontroly kliknutím na příslušné tlačítko. Tato akce má přenést uživatele na stránku s detailními informacemi o kontrole.
3. Nahlédnutí na soupis aktivních poškození vozidla
   Po seznámení s výsledkem se uživatel vrátí zpět na seznam kontrol a zvolí možnost „Aktuální poškození“. Očekávaný výsledek je přesměrování na seznam aktuálních poškození, z nichž si uživatel jedno vybere, „rozklikne“, a tím se mu ukáže detail daného poškození.

Celkovým výsledkem tohoto scénáře je obeznámení s okolnostmi, za kterých bude uživatel – ambassador – tvořit nový záznam o kontrole.

- **Tvorba záznamu o novém defektu vozidla**

  1. Výběr lokace defektu
     Před zahájením tohoto scénáře je uživatel povinen se přihlásit do aplikace. Následně zvolí možnost „Kontrola vozidel“, která ho přenese na seznam vozů, ze kterých si vybere konkrétní vůz. Dále uživatel zvolí možnost přidání nové kontroly nebo úpravy poslední z kontrol vozu. Obě tyto možnosti ho mají přenést na stránku detailu kontroly, ve které je možnost data upravovat. Poté uživatel zvolí možnost „Přidat defekt“ v první sekci detailu kontroly. Očekávaný výsledek této akce je přesměrování na stránku tvorby defektu.

     Tvorba záznamu začíná výběrem lokace. Uživatel tedy nejdříve „rozklikne“ volbu „Lokace“ a následně zvolí z možností „Interiér“, nebo „Exteriér“. Očekávaným výsledkem je zobrazení další hladiny stromu lokalizace předmětu podle zvolené možnosti (další hladina je zobrazena ve stejném formátu, jako předchozí). Tuto akci bude uživatel opakovat, dokud nebude lokace specifikována dostatečně nebo dokud nedojde uživatel na konec stromu lokalizace defektu.
  2. Přidání fotodokumentace
     Uživatel po záznamu místa poškození přidá fotodokumentaci. Učiní tak zvolením možnosti „Přidat foto“, očekávaným výsledkem je zobrazení průvodce soubory operačního systému zařízení, na kterém uživatel pracuje. Po zvolení fotografií a potvrzení výběru se nahrané fotky zobrazí v sekci fotografií. Ke každé z nich se má také zobrazit možnost odebrání dané fotky. Uživatel se také (neúspěšně) pokusí přidat fotografii větší než 15MB, takováto fotografie se ale neukáže v seznamu.

3. Slovní popis poškození
   Posledním krokem tvorby záznamu o defektu je slovní popsání situace. Uživatel vloží text o více než 512 znacích. Očekávaným výsledkem je neplatnost tlačítka k odeslání záznamu. Poté uživatel ubere text tak, aby jeho délka byla pod hranici 512 znaků. Zvolením možnosti „Hotovo“ je ambassador přesměrován zpět do tvorby kontroly, v sekci defektů se má zobrazit nově přidaný záznam.

V tomto scénáři uživatel otestoval možnost přidání nového záznamu o defektu v rámci úpravy či tvorby kontroly. Tento postup zahrnoval záznam místa poškození, přidání fotografií poškození i slovní popis. Celkovým výsledkem je validně vytvořený nový záznam o defektu.

- **Manipulace s aktivními defekty vozu**

  1. Úprava lokace defektu
     Tento uživatelský scénář začíná obdobně jako předchozí, přihlášením uživatele a zvolením „Kontrola vozidla“. Po výběru vozidla a zvolení přidávání nové / upravování poslední kontroly se uživatel dostane do detailu kontroly, který jde editovat. Součástí tohoto režimu komponenty detailu kontroly je také možnost editovat defekty. Kliknutím na příslušné tlačítko defektu má být uživatel přesměrován do komponenty tvorby defektu. Očekávaným výstupem je tedy stránka s možností editace defektu, na které jsou předvyplněné informace k editaci. Postup editace je poté shodný s postupem tvorby kontroly viz scénář 5.2.2.
  2. Smazání záznamu o defektu
     Při úpravě záznamu o defektu je kromě tlačítka „Upravit defekt“ možnost daný defekt i odstranit. Očekávaný výsledek této volby je přesměrování na stránku tvorby/úpravy kontroly a absence právě smazaného defektu v první části této stránky.

Tento scénář završuje testovací scénáře zaměřené na práci s defekty vozu.

- **Tvorba nového záznamu o kontrole vozidla**

  1. Vyplnění informací o vybavení vozu
     Tento scénář předpokládá splnění úvodních kroků stejných jako v předchozích scénářích, tedy přihlášení do aplikace následované zvolením vozu a úpravy či tvorby nové kontroly. Kromě práce s defekty, testované scénáři výše, lze v režimu „editace“ detailu kontroly i měnit symboly indikující funkčnost jednotlivých částí vybavení vozu. Uživatel tedy pro změnu stavu kteréhokoliv z vybavení stiskne tlačítko vedle něj. Očekávaným výsledkem je změna stavu na opačný. Přidat lze též poznámku. Uživatel nejdříve zkusí přidat

poznámku delší než 200 znaků, takovou poznámku (a díky tomu celý upravený záznam o kontrole) nebude možné odeslat.

