TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro monitorování provozu vozidel přepravní společnosti

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Zdeněk Devátý**
Vedoucí práce: Ing. Igor Kopetschke
Konzultanti: Ing. Michal Bohuslávek
Bc. Mojmír Křížek
Ing. Patrik Drhlík

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2019/2020**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení:	**Bc. Zdeněk Devátý**
Název práce:	**Aplikace pro monitorování provozu vozidel přepravní společnosti**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**
Vedoucí práce:	**Ing. Igor Kopetschke**
Rozsah práce:	**40—50 stran**

Zásady pro vypracování:

1. Proveďte analýzu potřebných funkcionalit pro výslednou aplikaci tak, aby pokryly potřeby přepravní společnosti.
2. Proveďte rešerši již existujících SW řešení a zdůvodněte potřebu vytvořit vlastní software.
3. Navrhněte mobilní aplikaci reflektující požadavky na základě bodu 1). Aplikace bude komunikovat se serverem společnosti přes RESTful API.
4. Implementujte aplikaci a vytvořte dokumentaci.
5. Pokud to bude možné, získejte zpětnou vazbu s testovacího, případně již ostrého provozu.

Seznam odborné literatury:

[1] Richardson L., Ruby S.: RESTful Web Services: Web services for the real world, O'Reilly Media, 2007, ISBN-13: 978-0596529260

[2] Lacko L.: Vývoj aplikací pro Android, Computer Press, 2015, ISBN-13: 978-8025143476

V Liberci dne .....

.....
Ing. Igor Kopetschke

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

4. 1. 2021 Bc. Zdeněk Devátý

Aplikace pro monitorování provozu vozidel přepravní společnosti

Abstrakt

Cílem práce je vytvořit mobilní aplikaci umožňující dispečinku přepravní společnosti sledovat pohyb a stav vozidel. Jsou zde analyzovány potřebné funkce aplikace a z nich je vytvořena technická specifikace. Následně je navržena a implementována multiplatformní mobilní aplikace komunikující přes dané rozhraní s informačním systémem společnosti. Práce zahrnuje í zkušenosti z testovacího provozu a specifický postup při nasazování aplikace pro zachování zpětné kompatibility.

Klíčová slova: Mobilní aplikace, přeprava, logistika

Vehicle monitoring application

Abstract

The goal of this thesis is to create a mobile application allowing vehicle monitoring to transportation company dispatchers. There are analyzed the required application features, and a technical specification is created from them. Subsequently, a multiplatform mobile application communicating via a given interface with the company's information system is designed and implemented. The thesis also includes experience from test operation and a specific procedure for deploying the application to maintain backward compatibility.

Keywords: Mobile application, transportation, logistics

Poděkování

Rád bych poděkoval všem, kteří mě během tvorby této práce podpořili a díky kterým pro mě celé studium na univerzitě bylo lidsky í vědomostně přínosné.

Za konzultace návrhů a struktury systému, za úvod do frameworku Ionic a za rady ohledně programování v JavaScriptu děkuji svým konzultantům.

Za porozumění principům informatiky, přehled v tomto širokém oboru í mimo něj a za předané lidské hodnoty děkuji svým učitelům.

Za lásku, lidskost a úžasnou podporu během studia, před ním í po něm, děkuji svým rodičům.

Za podporu obzvlášť ve chvílích, kdy se mi vůbec nechtělo psát děkuji své přítelkyni Aničce.

Za nezapomenutelná studentská léta děkuji svým přátelům a spolubydlícím. Járovi, v jehož přítomnosti neexistuje nuda, pouze nejistota, kdo bude terčem dalšího vtipu. Lukášovi, který dělá robota a už nenosí boty značky Nike, děkuji za arašídové máslo. Vítkovi děkuji za inventury a patetické diskuse nad kafíčkem. Mirkovi, Pu!

Obsah

Seznam zkratek

API	Application Programming Interface
JVM	Java Virtual Machine
AVD	Android Virtual Device
ADB HTTP	Hypertext Transfer Protocol
NPM	Node Package Manager
UI	User Interface - Uživatelské rozhraní
GUI	Graphical User Interface - Grafické uživatelské rozhraní
TDD	Test-Driven Development
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
REST	Representational State Transfer
JSON	JavaScript Object Notation
TLS	Transport Layer Security
OS	Operační systém
IS	Informační systém

Úvod

Dopravní společnost požádala o vývoj nového systému na správu vozového parku. Tento systém by měl nahradit systém stávající, který je příliš nákladný na údržbu a rozšiřování.

Tato práce se zabývá návrhem a implementací mobilní aplikace pro nový systém. Aplikace má sloužit jako komunikační rozhraní mezi řidiči a dispečinkem společnosti a předávat do informačního systému informace o jízdě a vozidle.

Před vývojem jsou stanoveny požadavky a na jejich základě jsou zkoumána dostupná komerční řešení. Poté je podle požadavků navržena a implementována multiplatformní mobilní aplikace sloužící jako tenký klient pro sledování provozu vozidel.

Pozornost je věnována podpoře velkého množství zařízení a odolnosti aplikace vůči uživatelským chybám a vnějším vlivům.

Aplikace má být vyvíjena s důrazem na budoucí údržbu a rozšiřování, proto má být zvláštní pozornost věnována návrhu aplikace a srozumitelnosti kódu.

Při vývoji je dbáno na plynulost přechodu mezi stávajícím a novým systémem a na zachování zpětné kompatibility.

Dokončená aplikace je postupně nasazována a testována v provozu. Na základě výsledků tohoto testování jsou laděny chyby a upravována funkcionalita.

1 Specifikace požadavků

Vlastnosti, které jsou od aplikace požadovány, jsou rozděleny na funkční, nefunkční a technické.

Funkční požadavky obsahují požadavky, které specifikují, co má aplikace umět. Nefunkční požadavky specifikují požadavky na návrh a implementaci aplikace, které přímo neupravují funkcionalitu aplikace. Technické požadavky upřesňují konkrétní omezení technického charakteru, na které je nutné brát zřetel při návrhu a implementaci aplikace.

1.1 Funkční požadavky

Aplikace má sloužit dispečinku ke sledování stavu vozidel společnosti. Z toho vyplývají následující požadavky.

1.1.1 Jízdní režimy

Řidič by měl mít možnost předávat dispečinku pomocí aplikace informaci o tom, jakou činnost právě vykonává. Aplikace by měla mít režimy signalizující rozvoz zboží, nakládání či vykládání zboží, tankování, pauzu a další.

1.1.2 Sledování provozu

Hlavní funkcí aplikace je sledování provozu vozidel. Aplikace by měla zaznamenávat polohu vozidla v čase a předávat tuto informaci informačnímu systému.

Sledování polohy by mělo probíhat, í když je displej zařízení zhasnutý, nebo je aktivní jiná aplikace. Poloha by se také měla zaznamenávat nezávisle na dostupnosti Internetového připojení.

Řidiči mohou vozidla používat také pro osobní jízdy. Při volbě soukromého jízdního režimu může být poloha zaznamenávána, ale pouze pro účely měření vzdálenosti. v tomto režimu dispečink nemá mít přístup k poloze vozidla, poloha vozidla tedy nebude odesílána na server.

Při volbě režimů, ve kterých vozidlo stojí, může být poloha také zaznamenávána, ale tato informace pro IS není relevantní, takže také bude sloužit pouze pro interní účely aplikace.

1.1.3 Zadávání dodatečných informací

Dispečink potřebuje mít přehled o tom, kolik nákladu vezou jednotlivá vozidla. Zejména u cisteren, které vezou stejnou kapalinu na více míst, je toto důležitá informace.

Je tedy žádoucí, aby řidiči měli možnost zadat do aplikace množství převážené látky, které naložili a vyložili. Dispečink tím získá přehled o aktuálním naložení vozidla a o tom, jaké množství nákladu bylo ve kterém místě naloženo nebo vyloženo.

Pro pravidelnou údržbu vozidla je důležité přenášet také informaci o stavu odometru, aby mohl být v informačním systému přehled o tom, která vozidla potřebují servisní prohlídku. Tuto informaci může vkládat do aplikace řidič pomocí formuláře. Spolehlivější možností je získávat tuto informaci přímo z řídící jednotky vozidla přes diagnostické rozhraní. Tím by bylo možné přenášet í další diagnostické informace, které by pomohly jak s udržováním vozidel v dobrém stavu, tak s řešením technických problémů vozidel na cestách.

Další užitečnou informací je průměrná spotřeba paliva jednotlivých vozidel. Když aplikace umožní řidičům zadávat množství natankovaného paliva a ujetých kilometrů mezi tankováním (odvozené ze stavu odometru), je možné spočítat průměrnou spotřebu vozidla. Na základě této informace může společnost podniknout kroky ke snížení svých nákladů na paliva í snížení dopadu na životní prostředí.

Aplikace by měla tedy sloužit í jako komunikační prostředek mezi řidičem a dispečinkem.

1.1.4 Propojení s IS

Přepravní společnost používá vlastní informační systém, který zajišťuje zobrazování tras a informací o vozidlech. Z těchto dat také generuje knihy jízd. Tento informační systém může být využit k ovládání aplikace nebo dalšímu využití dat.

Aplikace má tedy data předávat tomuto informačnímu systému. k tomuto účelu slouží serverová část aplikace (dále backend), která zajišťuje veškerou komunikaci mezi serverem a jednotlivými zařízeními. Tento backend data na serveru přijímá a předává informačnímu systému, na což má patřičná oprávnění. Zajišťuje také autenizaci uživatelů a nastavení parametrů aplikace z IS.

1.2 Nefunkční požadavky

Aplikace má být odolná vůči chybám uživatelů a vnějším vlivům. Vzhledem k citlivé povaze dat o poloze vozidel by měla být také dobře zabezpečená.

1.2.1 Snadné ovládání

Aplikaci mají využívat převážně řidiči z povolání. Tito řidiči mají různé úrovně technických schopností, ale zejména pro ně není používání aplikace hlavní pracovní náplní. Ovládání aplikace by tedy mělo být uzpůsobeno k tomu, aby řidiči neztráceli ovládáním více času a vjemových schopností, než je nezbytně nutné.

V první řadě je tedy žádoucí zachovat logiku ovládání původní verze aplikace. Volení jízdích režimů, zadávání údajů o stavu vozidla í rozložení ovládacích prvků by mělo zůstat podobné původní aplikaci, aby byl pro uživatele přechod co možná nejsnazší.

Ovládání by mělo být co nejintuitivnější. Například tlačítka by měla být reprezentována přehlednými ikonami a všechny důležité informace by měly být vidět najednou.

1.2.2 Odolnost vůči uživatelským chybám

Při používání aplikace se mohou její uživatelé dopustit chyb či překlepů. v případě, že uživatel zvolí omylem špatný režim, měla by aplikace umožnit návrat do režimu předchozího. Aplikace by neměla dovolit uživateli opustit formulář bez vyplnění povinných údajů a měla by pomocí heuristických odhadů zabránit odeslání údajů, které jsou pravděpodobně zadané nesprávně.

Aplikace má také zabránit stavům, kdy řidič například zapomene zvolit jízdní režim, zapnout připojení k Internetu nebo povolit aplikaci přístup k poloze zařízení. v takových případech by měla aplikace na tuto skutečnost řidiče upozornit.

1.2.3 Nezávislost na připojení

Vzhledem k tomu, že aplikace má být provozována na mobilních zařízeních, bude připojení k Internetu zajištěno pouze prostřednicvím mobilních sítí. Ne všechna místa jsou pokryta mobilním signálem a přenosy dat nemusí být stabilní či vždy dostupné. Proto by měla aplikace být schopna fungovat v co největší míře bez připojení k serveru.

1.3 Technické požadavky

Aplikace bude provozována převážně na zařízeních s OS Android. Nasazování aplikace má probíhat tak, aby se předešlo komplikacím s přesunem.

1.3.1 Cílová zařízení

Ve vozech dopravní společnosti jsou nainstalovány mobilní telefony. Všechny telefony mají operační systém Android. Dle průzkumu dat ze zařízení se verze pohybují od verze 5.1 (API úrovně 22[a]) do nejnovějších verzí (brzy začne společnost rozmisťovat zařízení s OS Android verze 11 (API úrovně 30)). Aplikace tedy musí podporovat OS Android od verze 5.1.

V plánu je také instalace zařízení do palubní desky s dotykovou obrazovkou o velikosti okolo 10 palců. Tato zařízení rovněž budou používat OS Android, avšak je třeba brát v potaz rozdílný poměr stran a velikost obrazovky.

Někteří zaměstnanci společnosti, například obchodní cestující, však chtějí používat aplikaci na svých soukromých zařízeních. Některé z těchto zařízení mají operační systém

[a] API level - verze programovacího rozhraní OS Android (`http://developer.android.com/guide/appendix/api-levels.html`). API úrovně jsou z velké části dopředně kompatibilní.

Apple iOS. Je zde proto požadavek, aby bylo možné do aplikace přidat podporu í pro tento operační systém.

V této práci není sice cílem zahrnout podporu pro iOS, ale je nutné na tento požadavek brát ohled při návrhu a volbě technologií.

1.3.2 Postupné nasazení

Vzhledem k tomu, že původní, nahrazovaná aplikace je aktivně používaná, je nutné zajistit nepřerušený provoz. Cílem je tedy omezit vznik zásadních chyb v systému a také zásadních změn v uživatelském rozhraní, aby uživatelé zvládli přechod mezi aplikacemi bez problémů.

Byl proto sestaven postupný plán nasazení:

- Vývoj nové aplikace komunikující s původním rozhraním.
  - Nová aplikace co nejpřesněji replikuje funkcionalitu, vzhled a rozhraní původní aplikace.
- Testování a ladění chyb v aplikaci.
- Nasazení nové serverové části obalující původní rozhraní.
- Použití nového rozhraní v nové aplikaci.
- Testování a ladění chyb v celém systému.
- Rozšíření aplikace o nové funkce.
- Ukončení podpory původní aplikace.

Každý krok nasazování přitom začíná testováním na malé podmnožině používaných zařízení, aby případné chyby neměly příliš velký dopad. Po úspěšném otestování na této podmnožině se daná verze uvolní pro další zařízení.

2 Dostupná řešení

Existuje vícero společností, které se správou vozového parku zabývají. Prozkoumal jsem veřejně dostupné nabídky několika zahraničních a několika českých společností.

2.1 Zahraniční společnosti

Příkladem takové společnosti je Samsara[a]. Samsara poskytuje software, který splňuje veškeré funkční požadavky kromě propojení s informačním systémem, který přepravní společnost používá.

Tato společnost nemá veřejně dostupný ceník a při osobním kontaktu jsem se žádnou cenu nedozvěděl, ale na svém webu mají tabulky s výpočtem návratnosti investice[b], ze kterých je možné usoudit, jak se ceny služeb budou pohybovat.

Uvádí zde cenu 129 amerických dolarů za hardware. To je cena zhruba stejná jako cena základních Android zařízení prodávaných v České republice. k tomu je však nutné, aby řidiči měli vlastní chytrý telefon, aby mohli ovládat aplikaci k tomuto zařízení dodávanou. Tím se cena hardwaru dostává na dvojnásobek.

Významější část nákladů však tvoří cena licence, která činí 400 dolarů ročně. Když bude tato licence použita na desítky až stovky zařízení, bude tvořit pro společnost příliš vysoké náklady, mnohonásobně převyšující náklady na údržbu vlastní aplikace.

Kontaktováním společnosti Samsara jsem se navíc dozvěděl, že své služby v České republice neprovozují.

Podobná situace je í u dalších amerických společností, například USFleetTracking[c], Fleetistics[d], nebo Onfleet[e]. U všech těchto společností je příliš vysoká pravidelná platba za služby, u většiny í nedostatečná funkcionalita. Zda své služby tyto společnosti provozují í v České republice jsem nezjišťoval, posunul jsem se přímo na české společnosti.

2.2 České společnosti

V České republice podobné služby nabízí Česká společnost pro platební karty s.r.o. (CCS) se svým produktem CCS Carnet[f]. Zde jsou ceny mnohem přívětivější než u zahraničních

[a] https://www.samsara.com/
[b] https://www.samsara.com/roi-calculator
[c] https://www.usfleettracking.com/gps-tracking-service-pricing/
[d] https://www.fleetistics.com/
[e] https://onfleet.com/
[f] https://www.ccs.cz/carnet-pridana-hodnota

společností - 4 900 Kč za GPS jednotku a 300 Kč měsíčně za licenci na jedno zařízení. I tato cena však je pro společnost nevýhodná oproti vlastnímu řešení. Zde navíc není řešení, kterým by řidiči předávali informace dispečinku.

Česká aplikace Webdispečink[g] splňuje velkou část požadavků včetně API pro propojení s informačním systémem, avšak také nenabízí mobilní aplikaci pro volbu jízdních režimů a zadávání údajů, případně další komunikaci s řidičem.

Aplikace GPShlídač.cz[h] je cenově dostupnější (zařízení 2 299 Kč, 199 Kč měsíčně tarif) a obsahuje mobilní aplikaci umožňující přepínání řidičů ve vozidle a jízdních režimů - osobní/služební. Avšak tyto režimy nejsou dostačující, navíc systém nemá přístupné API pro propojení s vlastním IS.

Společnost Lokátory.cz[i] podobnou funkcionalitu také nabízí, také s vlastním zařízením k montáži do vozu. Zde je opět problém s absencí aplikace, o přístupném API se nezmiňují, a cena je zde 399 Kč za měsíc služby na vozidlo.

2.3 Shrnutí

České společnosti nabízí své produkty za nižší cenu, avšak s nedostatečnou funkcionalitou. I tato nižší cena však není výhodnější než vývoj a údržba vlastní aplikace. Nenašel jsem tedy žádný produkt, který by společnost mohla využít místo vlastní aplikace.

[g] https://www.webdispecink.cz/
[h] https://www.gpshlidac.cz
[i] https://lokatory.cz/

3 Návrh aplikace

3.1 Komunikační rozhraní

Pro komunikaci se serverem je v první fázi přechodu na nový systém nutné dodržet původní rozhraní. Pro další rozšiřování funkcionality však bylo toto rozhraní nevhodné, jelikož bylo tvořeno pouze jedním koncovým bodem pro všechny účely. Kvůli tomu je původní rozhraní nepřehledné, snadno nerozšiřitelné a neúsporné na objem přenášených dat.

Proto jsem definoval nové API, které bude aplikace využívat v pozdější fázi nasazování. Definici API a implementaci nového backendu jsem se podrobně věnoval ve svém magisterském projektu, zde uvedu pouze důležité části.

3.1.1 Struktura

Kompletní definici API přikládám v příloze A.

API jsem navrhl tak, aby odpovídalo principům REST a používalo protokol HTTP tak, jak jej Roy Fielding navrhl[5].

API zdaleka nevyužívá všech možností REST, ale drží se několika základních principů. Používá HTTP požadavky typu `POST` a `GET` a je bezstavové - veškeré informace o stavu si udržuje klient.

Inicializace

Po prvním spuštění se aplikace zaregistruje na serveru pomocí `POST` požadavku na koncový bod `device`. Tím získá své unikátní ID, kterým se bude při dalších požadavcích identifikovat. Toto ID bude součástí autentizační hlavičky, nikoliv těla požadavků.

Při každém spuštění aplikace pošle `GET` požadavek na koncový bod `drivers`, který poskytne seznam řidičů a jejich rolí.