2. Vyplnění informací o technickém stavu
   V tomto kroku uživatel provede úpravu informací o technickém stavu. V sekci světel funguje přepínání indikace funkčnosti stejným způsobem jako u vybavení vozu. Následně uživatel zadá jako aktuální hodnotu tachometru číslo s počtem číslic vyšším než 7. Takovýto údaj je nevalidní a očekávaným efektem je znemožnění úpravy či odeslání nového záznamu defektu. Stejné pravidlo platí pro část varovných kontrolek a ostatního poškození, zde je ale limit pro délku vstupu roven 200. Pokud uživatel zadá více, bude na nedostatek upozorněn a nebude mu umožněno odeslat výsledek.

- **Výměna pneumatik**

1. Obeznámení s předchozími výsledky výměn pneumatik
   V prvním kroku uživatelského testovacího scénáře pro výměnu pneumatik je potřeba, aby se uživatel přihlásil, zvolil sekci „Výměna pneu“ a následně zvolil také automobil, u kterého chce výměnu zaznamenat. Očekávaným výstupem je přesměrování na seznam všech výměn pneumatik pro daný automobil. Uživatel následně kliknutím na některý prvek seznamu rozšíří danou položku o detailní informace o výměně.
2. Tvorba nového záznamu o výměně
   Dalším krokem je zvolení možnosti „Přidat výměnu“. Tím se uživatel má dostat na stránku formulářového typu, do které následně vyplní všechny požadované hodnoty. Vstupy pro značku automobilu jsou omezeny na 50 znaků, uživatel tedy zadá delší text. Očekávané chování je zamezení odeslání formuláře a indikace problému. U hloubky dezénu je maximální délka vstupu 4 číslice, její překročení vyvolá obdobný efekt.

Uživatelské testovací scénáře jsou příkladem systémového testování a prověřují funkcionalitu celého rozšíření klientské webové aplikace o podporu kontroly vozidel. Částečně postihují i funkcionalitu původní verze aplikace vzhledem k předpokládanému úspěšnému přihlášení uživatele. Žádná funkcionalita rozšíření totiž není dostupná nepřihlášenému uživateli.

Závěr

Cílem této práce byl návrh a implementace rozšíření klientské webové aplikace Uniqway o podporu kontroly vozidel. Mezi dílčí cíle patřila analýza současného stavu aplikace, návrh jednotného způsobu lokalizace defektu na vozidle a otestování aplikace. Nepsaným cílem bylo také uvést čtenáře do problematiky vývoje front-end části webových aplikací. Všechny tyto cíle byly splněny.

Kapitola 2 splňuje cíl představit čtenáři vývoj webu, konkrétně části front-end. Uvádí čtenáře do problematiky definicí základních pojmů a vysvětlení konceptů. Dále se kapitola věnuje použitému javascriptovému frameworku Vue.js. Literární rešerše je zakončena popisem webových aplikací v Uniqway a analýzou původního stavu klientské webové aplikace.

Praktická část počíná kapitolou 3, ve které je popsán průběh návrhu celého rozšíření aplikace, včetně definice struktury kontroly a identifikace místa poškození automobilu. Navrhuje i komunikaci aplikace se serverem.

Následující kapitola popisuje proces a výslednou strukturu implementace rozšíření, zahrnuje i popis navržené implementace se serverem. Součástí toho je i výčet všech použitých rozšíření frameworku Vue.js.

Testováním se zabývá kapitola 5. Krom automatizovaných testů přináší i uživatelské scénáře pro fyzického testera, které budou po implementaci serverové části aplikace provedeny.

Krom splněných cílů bylo nad rámec zadání vypracováno několik vlastností navíc. Mezi ně se řadí například povinnost slovně okomentovat vyřešení defektu při jeho odstraňování z evidence, což pomůže budoucí optimalizaci oprav poškození. Další je například evidence typů doplňovaných provozních kapalin vozidla. Tato vlastnost je důležitá zejména kvůli návaznosti na další kontrolu, kdy se může správce při doplňování kapalin na tento aspekt předchozí kontroly ohlížet.

Budoucí vylepšení aplikace

Mezi budoucí vylepšení rozšíření klientské webové aplikace pro kontrolu vozidel patří například exportování dat. Možnost exportu dat o kontrolách či defektech do různých formátů by ulehčila komunikaci o stavu automobilů s lidmi, kteří do aplikace nemají přístup.

Dalším aspektem, který by zvýšil všeobecnou použitelnost aplikace, je možnost filtrování seznamů kontrol či defektů. To je možné realizovat z několika hledisek, například dle data založení nebo uživatele, který kontrolu či defekt zaznamenával.

V neposlední řadě lze aplikaci vylepšit přizpůsobením pro řazení seznamů (kontrol, defektů, výměn pneumatik) podle určitých kritérií. Samotná logika řazení by byla implementována na serveru kvůli co nejnižšímu zatížení klientského zařízení, avšak aplikaci je nutné na tuto možnost připravit.