Odesílání stavu

Zbývající dva koncové body slouží k odesílání aktuálního stavu vozidla a zařízení. Oba jsou typu `POST`.

Koncový bod `location` slouží k periodickému odesílání polohy a rychlosti vozidla, včetně informací o tom, z jakého zdroje a s jakou přesností informace pochází. Pokud v jeden okamžik přijde informace o poloze vozidla z GPS s přesností na jednotky metrů a o několik vteřin později přijde informace o poloze z GSM signálu s přesností na půl kilometru,

tak druhá informace nepřináší zpřesnění polohy. Na to server může reagovat zahozením méně přesné informace.

Koncový bod `status` je kontaktován při změně stavu vozidla nebo zařízení. Přijímá informace o jízdním režimu, řidiči, ujetých kilometrech, natankovaném palivu, naloženém zboží a stavu baterie.

Bylo by sice možné tyto dva koncové body sloučit, avšak vzhledem k velmi rozdílným četnostem použití těchto koncových bodů a jejich rozdílným účelům jsem se rozhodl je oddělit. Lokace bude odesílána dle potřeby během jízdy, zatímco stav bude odesílán pouze při interakci uživatele s aplikací, případně periodicky v delších časových intervalech, a to í mimo jízdu.

3.1.2 Zabezpečení

Samotný přenos dat mezi klientem a serverem je zabezpečený pomocí TLS na transportní vrstvě. Na této vrstvě tedy není potřeba další zabezpečení řešit.

Klíčovou částí zabezpečení je však autentizace klientů. Není totiž žádoucí, aby třetí strana předávala falešné informace serveru nebo aby server poskytoval data někomu jinému než mobilnímu klientovi, kterému jsou data určena.

Protokol HTTP je bezstavový, což zužuje výběr možností autentizace. Rozhodl jsem se využít autentizaci pomocí *bearer token*, který je využitý jako autentizační prvek v autorizačním frameworku OAuth 2.0[6].

Token se předává v HTTP hlavičce pod klíčem `Authorization`. k tomuto poli se síťové prvky chovají odpovídajícím způsobem, a to zejména tak, že toto pole nekešují.[9]

JSON Web Token

JSON Web Token[7] (dále jen JWT) je jednou z možností, co může obsahovat bearer token.

Tento standard definuje bezpečný způsob předávání informací mezi účastníky přenosu a je implementován ve všech hlavních webových frameworcích[10], což naznačuje, že je značně rozšířený.

JWT obsahuje následující informace:

1. Hlavičku, obsahující informace o tokenu samotném.
2. Tělo, nesoucí přenášenou informaci.
3. Podpis pro ověření, že informace jsou nezměněné a pochází z důvěryhodného zdroje.

Hlavička í tělo jsou textové řetězce ve formátu JSON.

JWT může být reprezentován JWS (JSON Web Signature) nebo JWE (JSON Web Encryption). JWE zašifruje celý obsah, zatímco JWS pouze umožní ověřit původ obsahu. Vzhledem k tomu, že obsahem přenášených zpráv jsou informace, které mají stejnou úroveň utajení jako nezašifrované tělo HTTP požadavku, není nutné tyto zprávy šifrovat. Proto jsem zvolil JWS.

Token tedy tvoří následující řetězec:

```
base64Url(hlavička) + "." +
base64Url(tělo) + "." +
šifrovací_funkce(
  base64Url(hlavička) + "." +
  base64Url(tělo),
  sůl|klíč
)
```

Smyslem použití JWT tedy je umístit do těla tokenu identifikátor konkrétního klienta, doplněný o nějakou v čase se měnící informaci, aby požadavek nemohl být opakován třetí stranou.

Token po ověření podpisu bude sloužit k identifikaci klienta a k předání záznamu o času vzniku informace. Tyto dvě položky dohromady vytvoří unikátní identifikátor odeslaných dat. Duplicity tedy mohou vzniknout pouze chybou v softwaru nebo zneužitím API. Zároveň nebude možné, aby při zneužití API bez znalosti šifrovacího klíče podpisu vznikla nová validní kombinace identifikátoru klienta a času. Aplikace tedy může duplicity ignorovat.

3.2 Uživatelské rozhraní

Základ uživatelského rozhraní by měl vycházet z původní aplikace, aby byl přechod mezi aplikacemi pro řidiče snadný. Zároveň však je žádoucí opravit některé nedostatky, které UI původní aplikace mělo. Výsledné uživatelské rozhraní je tedy kompromis mezi těmito dvěma aspekty.

Původní uživatelské rozhraní požadovalo při prvním spuštění prvotní konfiguraci (formou vyskakovacího okna), kde bylo zadáno ID zařízení. Toto ID bylo pro dané zařízení přiřazeno z databáze.

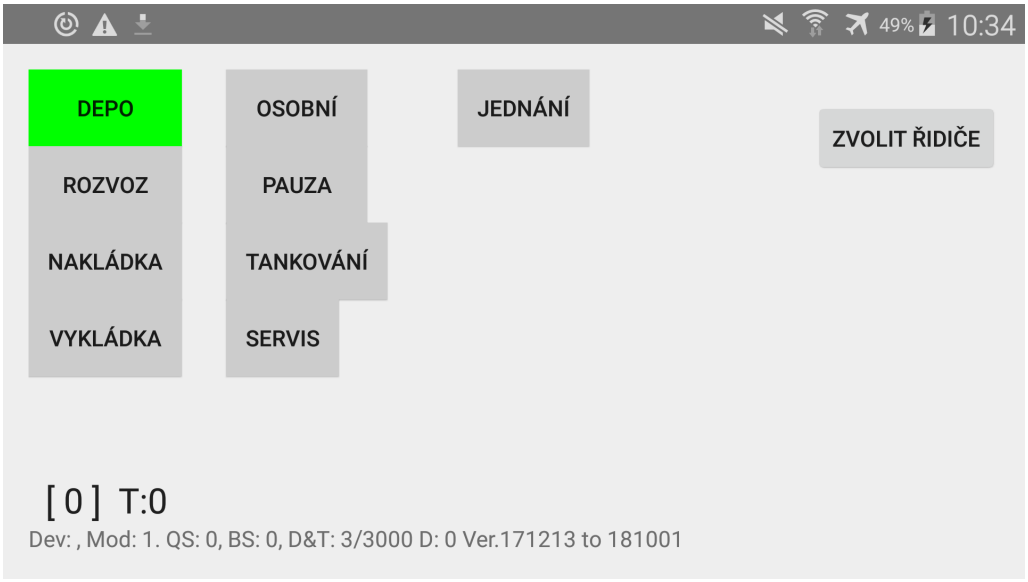
Po opuštění konfigurace byla zobrazena aktivita s výběrem jízdních režimů. Ještě než uživatel mohl zvolit jízdní režim, musel zvolit své jméno ze seznamu řidičů. Když tuto akci neprovedl, aplikace jej na tuto skutečnost upozornila. Avšak UI aplikace jej k tomuto kroku nenavedlo.

Po zvolení řidiče již uživatel mohl používat volbu jízdních režimů. Pokud zvolil jízdní režim, který požadoval dodatečné informace, zobrazilo se v aplikaci vyskakovací okno pro zadání těchto informací.

Tlačítka pro volbu jízdních režimů by mohla být větší, aby usnadnila uživateli používání, a mohla by graficky znázorňovat jejich činnost.

Uživatelské rozhraní původní aplikace vyžadovalo zobrazení na šířku, což nemusí vyhovovat každému uživateli.

Klíčová část uživatelského rozhraní, která musí být zachována podle původní aplikace, je stránka s volbou režimů. Tyto režimy musí zůstat stejné, se stejnými názvy a stejnou logikou ovládání.



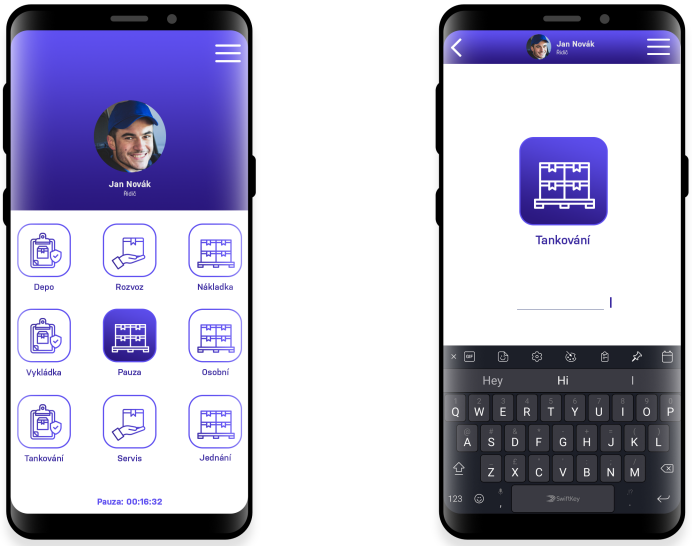
Obrázek 3.1: Snímek obrazovky původní aplikace

V nové aplikaci jsem se rozhodl vylepšit průběh po zapnutí aplikace. Stránka s volbou řidiče (respektive s přihlášením řidiče) se zobrazí zvlášť, ihned po zapnutí aplikace, a dokud řidič nebude přihlášen, nebudou žádné jiné prvky v aplikaci přístupné. Díky tomu by mělo být používání aplikace přímočařejší, což by mělo omezit zmatení nových uživatelů.

Jak bylo zmíněno, zobrazení na šířku nemusí vyhovovat všem uživatelům, proto by nové uživatelské rozhraní mělo podporovat varianty na šířku í na výšku, ale í různé velikosti a poměry stran displejů, neboť bude aplikace instalována na různých mobilních telefonech a na vestavěných 10palcových displejích.

Tlačítka byla navržena tak, aby pokrývala větší plochu, na kterou může uživatel kliknout, a graficky znázorňovala akci, kterou představují.