Bibliografie

1. *A Brief History of the Internet* [online]. University System of Georgia, 2021 [cit. 2021-03-10]. Dostupné z: https://www.usg.edu/galileo/skills/unit07/internet07_02.phtml. (Překlad autora).
2. P. CHRISTENSSON. GUI. *Tech Terms* [online]. 2006 [cit. 2021-03-19]. Dostupné z: https://techterms.com/definition/gui. (Překlad autora).
3. CHRISTENSSON, P. User Interface. *Tech Terms* [online]. 2009 [cit. 2021-03-19]. Dostupné z: https://techterms.com/definition/user_interface. (Překlad autora).
4. POWELL, Adam. Web 101: A History of the GUI. *Wired* [online]. 1997 [cit. 2021-03-08]. Dostupné z: https://www.wired.com/1997/12/web-101-a-history-of-the-gui/. (Překlad autora).
5. VILLINGER, Sandro. What is RAM and Why is it Important? *Avast* [online]. 2019 [cit. 2021-03-19]. Dostupné z: %7Bhttps://www.avast.com/c-what-is-ram-memory%7D. (Překlad autora).
6. SWARENGIN, Shawnee. 10 Most Popular Apps on the Apple and Google Play App Stores. *CleverTap* [online]. 2020 [cit. 2021-03-15]. Dostupné z: https://clevertap.com/blog/most-popular-apps/. (Překlad autora).
7. KOZÁK, David. Jak fungují webové služby. *Interval* [online]. 2002 [cit. 2021-03-09]. Dostupné z: https://www.interval.cz/clanky/jak-funguji-webove-sluzby/. (Překlad autora).
8. BRILL, G. *CodeNotes for Web Services in Java and .NET*. Random House Publishing Group, 2002. ISBN 9781588362544. Dostupné také z: https://books.google.cz/books?id=SVdSUcdONukC. (Překlad autora).

9. PAVLOVIČ, Jan. *Webové služby, přednáška 11, Vývoj programových systémů v jazyce Java* [online]. Fakulta informatiky, Masarykova univerzita, 2004 [cit. 2021-03-13]. Dostupné z: https://is.muni.cz/el/fi/podzim2004/PA165/um/prednaska11/index.html.

10. W3SCHOOLS. *XML tutorial* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://www.w3schools.com/xml/. (Překlad autora).

11. ORACLE. *What Are RESTful Web Services?* [online]. 2013 [cit. 2021-03-14]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. Technická zpráva. Oracle. (Překlad autora).

12. THOMPSON, Henry S. What's a URI and why does it matter? *School of informatics, University of Edinburgh* [online]. 2010 [cit. 2021-03-19]. Dostupné z: %7Bhttp://www.ltg.ed.ac.uk/~ht/WhatAreURIs/%7D. (Překlad autora).

13. HOWE, Denis. ISO/OSI Reference Model. *The Free On-line Dictionary of Computing* [online]. 2003 [cit. 2021-03-15]. Dostupné z: %7Bhttps://encyclopedia2.thefreedictionary.com/ISO%5C%2fOSI+Reference+Model%7D. (Překlad autora).

14. MOZILLA CONTRIBUTORS. *HTTP* [online]. 2021 [cit. 2021-03-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP. (Překlad autora).

15. R. FIELDING ET AL. *Hypertext Transfer Protocol – HTTP/1.1* [online]. 1999 [cit. 2021-03-27]. Dostupné z: https://tools.ietf.org/html/rfc2616#page-8. Technická zpráva. The Internet Society. (Překlad autora).

16. MOZILLA CONTRIBUTORS. *Idempotent* [online]. 2021 [cit. 2021-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Idempotent. (Překlad autora).

17. MOZILLA CONTRIBUTORS. *Safe* [online]. 2021 [cit. 2021-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/safe. (Překlad autora).

18. FLORIDA STATE UNIVERSITY. Cache and Cookies. *ITS Knowledge Base* [online] [cit. 2021-03-18]. Dostupné z: https://faq.its.fsu.edu/support-services/cache-and-cookies. (Překlad autora).

19. MOZILLA CONTRIBUTORS. *Cacheable* [online]. 2021 [cit. 2021-03-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/cacheable. (Překlad autora).

20. LINDLEY, Cody. *Frontend developer handbook*. Frontend Masters, 2018. Dostupné také z: https://frontendmasters.com/books/front-end-handbook/2018/what-is-a-FD.html. (Překlad autora).

21. DOSTÁLOVÁ, Zuzana. Frontend vs. Backend. *Czechitas* [online]. 2014 [cit. 2021-03-16]. Dostupné z: `https://www.czechitas.cz/cs/blog/zaciname-s-it/frontend-vs-backend`.

22. BERNHAUER, David. *HyperText Markup Language, Tvorba webových aplikací BI-TWA*. Fakulta informačních technologií, katedra Softwarového inženýrství, ČVUT v Praze, 2020.

23. BOS, Bert. *Cascading Style Sheets* [online]. 2021 [cit. 2021-03-17]. Dostupné z: `https://www.w3.org/Style/CSS/`. Technická zpráva. W3C. (Překlad autora).

24. ŽÁRA, O. *JavaScript*. Albatros Media a.s., 2015. ISBN 9788025145937. Dostupné také z: `https://books.google.cz/books?id=BBy2DwAAQBAJ`.

25. TUNG, Liam. Programming language popularity: JavaScript leads – 5 million new developers since 2017. *ZDNet* [online]. 2020 [cit. 2021-03-14]. Dostupné z: `https://www.zdnet.com/article/programming-language-popularity-javascript-leads-5-million-new-developers-since-2017/`. (Překlad autora).

26. MOZILLA CONTRIBUTORS. *Dynamic typing* [online]. 2021 [cit. 2021-03-19]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing`. (Překlad autora).

27. W3SCHOOLS. *JavaScript Objects* [online]. 2021 [cit. 2021-03-17]. Dostupné z: `https://www.w3schools.com/js/js_objects.asp`. (Překlad autora).

28. MORRIS, Scott. TECH 101: What is a Javascript framework? Here's everything you need to knwo. *skillcrush* [online] [cit. 2021-03-18]. Dostupné z: `https://skillcrush.com/blog/what-is-a-javascript-framework/#whatis`. (Překlad autora).

29. MOZILLA CONTRIBUTORS. *What is AJAX?* [online]. 2021 [cit. 2021-03-16]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started`. (Překlad autora).

30. V. SOLOVEI, O. OLSHEVSKA, Y. BORTSOVA. The difference between developing single page application and traditional web application based on Mechatronics robot laboratory Onaft application. *Odessa National Academy of Food Technologies* [online]. 2018 [cit. 2021-03-16]. Dostupné z: `https://journals.onaft.edu.ua/index.php/atbp/article/view/874`. (Překlad autora).

31. DAVIS, H. *Search Engine Optimization*. O'Reilly Media, 2006. ISBN 9781491905883. Dostupné také z: `https://books.google.cz/books?id=hYSMBAAAQBAJ`. (Překlad autora).

32. P. KLAUZINSKI, J. MOORE. *Mastering JavaScript Single Page Application Development*. Packt Publishing, 2016. ISBN 9781785886447. Dostupné také z: https://books.google.cz/books?id=KpncDgAAQBAJ. (Překlad autora).

33. CHRISTENSSON, P. API Definition. *Tech Terms* [online]. 2016 [cit. 2019-04-11]. Dostupné z: https://techterms.com/definition/api. (Překlad autora).

34. GOOGLE. *Introduction to Angular concepts* [online]. 2021 [cit. 2021-04-13]. Dostupné z: https://angular.io/guide/architecture. (Překlad autora).

35. GURU99. MVC Tutorial for Beginners: What is, Architecture & Example. *Guru99* [online]. 2021 [cit. 2021-03-16]. Dostupné z: https://www.guru99.com/mvc-tutorial.html. (Překlad autora).

36. DAITYARI, Shaumik. Angular vs React vs Vue: Which Framework to Choose in 2021. *codeinwp* [online]. 2021 [cit. 2021-03-16]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/. (Překlad autora).

37. W3SCHOOLS. *JavaScript HTML DOM* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://www.w3schools.com/js/js_htmldom.asp. (Překlad autora).

38. FACEBOOK. *Virtual DOM and Internals* [online]. 2021 [cit. 2021-04-17]. Dostupné z: https://reactjs.org/docs/faq-internals.html. (Překlad autora).

39. CODECADEMY. React: The Virtual DOM. *Codeacademy* [online]. 2021 [cit. 2021-03-16]. Dostupné z: https://www.codecademy.com/articles/react-virtual-dom. (Překlad autora).

40. BOOTSTRAP. *Build fast, responsive sites with Bootstrap* [online]. 2021 [cit. 2021-03-19]. Dostupné z: https://getbootstrap.com. (Překlad autora).

41. W3SCHOOLS. *HTML Responsive Web Design* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://www.w3schools.com/html/html_responsive.asp. (Překlad autora).

42. YOU, Evan. *The Vue Instance* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://vuejs.org/v2/guide/instance.html. (Překlad autora).

43. YOU, Evan. Vue Mastery. *Vue Mastery* [online]. 2021 [cit. 2021-03-19]. Dostupné z: https://www.vuemastery.com. (Překlad autora).

44. YOU, Evan. *Components Basics* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://vuejs.org/v2/guide/components.html. (Překlad autora).

45. YOU, Evan. *Computed Properties And Watchers* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://vuejs.org/v2/guide/computed.html. (Překlad autora).

46. YOU, Evan. *Instance Lifecycle Hooks* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks. (Překlad autora).

47. YOU, Evan. *Vue Directives* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://vuejs.org/v2/guide/syntax.html#Directives. (Překlad autora).

48. YOU, Evan. *What is a "State Management Pattern"?* [online]. 2021 [cit. 2021-04-19]. Dostupné z: https://vuex.vuejs.org/#what-is-a-state-management-pattern. (Překlad autora).

49. HEUSSER, Matt. debugging. *TechTarget* [online]. 2019 [cit. 2021-03-15]. Dostupné z: %7Bhttps://searchsoftwarequality.techtarget.com/definition/debugging%7D. (Překlad autora).

50. YOU, Evan. *State* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://vuex.vuejs.org/guide/state.html#single-state-tree. (Překlad autora).

51. ŠIDLOVSKÝ, Marko. *Webová aplikace správy vozového parku pro systém sdílení automobilů více uživateli*. 2017. Bakalářská práce. FEL ČVUT.

52. SEVERA, Štěpán. *Webová aplikace pro uživatele systému sdílení automobilů*. 2019. Bakalářská práce. FIT ČVUT.

53. BRDA, Jiří. Co je UX design a kde se s ním setkáte. *Jiří Brda* [online]. 2016 [cit. 2021-03-20]. Dostupné z: %7Bhttp://www.jiribrda.cz/co-je-ux-design-a-kde-se-s-nim-setkate.html%7D.

54. MLEJNEK, Jiří. *Analýza a sběr požadavků, přednáška 3, Softwarové inženýrství BI-SI1*. Fakulta informačních technologií, katedra Softwarového inženýrství, ČVUT v Praze, 2019.

55. HANNAH, Jaye. What Exactly Is Wireframing? A Comprehensive Guide. *Career Foundry* [online]. 2019 [cit. 2021-03-27]. Dostupné z: %7Bhttps://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/%7D. (Překlad autora).

56. MOCKOON. *Create mock APIs in seconds* [online]. 2017 [cit. 2021-03-29]. Dostupné z: https://mockoon.com. (Překlad autora).

57. ZABRISKIE, Matt. *Axios* [online]. 2021 [cit. 2021-04-02]. Dostupné z: https://github.com/axios/axios. (Překlad autora).

58. MOZILLA CONTRIBUTORS. *Promise* [online]. 2021 [cit. 2021-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. (Překlad autora).