Při přepínání do režimů Depo a Tankování by měla aplikace požádat uživatele o zadání ujetých kilometrů (stav odometru). Toto zadávání by mělo být v budoucnu nahrazeno přímou komunikací zařízení s vozidlem. v režimu Tankování by měl být uživatel dotázán také na objem natankovaného paliva a v režimu Nakládka nebo Vykládka na naložené, respektive vyložené zboží.



Obrázek 3.2: Návrh uživatelského rozhraní

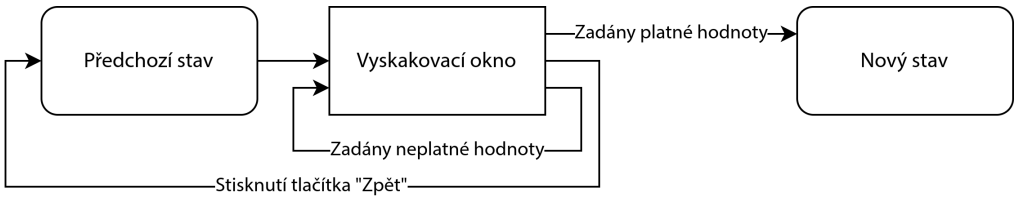
3.2.1 Ochrana proti uživatelským chybám

Pro zachování správnosti dat bylo potřeba také zabránit uživatelům ve tvoření chyb. Stanovil jsem proto omezení pro některé uživatelské akce.

Zvolení režimu Depo by měla aplikace umožnit pouze v případě, že se zařízení nachází fyzicky v depu. Pokud naopak řidič v depu jede vysokou rychlostí, nebo depo opustí, aplikace by jej měla donutit zvolit jiný jízdní režim. k tomuto jsem zvolil hlasová upozornění dávající řidiči instrukce. Hlasová upozornění mají výhodu v tom, že řidiče upozorní í navzdory vypnuté obrazovce. Také se řidič může plně věnovat řízení, ale zároveň je to dostatečně rušivý prvek na to, aby řidič toto upozornění neignoroval.

K tomu, aby aplikace mohla uživatele upozornit na opuštění depa bez zvolení jízdního režimu, je žádoucí, aby se aplikace spustila při startu zařízení.

V režimech, které vyžadují zadání dodatečných údajů (stav odometru, natankováno, ...), by mělo být uživatelům znemožněno nezadat požadované informace. Zároveň však je možné, že řidič zvolil daný jízdní režim omylem, proto by mělo být uživateli umožněno z režimu zároveň odejít od předchozího stavu. Navrhl jsem proto schéma, jak by se aplikace měla chovat v takových případech. Toto schéma je znázorněno na obrázku 3.3.



Obrázek 3.3: Návrh chování vyskakovacího okna

3.3 Přihlašování

Původní aplikace neměla žádný systém přihlašování. Server pouze poskytl seznam řidičů společnosti a aplikace nechala uživatele zvolit své jméno. Toto je potřeba s novou verzí aplikace zachovat, avšak s přechodem na nové API bude možné tuto zpětnou kompatibilitu přerušit a implementovat přihlašování pomocí jména a hesla.

3.4 Aktualizace

Běžné aplikace na Android zařízeních je možné instalovat a aktualizovat pomocí softwrových repozitářů, například F-Droid nebo Google Play Store. F-Droid však vyžaduje svobodný software a neumožňuje distribuovat neveřejné aplikace. Google Play Store umožňuje distribuovat neveřejné aplikace, ale vyžaduje, aby každý uživatel měl účet Google. To se jeví jako zbytečná komplikace, proto jsem se rozhodl vytvořit vlastní systém aktualizací. Aplikace se bude pravidelně dotazovat serveru zda je dostupná nová verze aplikace, upozorní na tuto skutečnost uživatele a umožní mu ji stáhnout.

4 Implementace aplikace

4.1 Technologie

Z počátku vývoje aplikace jsem počítal s tím, že bude aplikace používaná pouze na zařízeních s operačním systémem Android. Z tohoto pohledu bylo na první pohled jasnou volbou vyvinout aplikaci nativně na platformě Android.

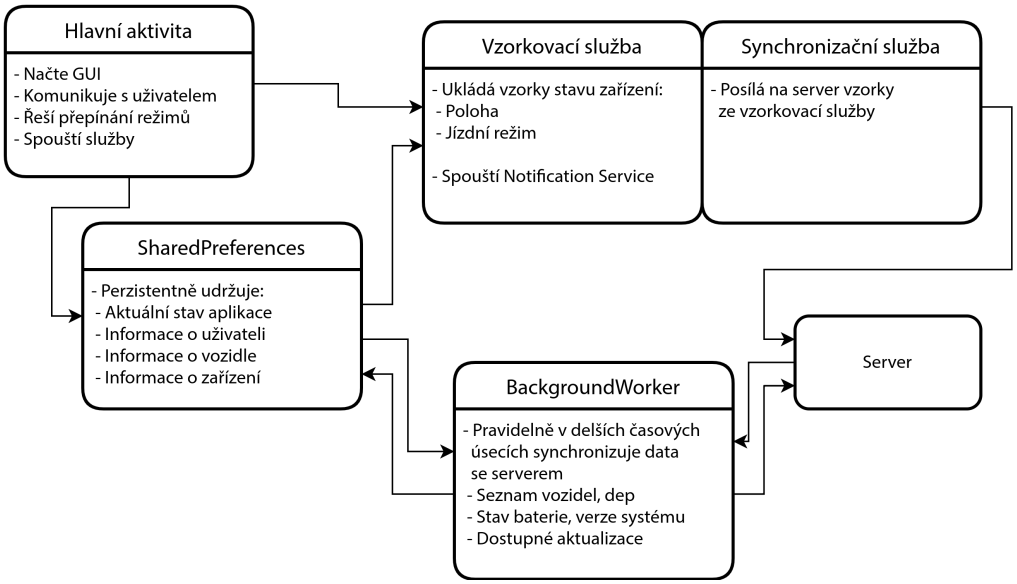
Pro vývoj na platformu Android se běžně používají jazyky Java a Kotlin.

Kotlin je jazyk modernější, novější (2011) a míněný jako „jednodušší, bezpečnější, se statickou kontrolou chyb jako dereference prázdných ukazatelů, a stručnější než Java“.[8] Tohoto mohli autoři jazyka dosáhnout díky tomu, že nemusí zachovávat zpětnou kompatibilitu s dlouhou historií vývoje jazyka na rozdíl od Javy. Avšak tento jazyk je založený na JVM, díky čemuž je použitelný na všech platformách podporovaných JVM.

Tyto výhody jazyka Kotlin však vyvažuje výhoda jazyka Java v tom, že tento jazyk používá 40% vývojářů, zatímco Kotlin pouhých 8%.[1] Vzhledem k tomu, že v budoucnu by se mohli do vývoje připojit další vývojáři, bylo by jednodušší, kdyby se nemuseli učit s novým, neznámým jazykem. Také by se použitím běžně používané technologie v budoucnu zvýšil tzv. bus factor - bylo by snazší pro nové vývojáře rozumět již napsanému kódu.[4]

Zejména z tohoto důvodu jsem tedy zvolil pro vývoj jazyk Java.

4.1.1 Návrh a vývoj v jazyce Java



Obrázek 4.1: Návrh aplikace pro platformu Android

Základ každé Android aplikace tvoří aktivita (aktivity tvoří prezenční vrstvu), na ni se mohou vázat další aktivity a služby[11]. Navrhl jsem uživatelské rozhraní obsahující pouze jednu aktivitu. Tato aktivita měla zajišťovat veškerou komunikaci s uživatelem a přepínání režimů a sloužila jako vstupní bod do aplikace.

Data, která uživatel zadal do aktivity, se ukládala do perzistentního úložiště, z něhož čerpaly služby. Aktivita také na základě volby jízdních režimů spouštěla snímkovací a synchronizační službu.

Snímkovací službu jsem navrhl tak, aby v pravidelných intervalech odebírala snímky polohy vozidla, jízdního režimu, rychlosti a naložení vozidla a poté tyto informace předávala synchronizační službě.

Synchronizační služba měla za úkol snímky ukládat do fronty a postupně je posílat na server. Pro optimalizaci využití datových přenosů synchronizační služba snímky měla seskupovat a posílat po dávkách.

Pro vyřešení přechodu mezi starým a novým backendem jsem vytvořil rozhraní, které implementovala tato synchronizační služba. Toto rozhraní měla implementovat í budoucí služba komunikující s novým backendem.

Ukládání dat

Pro ukládání vícerozměrných dat, jako například fronty snímků ze snímkovací služby, jsem zvolil použití databáze. Vestavěné Android knihovny nabízí implementaci databáze SQLite, která je pro toto použití ideální (zejména díky šetrnosti k paměti[11]).

Perzistentní uložení dat ve tvaru klíč-hodnota se v OS Android nazývá preference. Tyto preference se ukládají do objektů `Preferences` a `SharedPreferences`. `Preferences`

jsou preference pro danou aktivitu, kdežto `SharedPreferences` jsou sdílené pro celou aplikaci. Pro ukládání preferencí jsem tedy použil `SharedPreferences`.

Řešení omezení práce na pozadí

Operační systém Android omezuje aplikacím možnost pracovat na pozadí (tzn. při zhasnuté obrazovce či během používání jiné aplikace). Tato omezení jsou v systému postupně zaváděna a zpřísňována proto, aby umožnily uživateli kontrolovat spotřebu energie aplikacemi.

Pro aplikace, které potřebují vykonávat činnost na pozadí, existuje několik cest, jak toho dosáhnout. Tyto cesty se liší podle toho, jakým způsobem chce aplikace na pozadí pracovat[a].

Pokud aplikace potřebuje provést dlouhotrvající akci, spustí nové vlákno, v případě jazyka Kotlin spustí korutinu. Běh tohoto vlákna se však ukončí, jakmile uživatel aplikaci opustí. Proto pro akce, které trvají dlouho a musí být dokončeny (například synchronizace se serverem) doporučuje dokumentace Android použít API WorkManager.

Pro akce běžící na pozadí, kde přesně nezáleží na čase spuštění, také doporučují použít WorkManager. Toto API umožňuje spouštět úkoly jednorázově nebo opakovaně, avšak s rozestupem minimálně 15 minut a bez záruky času spuštění.[b]

Pro akce, kde na čase spuštění záleží, je nutné použít jiné technologie. Dříve bylo možné použít třídu AlarmManager, která umožňovala pravidelně spouštět vlákna, ale od API 19 již nezaručuje přesný čas spuštění[c].

Abych zamezil problémům s během aplikace na pozadí, vytvořil jsem notifikaci, kterou aplikace zobrazuje v době, kdy zaznamenává a odesílá data. Tato notifikace způsobí, že Android aplikaci neoznačí za aplikaci na pozadí, ale nechá jí stejná oprávnění, jako když běží na popředí[d].

Tuto notifikaci zobrazuje a skrývá vzorkovací služba, pro jejíž fungování je běh na pozadí zásadní. Na tuto službu je vázaná i synchronizační služba, tudíž obě služby mohou fungovat na pozadí stejně spolehlivě jako na popředí.

Pro pravidelnou synchronizaci dat se serverem, kde tolik nezáleží na čase spuštění a není potřeba tak vysoká frekvence spouštění, měl být použit BackgroundWorker. Tato třída měla využívat výše zmíněné API WorkManager, který pro komunikaci se serverem nalezne vhodnou chvíli tak, aby maximálně šetřil zdroje zařízení.

4.1.2 Mobilní framework

Ve chvíli, kdy jsem měl dokončený návrh struktury aplikace a implementoval jsem základy funkcionality v jazyce Java, ukázalo se, že bude potřeba provést zásadní změnu ve vývoji. Někteří zaměstnanci společnosti totiž chtěli aplikaci používat na svých zařízeních s operačním systémem iOS.

[a] `https://developer.android.com/guide/background/`
[b] `https://developer.android.com/reference/androidx/work/PeriodicWorkRequest`
[c] `https://developer.android.com/reference/android/app/AlarmManager`
[d] `https://developer.android.com/guide/components/foreground-services`

V tomto případě by bylo nutné přepsat aplikaci také do jazyka Swift, který nahradil Objective-C v roli hlavního jazyka pro vývoj na iOS.[12] To by však znamenalo udržovat dva oddělené zdrojové kódy k jedné aplikaci v různých jazycích a to by nebylo časově ani finančně efektivní. Jedinou výhodu, kterou by to přineslo, by byla možnost optimalizace pro každou platformu zvlášť. Tato aplikace však není na výkon zařízení nijak náročná, proto je výhodnější hledat jiné řešení než paralelní vývoj dvou aplikací.

Hledal jsem proto možnosti, jak by mohly aplikace pro iOS a Android sdílet co největší část kódu. Jednou ze zkoumaných možností bylo použití balíku `golang/mobile`. Tento balík umožňuje psát část kódu nebo celý zdrojový kód aplikace v jazyku Go. Avšak tato knihovna je v experimentálním stádiu a není aktivně udržovaná[e], takže pro praktické využití není vhodná.

Na podobném principu funguje sada nástrojů od společnosti Xamarin (vlastněné společností Microsoft). Aplikace pro různé platformy sdílí kód pro logiku aplikace v jazyce C# na platformě .NET a využívají (na rozdíl od `golang/mobile`) sdílenou knihovnu Xamarin.Forms pro vývoj uživatelského rozhraní. Díky tomu není potřeba velké množství platformě závislého kódu. Sada Xamarin má také mnoho uživatelů[1], což je do jisté míry zárukou kvality a dostupné komunitní dokumentace. Nedostatkem nástrojů Xamarin je však jejich závislost na vývojářském prostředí Microsoft Visual Studio a chybějící oficiální podpoře pro vývoj v operačním systému Linux[f].

Společnost Google vyvíjí také framework Flutter. Tento framework nabízí také možnost multiplatformního vývoje se společným zdrojovým kódem. Jeho nedostatkem je použití jazyka Dart, který není příliš používaný[1]. Z toho plynou potíže s nedostatkem komunitní dokumentace a vývojářů pro budoucí vývoj.

Jako nejlepší se nakonec ukázalo použití mobilních frameworků, které využívají webové technologie. Jejich velkou výhodou je, že jsou srozumitelné pro webové vývojáře, kterých je mezi programátory většina[1]. Tyto frameworky používají kombinaci webových technologií pro uživatelské rozhraní a platformně závislého kódu pro komunikaci s rozhraním operačního systému.

Volba frameworku

Z webových frameworků jsem vybíral mezi třemi frameworky.

Framework React Native vyvíjený společností Facebook je plně programovaný v jazyce JavaScript a komunikuje se systémem asynchronně pomocí tzv. mostu[13].

NativeScript společnosti Progress nabízí podobnou funkcionalitu jako React Native s rozdílem v tom, že od základu používá jazyk TypeScript a má menší komunitu.

Pro tento projekt jsem však vybral framework Ionic společnosti Drifty, který má od frameworků NativeScript a React Native odlišný princip fungování. Zatímco předchozí zmíněné frameworky se překládají podle cílové platformy, Ionic aplikace zůstávají webovými aplikacemi, které jsou zobrazovány v systémovém webovém prohlížeči. k systémovým prostředkům pak Ionic aplikace přistupují pomocí rozšíření Apache Cordova nebo Capacitor[3].

[e] https://github.com/golang/mobile

[f] https://developercommunity.visualstudio.com/idea/390674/xamarin-development-on-linux.html

Výhodou frameworku Ionic je velké množství hotových knihoven pro komunikaci se systémovými rozhraními a hotové komponenty uživatelského rozhraní dodržující vzhled dané platformy.

Důležité bylo také programování v jazyku TypeScript, který přidává bezpečnost kódu kontrolou datových typů při transpilaci. Výhodou je také z mého pohledu lepší čitelnost kódu díky typovým anotacím.

Backend framework

Jak bylo zmíněno, Ionic aplikace mohou používat pro komunikaci se systémem dva různé frameworky. Původně byl Ionic vyvíjen pro spolupráci se zavedeným frameworkem Apache Cordova (kompatibilní také s proprietárním Adobe PhoneGap), který má dlouho zavedenou početnou komunitu uživatelů a velké množství komunitních doplňků.

V současné době vývojáři Ionicu postupně přecházejí na framework Capacitor, který sami vyvíjejí[3]. v době, kdy jsem s vývojem aplikace začínal (podzim 2019), však tato knihovna ještě nebyla schopna pokrýt všechnu požadovanou funkcionalitu a byla teprve v testovací fázi[g], což mne vedlo ke zvolení knihovny Cordova.

Frontend framework

Pro programování uživatelského rozhraní na platformě Ionic je možné použít většinu JavaScriptových frameworků (například Angular, React, Vue) nebo použít čistý JavaScript.

Zvolil jsem vývoj ve frameworku Angular, neboť je ve frameworku Ionic nejpoužívanější (Ionic byl původně úzce propojen s Angular[h]). Díky tomu existuje pro Ionic s Angular nejvíce dokumentace (oficiální í komunitní).

4.1.3 Porovnání nástrojů

Díky použití dvou různých sad vývojářských nástrojů mohu subjektivně porovnat pocit z vývoje pomocí těchto nástrojů.

Nevýhodou potřeby Android Studia pro vývoj je hardwarová náročnost tohoto nástroje. Načtení projektu v textovém editoru Sublime Text trvá vždy méně než jednu vteřinu, zatímco v Android Studiu při opakovaných testech na moderním počítači nejméně deset vteřin. Při prvním spuštění í více jak minutu. Pouhý běh Android Studia na pozadí zabere v počítači zhruba 2 GB operační paměti a znatelně zatíží procesor počítače. Použití virtuálního stroje a sestavení aplikace však používá stejné nástroje (Gradle, AVD) jako při použití mobilního frameworku, tudíž trvá zhruba stejnou dobu.

Značkovací jazyk pro popis vzhledu aplikace není uzpůsobený pro to, aby byl psaný ručně - zápis je příliš výřečný. Proto obsahuje Android Studio grafický nástroj na návrh vzhledu aplikace, který tento kód generuje.

Pro vývoj aplikací ve frameworku Ionic dřívě existoval nástroj Ionic Studio, který již na webových stránkách Ionic frameworku nabízen není. Avšak Ionic aplikace jsou vyví-

[g] Dnes již toto tvrzení neplatí - během léta 2020 se mi Capacitor osvědčil jako dobrá alternativa Apache Cordova.

[h] https://ionicframework.com/docs/v1/overview/

jeny pomocí webových technologií, díky čemuž je možné použít pro vývoj libovolný jiný nástroj. Vývoj uživatelského rozhraní pomocí JavaScriptu, HTML, CSS a frameworků Ionic a Angular umožňuje vývoj bez grafických nástrojů, čistě v textovém editoru. Úpravy mohou být testovány a laděny ve webovém prohlížeči, což umožňuje krátkou zpětnovazební smyčku pro vývoj uživatelského rozhraní. Použití běžných webových technologií umožňuje získání velkého množství podpory na Internetu, a to mimo oficiální dokumentace zejména od velmi velké komunity webových vývojářů. Tvorbu uživatelského rozhraní usnadňují Ionic komponenty, které dodržují vzhled jednotlivých platforem (pro Android je to Material Design), což je obdobné jako u Android Studia.

Ladící nástroje jsou v Android Studiu kvalitní a dobře použitelné. Jsou zde k dispozici standardní nástroje IntelliJ a místo konzole pro výpis slouží nástroj Logcat.