59. YOU, Evan. Introduction – Vue Router. *Vue Router* [online]. 2021 [cit. 2021-04-07]. Dostupné z: `https://router.vuejs.org/`. (Překlad autora).

60. YOU, Evan. What is Vuex? *Vuex* [online]. 2019 [cit. 2019-04-07]. Dostupné z: `https://vuex.vuejs.org/`. (Překlad autora).

61. FACEBOOK. *Jest* [online]. 2021 [cit. 2021-04-11]. Dostupné z: `https://jestjs.io/en/`. (Překlad autora).

62. MLEJNEK, Jiří. *Testování aplikací, přednáška 9, Softwarové inženýrství BI-SI1*. Fakulta informačních technologií, katedra Softwarového inženýrství, ČVUT v Praze, 2021.

Příloha A

Seznam zkratek

AJAX Asynchronous javascript and xml

API Application programming interface

CSS Cascading style sheets

DOM Document object model

FURPS Functionality, usability, reliability, performance, supportability

GUI Graphical user interface

HTML Hypertext markup language

HTTP Hypertext Transfer Protocol

JSON Javascript object notation

MPA Multiple page application

MVC Model, view, controller

RAM Random access memory

REST Representational state transfer

SEO Search engine optimization

SOAP Simple object access protocol

SPA Single page application

UC Use case

UI User interface

URI Uniform resource identifier

URL Uniform resource locator

UX User experience

WSDL Web services description language

XML Extensible markup language

XSS Cross-site scripting

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové soubory
  - `impl` ..... zdrojové kódy
    - `CarChecks` ..... zdrojové kódy implementace
    - `tests` ..... zdrojové kódy jednotkových testů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP-Gutwirth-Jiri-2020.pdf` ..... text práce ve formátu PDF
  - `zadani.pdf` ..... zadání práce ve formátu PDF

Příloha C

Evidované údaje o technickém stavu vozu

- hladina chladící kapaliny,
- hladina motorového oleje,
- hladina kapaliny do ostřikovačů,
- stav tachometru,
- funkčnost světel,
- svícení chybových kontrolek po startu,
- ostatní možné závady,
- způsobilost k dalšímu provozu

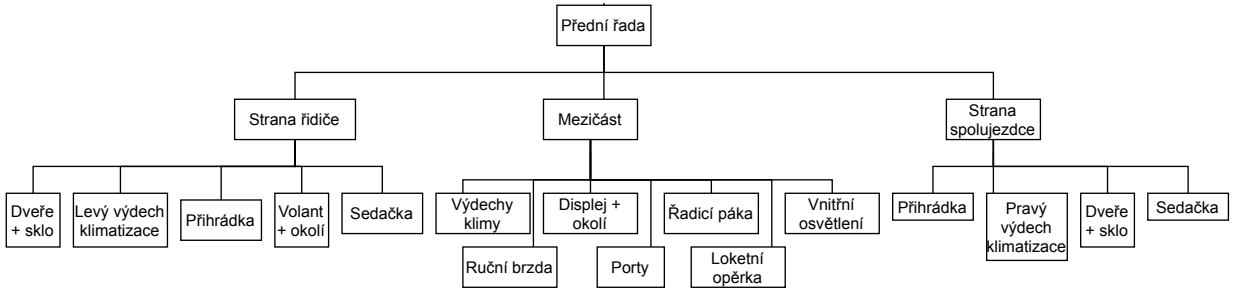
Příloha D

Seznam evidovaného vybavení vozu

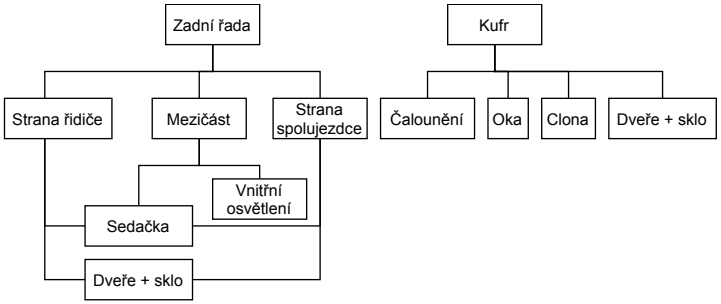
- lékárnička,
- žárovky,
- lepící sada,
- výstražný trojúhelník,
- reflexní vesty,
- dobíjecí kabely,
- držák na telefon,
- třífázový kabel na 230V,
- čip PRE (pouze u Škoda CITIGOe iV),
- čip ČEZ,
- karta CCS,
- karta VŠE,
- MB ŠKODA karta,
- zelená karta,
- technický průkaz,
- dálniční známka

Příloha E

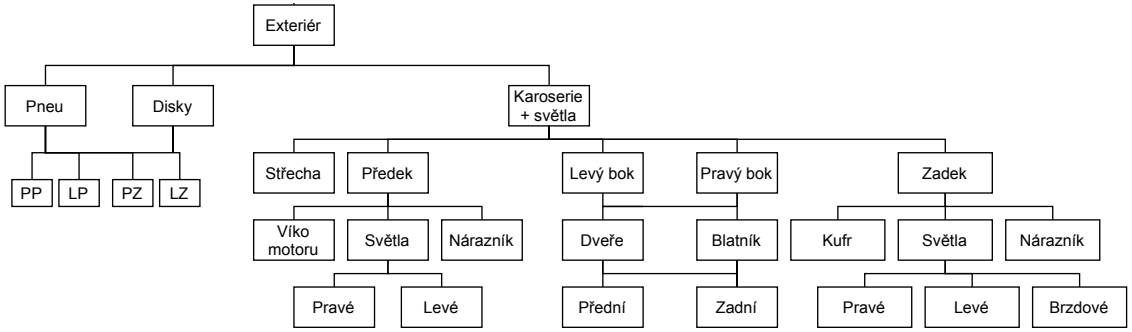
Kompletní podoba stromu lokalizace defektu