S frameworkem Ionic se aplikace dají ladit pomocí Chromium developer tools, které se připojí k instanci Android System WebView na telefonu připojeném přes ADB. Zde je možné na rozdíl od Android Studia provádět i experimentální úpravy vzhledu aplikace. Nevýhodu však tvoří transpilace kódu, kvůli čemuž kód zobrazený ve vývojářských nástrojích neodpovídá přesně původnímu zrojovému kódu. k vyhnutí se tomuto problému slouží takzvané „source mapy“, ale ani ty se během vývoje nedařilo zprovoznit tak, aby přesně mapovaly kód.

Pro vývoj na platformě Ionic bylo také důležité použití knihoven třetích stran. Zatímco Android má velkou část funkcionalit, jako například HTTP komunikaci se serverem či přístup k GPS na pozadí, ve vlastních knihovnách, pro použití těchto funkcionalit v Ionic frameworku bylo potřeba použít externí knihovny. Tyto knihovny často pouze obalují vestavěné Android knihovny a je možné je získat z balíčkovacího systému NPM. Žádná z vestavěných funkcionalit při vývoji díky tomu nechyběla.

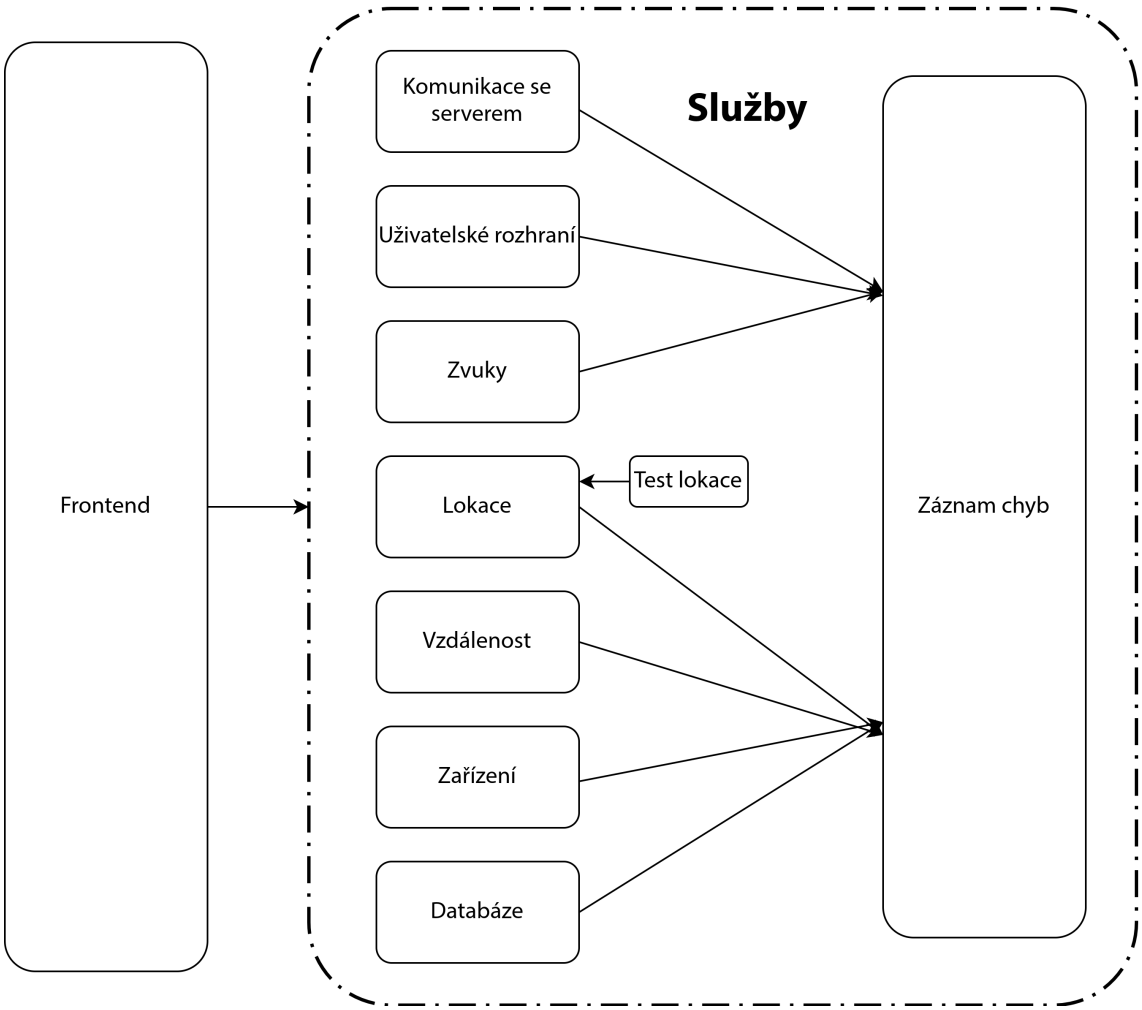
4.2 Implementace

Po změně technologií jsem vytvořil nový návrh aplikace. Liší se tak nejen implementace, ale i struktura aplikace, neboť jiné technologie vedou na jiná řešení.

4.2.1 Návrh

Vstupním bodem řídícím celou aplikaci je, stejně jako v původním návrhu implementace, stránka s uživatelským rozhraním. v průběhu vývoje jsem se pokusil tuto stránku rozdělit na jednotlivé komponenty, avšak to vedlo k příliš velké složitosti kódu, proto jsem od této cesty upustil.

Zbytek logiky aplikace jsem se rozhodl rozdělit do jednoúčelových služeb, aby byl kód přehledný.



Obrázek 4.2: Schéma aplikace

4.2.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní bylo tvořeno pomocí Ionic komponent tak, aby dodržovalo grafický styl dané platformy. Android aplikace tedy dodržuje systémový Material Design.

Stránka s voličem jízdních režimů je vytvořená pomocí podmíněně podbarvovaných prvků. Původně ikony jízdních režimů byly implementovány pomocí přepínače (`input type="radio"`), neboť režimy se vzájemně vylučují. Tento přístup jsem však později změnil na samostatné prvky, aby bylo možné zamezit přepnutí do neplatného režimu.

Tato stránka má také vlastní kaskádové styly, které podbarvují tlačítka zvoleného režimu a dělají tlačítka dostatečně velká a přehledná.

Obrázek 4.3: Grafické uživatelské rozhraní voliče režimů

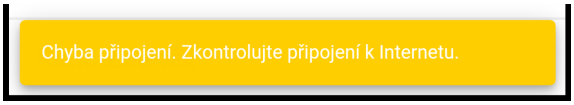
4.2.3 Komunikace s uživatelem

Vizuální upozornění

Pro upozornění uživatele na nějakou skutečnost, která přímo nesouvisí s používáním aplikace za jízdy, používá aplikace vizuální upozornění.

Pokud se jedná o upozornění vyžadující navedení uživatele na nějakou akci, je pro tento účel použito vyskakovací okno s dodatečnými informacemi. Toto se týká například chybějících oprávnění aplikace, kde je uživatel naveden do příslušné části nastavení telefonu. Konkrétně upozornění na chybějící oprávnění je zároveň blokující, bez něj uživatel nemůže aplikaci vůbec používat.

Pro upozornění na chybějící připojení k Internetu jsem použil `toast` - malé časově omezené textové upozornění. Toto upozornění je schopno se zobrazit nad ostatní okna í nad jiné aplikace v systému. Chybějící připojení k Internetu může být chybou řidiče, ale také se může stávat náhodně během jízdy. Na to je aplikace připravená, a proto není potřeba po řidiči vyžadovat nějakou akci.



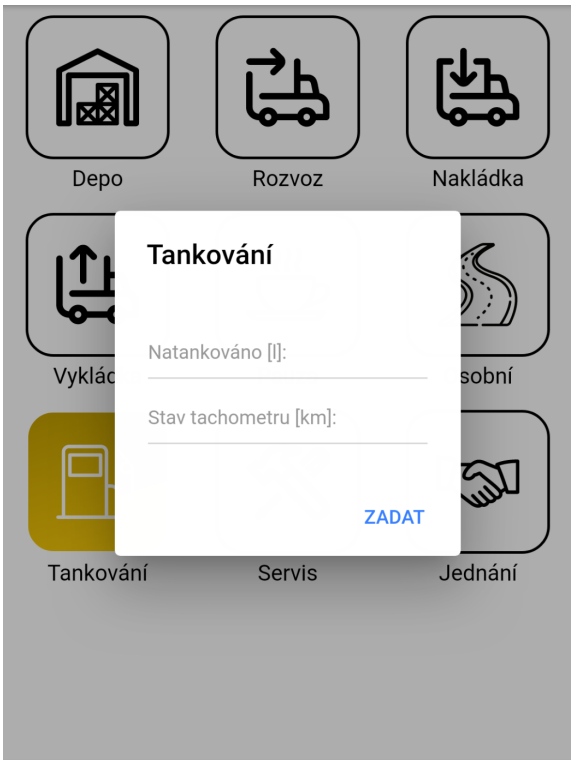
Obrázek 4.4: Upozornění na chybějící internetové připojení

Hlasová upozornění

V případě, že aplikace zjistí, že uživatel neprovedl akci, kterou měl, upozorní jej hlasově. Jedná se například o situace, kdy se uživatel v jízdním režimu „Pauza“ nebo „Depo“ rozjede nad povolenou rychlost nebo opustí depo. Pokud se uživatel zapomene přihlásit, a vůbec neprovede žádnou interakci s aplikací, aplikace jej také upozorní.

Celkem aplikace obsahuje deset různých namluvených zvuků, které se přehrávají podle různých kontrol při běhu aplikace.