Obrázek E.1: Interiérová část stromu lokalizace defektu – přední řada interiéru



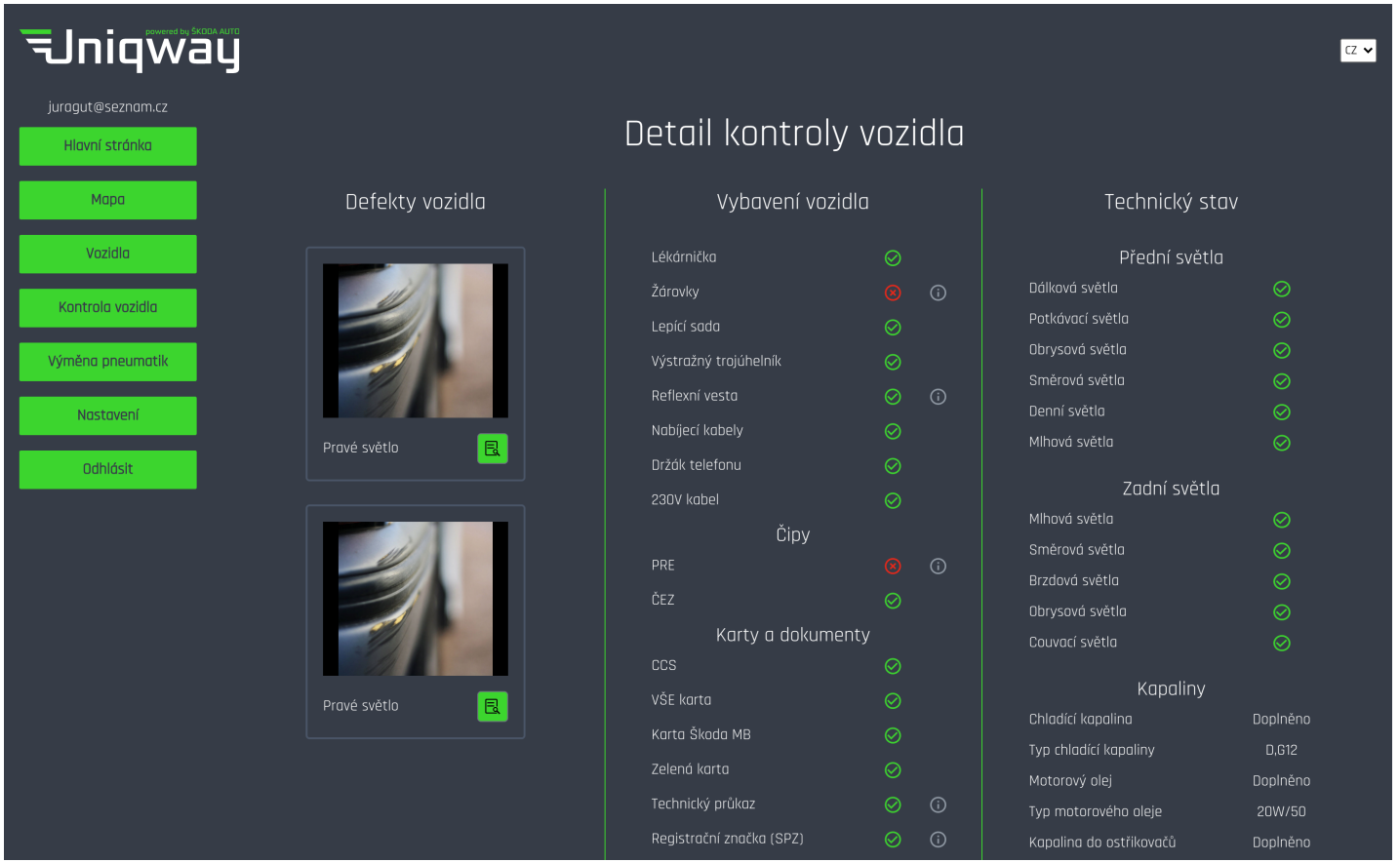
Obrázek E.2: Interiérová část stromu lokalizace defektu – zadní řada interiéru a kufr