Vyskakovací formuláře



Obrázek 4.5: Příklad vyskakovacího formuláře

Pokud po uživateli aplikace požaduje dodatečné informace, zobrazí vyskakovací okno. Toto okno obsahuje formulář s poli, které má uživatel vyplnit. Formulář má ověření správnosti zadaných údajů, například na základě heuristiky ujeté vzdálenosti. Vyskakovací okna se chovají podle schématu na obrázku 3.3.

4.2.4 Služby

Komunikaci aplikace s vnějším prostředím jsem pro přehlednost rozdělil do jednotlivých služeb.

Komunikace s uživatelem

Služba komunikující s uživatelem vyvolává vyskakovací okna umožňující požádat uživatele o potvrzení nějaké informace nebo vytvoření formuláře pro uživatele. Tento formulář se vytvoří podle zadaného pole požadovaných informací. Implemenotovaná jsou pouze číselná pole, protože aplikace jiný druh informací od uživatele v tuto chvíli nepotřebuje.

Služba také umožňuje upozornit uživatele s pomocí Android toast. Tato služba toast abstrahuje tak, že jej vystavuje jako „upozornění“ s nastavitelným textem a úrovní důležitosti. Tato úroveň je reprezentována barvou pozadí toastu.

Přehrávání zvuku

Přehrávání zvuku je v aplikaci zajištěno pomocí HTML5 audio.

Kdyby uživatel snížil hlasitost na minimum, neslyšel by hlasová upozornění. Proto jsem implementoval s pomocí Ionic knihovny `AudioManagement` u důležitých upozornění zvýšení hlasitosti. Díky tomu nebude možné, aby řidič upozornění aplikace ignoroval.

Také bylo potřeba zamezit překrývání zvuků. v případě, že se již stejný zvuk přehrává, znovu se přehrávat nezačne. Pokud se přehrává jiný zvuk, zastaví se a začne se přehrávat nový zvuk.

Ukládání dat

Pro ukládání dat v aplikaci používám dva odlišné přístupy.

Data typu klíč-hodnota jsou ukládána s pomocí knihovny `ionic-storage`. Tato knihovna využívá nativní rozhraní dané platformy pro perzistentní ukládání dat.

Dvourozměrná data aplikace ukládá do databáze SQLite 3.

Informace ze zařízení

Služba na získávání informací ze zařízení abstrahuje přístup k identifikátoru zařízení, stavu baterie a nabíjení. Identifikátor zařízení je zaveden při prvním spuštění aplikace. k informacím o baterii aplikace přistupuje s pomocí příslušné Ionic knihovny.

Snímání lokace

Snímání lokace jsem implementoval s využitím knihovny `mauron85/background-geolocation`. Tato knihovna nahrazuje systémové lokační služby. Během vývoje jsem narazil na jedinou funkcionalitu, kterou `background-geolocation` na rozdíl od systémových služeb neumožňuje, a to výpočet ujeté vzdálenosti.

Lokace je snímána v závislosti na režimu. v některých režimech tato služba (pojmenovaná `Tracker`) pouze hlídá, zda se uživatel nevzdálil z depa či nepřekročil povolenou rychlost. v režimu soukromé jízdy služba počítá ujetou vzdálenost.

V režimu Rozvoz zaznamenává `Tracker` aktuální polohu vozidla. S každou změnou polohy vozidla o více než 250 metrů je aktuální stav spolu s jízdním režimem a dalšími informacemi zaznamenán do snímku a předán do služby `Broker`, která zajistí odeslání na server.

Výpočet vzdálenosti

Pro ověřování uživatelem zadané hodnoty stavu odometru je potřeba sledovat ujetou vzdálenost od posledního záznamu. Na rozdíl od lokačních služeb Androidu neumožňuje použitá lokační knihovna počítat ujetou vzdálenost. Proto jsem použil výpočet pomocí haversinové rovnice. Bylo sice nutné model povrchu Země zjednodušit na tvar koule, avšak toto zjednodušení nemá velký vliv na použitelnost výsledků. Větší vliv na použitelnost má to, že je vzdálenost vypočítávána mezi po sobě jdoucími snímky lokace. Proto je celková přesnost vypočtené vzdálenosti závislá hlavně na přesnosti určení a frekvenci snímání lokace.

Odhady ujeté vzdálenosti tedy nejsou dostatečně přesné pro to, aby nahradily odometr vozidla, ale jsou dostatečně přesné na to, aby ověřily správnost zadaných údajů.

Komunikace se serverem

Komunikaci se serverem zajišťuje služba pojmenovaná `Broker`. `Broker` přijímá snímky stavů od služby `Tracker` a zajišťuje jejich přenos. Příchozí snímky ukládá do databáze, která slouží jako vyrovnávací paměť. `Broker` se snímky v databázi zachází jako s prvky ve frontě. v pravidelných intervalech se snímky z databáze načtou, spojí do jednoho požadavku (z důvodu šetření energií) a odešlou se na server.

`Broker` také zajišťuje ostatní komunikaci. Pro komunikaci s původním API se jedná o načtení seznamu řidičů. Pro nové API jde o periodické odesílání stavu zařízení, přihlašování a aktualizaci seznamu dep.

Tato služba je navržená tak, aby abstrahovala veškerou komunikaci. Díky tomu může být jedinou částí aplikace, jejíž implementaci je třeba změnit při změně API serverové části aplikace.

Důležitou vlastností služby `Broker` je odolnost vůči nedostupnosti internetového připojení. v případě výpadku připojení služba dále ukládá stavy do fronty a opakuje pokusy o odeslání dat.

Načtení seznamu řidičů probíhá také asynchronně. Při otevření aplikace je uživateli zobrazen naposledy stažený seznam uložený v databázi. Ve většině případů není potřeba čekat na aktualizování seznamu ze serveru. v případě prvního spuštění aplikace nebo změny v seznamu se změny projeví po stažení aktuálních dat.

Odchytávání chyb

V budoucnu je plánované odesílání kritických chyb na server. Během vývoje je užitečné naopak vypisovat chyby do konzole. Proto jsem vytvořil službu, která přepisuje výchozí chování frameworku Angular v odchytávání chyb. Tato služba implementuje rozhraní `NgErrorHandler`, a díky tomu jí projdou všechny chyby, které v aplikaci vzniknou. Služba poté umožní vlastní implementaci řešení neošetřených výjimek.

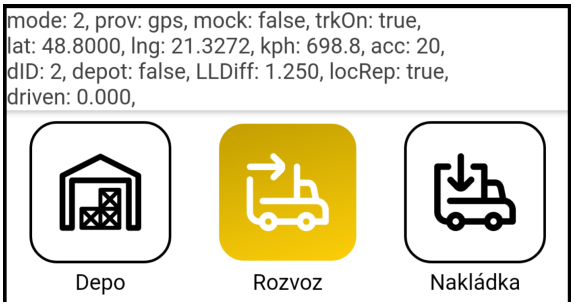
4.2.5 Testování

Při implementaci funkce na výpočet vzdálenosti mezi dvěma body na Zemi jsem potřeboval najít dostatečně dobrý výpočet. Abych mohl experimentovat s různými implementacemi, vytvořil jsem testovací sadu pomocí nástroje Jest. Díky tomu jsem mohl postupovat podle principů programování řízeného testy (TDD).

TDD je založeno na postupu, kde programátor napíše nejprve test funkcionality, kterou chce implementovat, a poté pracuje na implementaci této funkcionality, dokud testy neprochází[2].

Díky tomu bylo snazší implementovat funkci pro výpočet vzdálenosti. v ostatních částech aplikace jsem principy TDD nepoužil, neboť kód byl buď velmi jednoduchý, nebo obtížně testovatelný a psaní testů by zpomalilo vývoj.

4.2.6 Pomůcky pro hledání chyb



Obrázek 4.6: Testovací výstup

Pro zjednodušení testování aplikace v terénu jsem do aplikace přidal testovací výstup. Tento výstup zobrazuje informace o aplikaci v reálném čase přímo na displeji zařízení.

Je žádoucí, aby testovací výstup byl přístupný pro testování a pro ladění chyb u nasazených aplikací, ale aby nebyl přístupný běžným uživatelům. Proto je výstup v aplikaci skrytý a zobrazí se pouze po několikanásobném kliknutí na horní panel aplikace.

5 Zkušenosti z provozu

Aplikace byla vydána k prvnímu testování, které poskytlo podněty k ladění chyb a zkušenosti s provozem aplikace.

5.1 Technické zkušenosti

Při instalaci aplikace na různá zařízení jsem se setkal s problémy způsobenými softwarem výrobců telefonů. Systém Android má zdokumentovaná omezení pro aplikace, í způsoby, jakými je obcházet. Výrobci telefonů však k operačnímu systému přidávají svoje vlastní nadstavby, které tvoří další omezení. Tato omezení však nejsou tak dobře zdokumentována, a často není možné je řešit aplikačně.

Příkladem takového výrobce je společnost Xiaomi. Ta do svých telefonů instaluje nadstavbu MIUI, která zamezuje aplikacím běžet na pozadí í za podmínek, za kterých to OS Android dovoluje. Pro obejití tohoto omezení je nutné, aby uživatel měnil pokročilá nastavení telefonu. Proto musel být sepsán návod pro uživatele s různými verzemi operačního systému a nadstavby MIUI. Podobně bylo potřeba postupovat í u telefonů značky Huawei.

Pro běh aplikace je také klíčové používání polohových služeb. Aplikace je schopná detekovat zapnutí/vypnutí polohových služeb OS Android a uživatele na tuto skutečnost upozornit. Avšak na zařízeních s nadstavbou MIUI má tato funkcionalita vlastní implementaci, se kterou není aplikace schopna poznat, zda má přístup k poloze zařízení. Do budoucna bude potřeba tento stav detekovat, ať už pomocí API jednotlivých výrobců, či pomocí vlastní heuristiky.