Obrázek E.3: Exteriérová část stromu lokalizace defektu

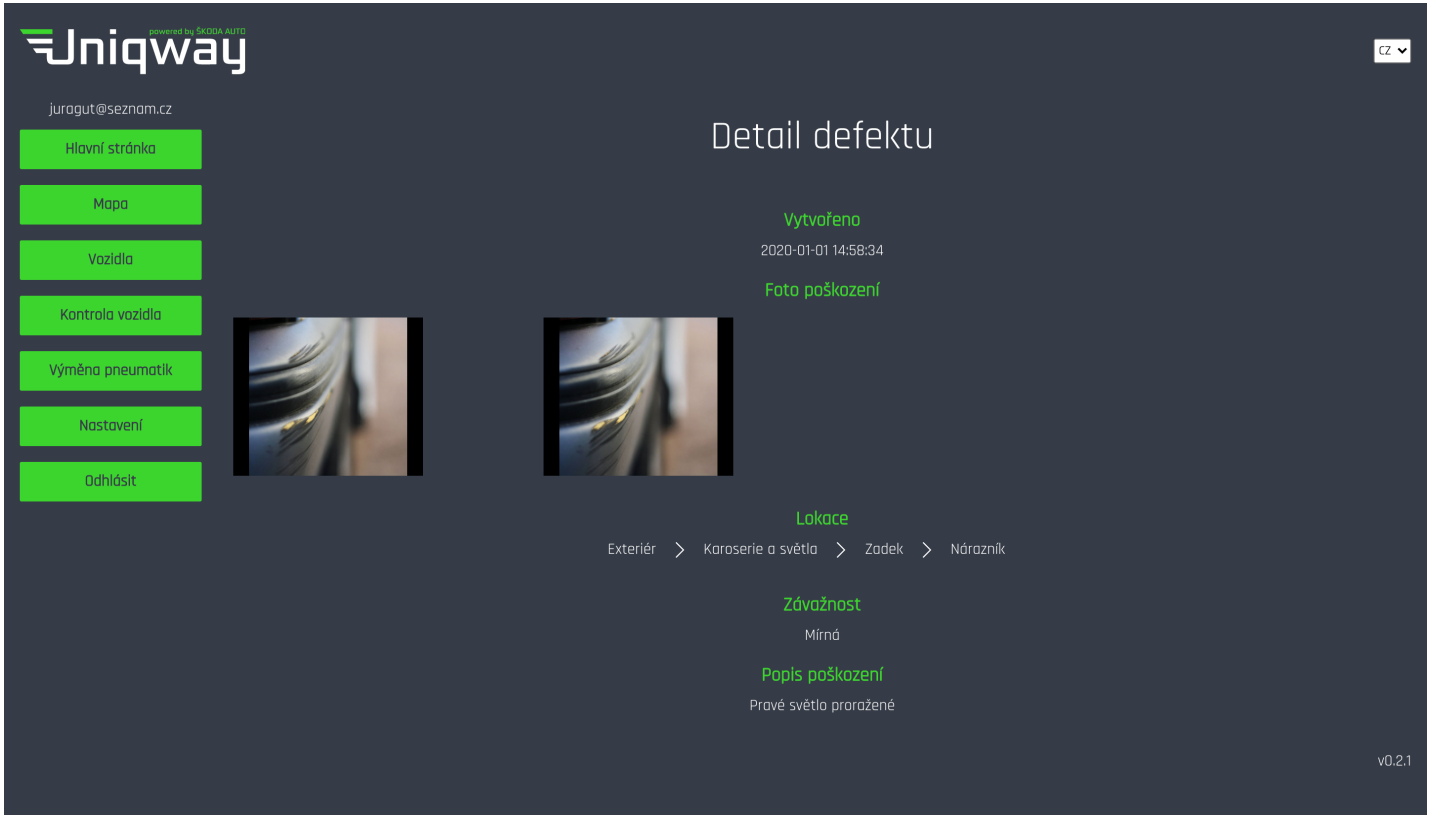
Příloha F

Snímky obrazovek

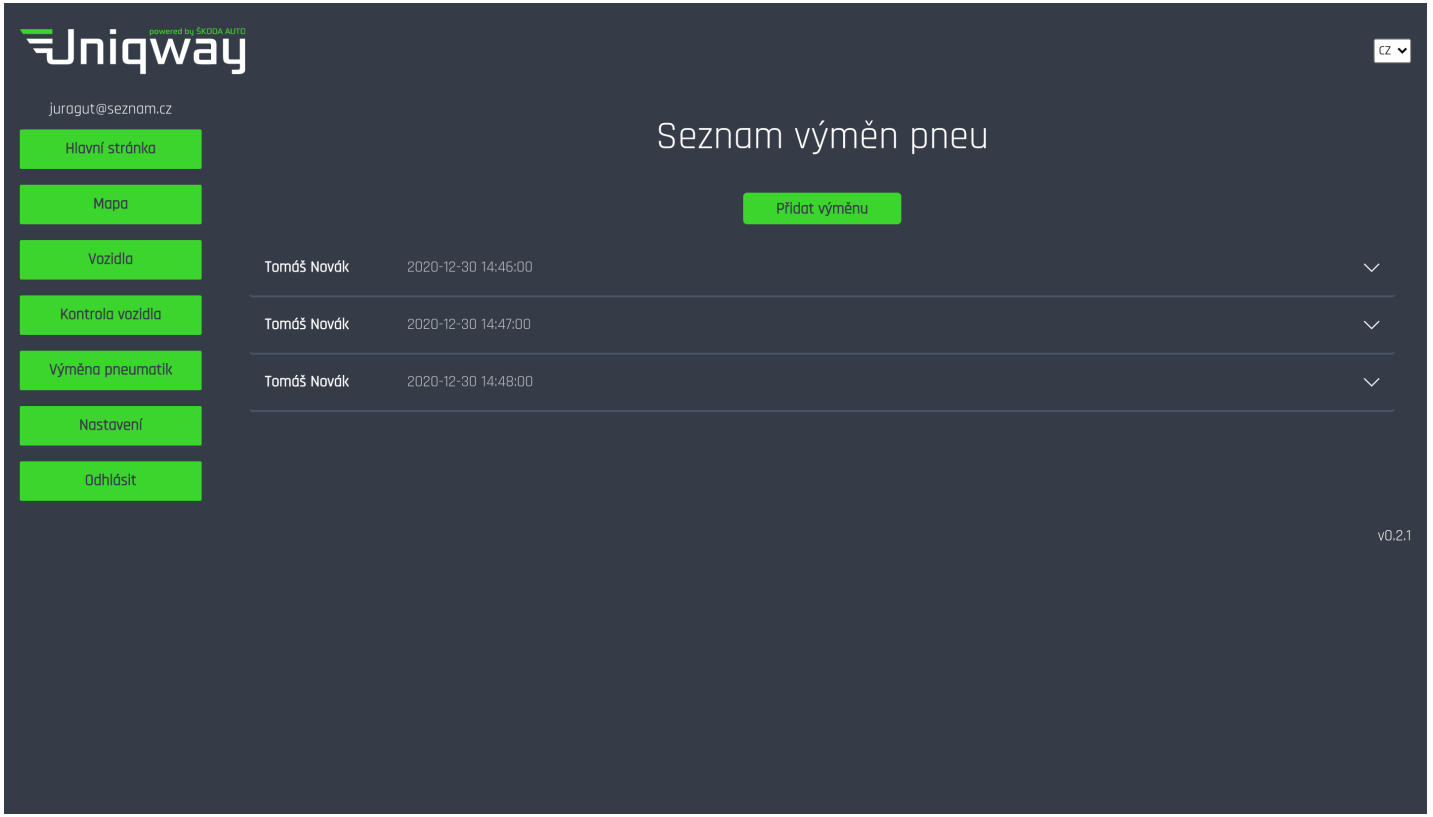


Obrázek F.1: Snímek obrazovky detailu záznamu o kontrole vozu

Obrázek F.2: Snímek obrazovky seznamu kontrol vozu

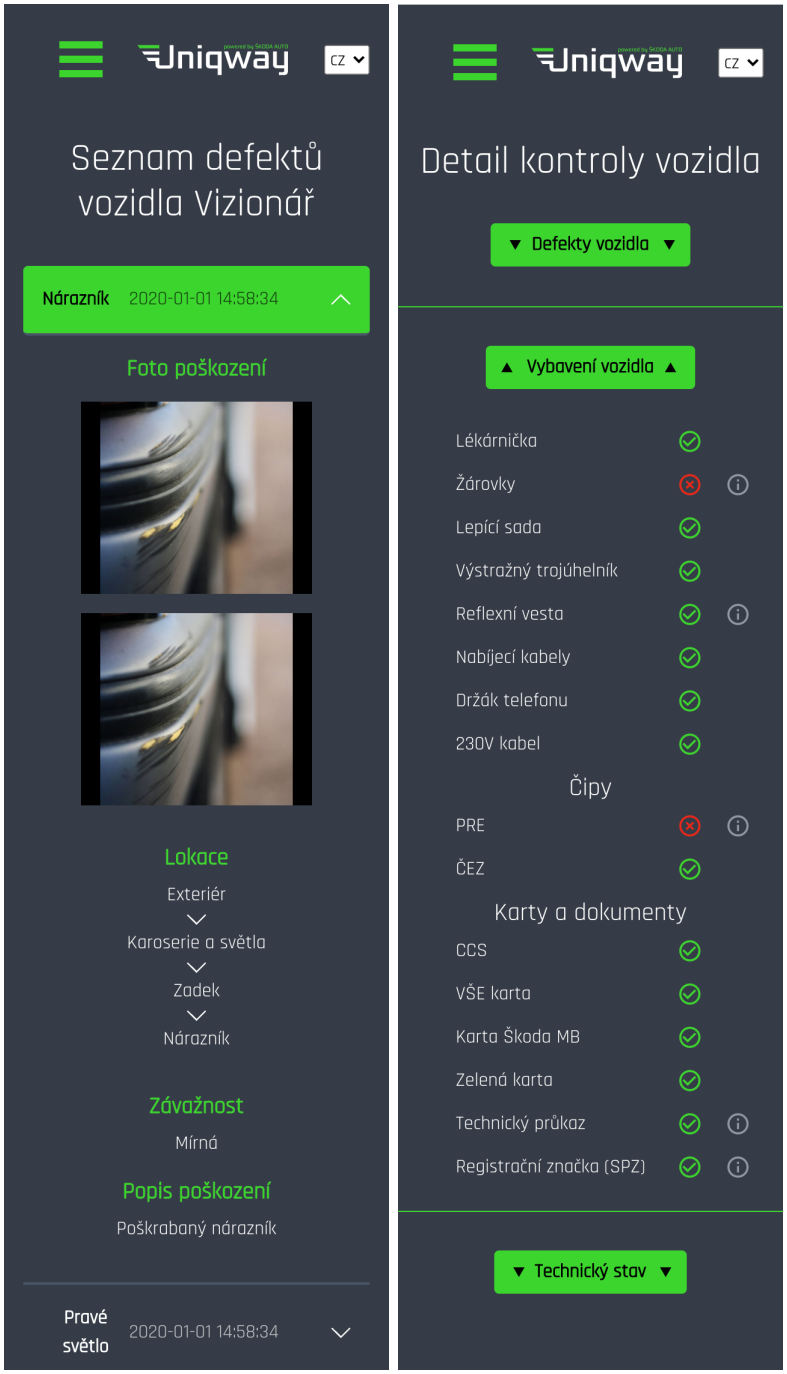


Obrázek F.3: Snímek obrazovky detailu záznamu o defektu vozu



Obrázek F.4: Snímek obrazovky seznamu záznamů o výměně pneumatik vozu

Obrázek F.5: Snímek obrazovky tvorby nového záznamu o výměně pneumatik vozu



Obrázek F.6: Snímky obrazovky detailu kontroly a seznamu aktuálních poškození vozidla v mobilní verzi