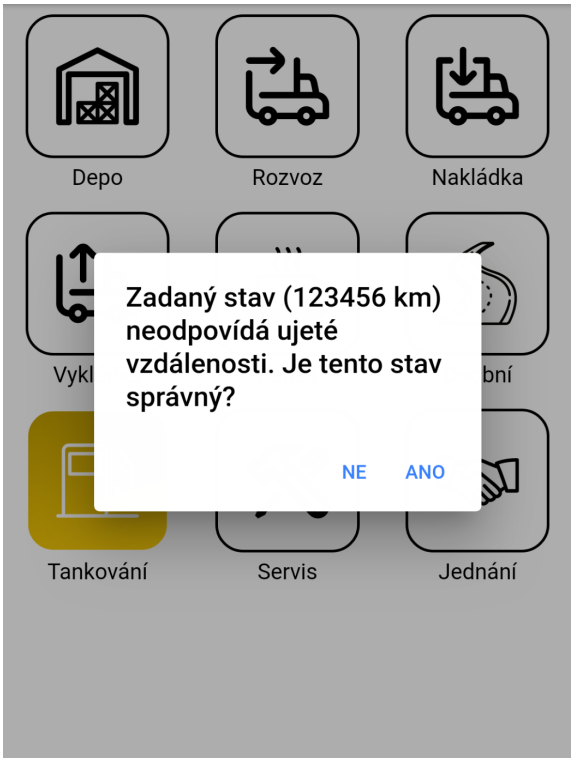
5.2 Uživatelské zkušenosti

V průběhu testování se stávalo, že uživatelé dělali chyby. Proto bylo potřeba do aplikace přidat ověření na ty nejčastější.

V případech, kdy uživatel nepovolil aplikaci přístup k poloze zařízení, aplikace nezískávala potřebné informace. Do aplikace jsem tedy přidal ověření přístupu k polohovým informacím. Uživatelé jsou nyní na vypnuté polohové služby upozorněni a přesměrováni do nastavení telefonu, kde mohou toto nastavení změnit.

Uživatelé také někdy zadali nereálné hodnoty do vyskakovacích oken. Na objem natankovaného paliva je možné použít ověření vycházející z předpokladu, že řidič nenatankoval záporné množství paliva nebo větší množství paliva, než je objem nádrže. Podobně je tomu u množství naloženého/vyloženého nákladu. Pro stav odometru jsem vytvořil heuristiku,

která zaznamenává ujetou vzdálenost od posledního zadaného stavu a porovnává s tímto stavem zadanou hodnotu, zda se neliší o více jak 20 %. Pokud se liší, je uživatel na chybu upozorněn a je mu nabídnuta náprava.



Obrázek 5.1: Upozornění na chybně zadané údaje

6 Další vývoj

Aplikaci jsem dovedl do fáze testování a ladění chyb v aplikaci. Dále bude potřeba pokračovat v nasazování nového systému jako celku podle postupu popsaného v kapitole 1.3.2.

Nyní bude pokračovat nasazování aplikace na další zařízení. Až bude ověřena stabilita a správná funkčnost aplikace, bude se nasazovat nová serverová část a bude vydána nová verze aplikace používající nové API. Po odladění chyb a ustálení provozu celého nového systému bude ukončen provoz původního backendu. Tento celý proces je naplánován v řádu jednotek let, proto nebyl cílem této práce.

V blízké době bude také testována funkčnost aplikace na zařízeních se sedmipalcovým displejem vestavěných do palubní desky vozidel. Toto zařízení mi dodavatel prodal s tvrzením, že se jedná o zařízení s OS Android verze 7, avšak ve skutečnosti se jednalo o verzi 4.4, kterou tato aplikace nepodporuje. Situaci s prodejcem řeším a čekám na nové zařízení.

Aplikace je také připravena na podporu operačního systému iOS. Tento systém jsem zatím netestoval, ale podle dokumentace by žádná z použitých knihoven neměla mít s podporou iOS problém. Podmínkou kompilace pro tato zařízení je pouze operační systém macOS a prostředí XCode.

Samotná aplikace bude rozšířena o systém aktualizací podle návrhu v sekci 3.4.

S novým API bude možné začít aplikaci rozšiřovat o další funkcionality. Jedná se zejména o přímou komunikaci s vozidlem. Některé vestavěné displeje umí komunikovat s vozidlem přes protokol CAN, avšak nejsou dle mých rešerší schopny získat informace o stavu vozidla. Slouží pouze k využití ovádacích prvků vozidla k ovládání zařízení. Pro komunikaci s vozidlem tedy bude pravděpodobně nejvhodnější využít Bluetooth adaptér OBD2.

Komunikace s vozidlem přes OBD2 umožní získávat v reálném čase informace o stavu provozních kapalin, stavu odometru a o diagnostických informacích. Tyto informace se budou předávat informačnímu systému a dispečeři se díky tomu včas dozvědí o problémech s vozidly.

Při přímé komunikaci s vozidlem bude také vidět, kdy vůz stojí na místě s nastartovaným motorem příliš dlouhou dobu, což umožní dispečerům motivovat řidiče k minimalizaci spotřeby paliva.

Také informace o rychlosti vozidla umožní lepší kontrolu jízdních režimů. Aplikace s touto informací nebude muset za každých okolností využívat polohových služeb k ověření, zda uživatel nezapomněl zvolit jízdní režim.

Dále je v plánu aplikaci rozšířit o rozvozový plán, který dispečink bude moci posílat řidiči přímo do aplikace a během jízdy jej aktualizovat. Aplikace poté může ověřovat plnění rozvozového plánu.

Závěr

V této práci jsem stanovil požadavky na systém umožňující dispečinku dopravní společnosti sledovat pohyb vozidel, který měl nahradit nedostačující stávající systém.

Provedl jsem průzkum dostupných řešení na trhu, z nichž žádné nebylo vhodné pro nahrazení stávajícího systému. Proto jsem navrhl a vytvořil novou mobilní aplikaci sledující pohyb vozidel a sbírající informace o jízdě.

Při návrhu aplikace jsem navrhl uživatelské rozhraní tak, aby bylo pro užvatele snadno ovladatelné a předcházelo chybám. Komunikační rozhraní jsem navrhl společně s postupem nasazování tak, aby byl zajištěn plynulý přechod mezi systémy bez výpadků v provozu.

Aplikaci jsem implementoval v multiplatformním mobilním frameworku Ionic. Při implementaci jsem se soustředil na kvalitní návrh, který umožní rozšiřování aplikace v budoucnu.

Aplikace komunikuje s řidičem vizuálně í hlasově, a to tak, aby v maximální možné míře předešla vzniku chyby. Komunikace se serverem je implementována robustně, aby aplikace bez problémů fungovala í s nestabilním připojením k Internetu.

Aplikaci jsem vydal k testování a na základě zpětné vazby jsem opravil chyby, upravil funkcionalitu a přidal další ochranu před uživatelskými chybami.

Další testování stále probíhá. Postupně budu nahrazovat tímto systémem systém stávající.

Díky tomuto novému systému bude možné aplikaci dále pro řidiče zjednodušit a zvýšit množství užitečných dat přenášených do informačního systému. Také bude možné rozšířit aplikaci o automatizované rozvozové plány.

Literatura

[1] 2020 Developer survey. https://insights.stackoverflow.com/survey/2020, 2020. cit. 2020-12-01.

[2] Beck, K. *Test-driven Development: By Example*. Addison-Wesley, 2003. ISBN 9780321146533.

[3] Capacitor. *Building cross-platform apps with Capacitor*. Drifty Co. cit. 2020-12-12.

[4] Coplien, J. – Schmidt, D. *Pattern Languages of Program Design*. Addison Wesley, 1995. Chapter 13, A Generative Development-Process Pattern Language.

[5] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. Dostupné z: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[6] Jones, M. – Hardt, D. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, RFC Editor, October 2012. Dostupné z: <http://www.rfc-editor.org/rfc/rfc6750.txt>.

[7] Jones, M. – Bradley, J. – Sakimura, N. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015. Dostupné z: <http://www.rfc-editor.org/rfc/rfc7519.txt>.

[8] Krill, P. JetBrains readies JVM-based language. https://web.archive.org/web/20201110150026/https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html, 2011. cit. 2020-12-01.

[9] Lodderstedt, T. – McGloin, M. – Hunt, P. OAuth 2.0 Threat Model and Security Considerations. RFC 6819, RFC Editor, January 2013. Dostupné z: <http://www.rfc-editor.org/rfc/rfc6819.txt>.

[10] Peyrott, S. E. *The JWT Handbook*. Autho Inc., 2016-2018. verze 0.14.1.

[11] Ujbányai, M. *Programujeme pro Android*. Grada, 2012.

[12] Wang, W. *Pro iPhone Development with Swift 5*. Apress, 2019.

[13] Zagallo, T. Bridging in React Native, Oct 2015. Dostupné z: <https://tadeuzagallo.com/blog/react-native-bridge/>.

A Specitikace API

```
headers:
    Authentication: JWT (
        header: {
            "alg": "HS256",
            "typ": "JWT",
        }
        payload: {
            "sub": int, // subject
            "toe": int, // time of event
        }
    )

status codes:
    "SUCCESS"
    "UNKNOWN_ERROR"
    "BAD_REQUEST"

POST /device:
    req: none

    res: {
        "status": string,
        "message": string,
        "result": int,
    }

GET /drivers:
    req: none

    res: {
        "status": string,
        "message": string,
        "result": [
            {
                "id": int,
                "name": string,
                "role": string, // jeden z~["sales", "driver"]
            }
        ]
    }

POST /location:
    req: {
```

```
        "lat": int,
        "lng": int,
        "accuracy": int,
        "speed": int,
        "provider": string,
    }

    res: {
        "status": string,
        "message": string,
    }

POST /status:
    req: {
        "mode": string,
        "driverID": int,
        "mileage": int,
        "fuel": int,
        "battery": int,
        "charging": bool,
    }

    res: {
        "status": string,
        "message": string,
    }
```