FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Wearables - aplikace pro sdílení vozidel Uniqway
Student:	Lukáš Norek
Vedoucí:	Ing. Václav Jirovský, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Navrhněte a implementujte aplikaci pro chytré hodinky pro uživatele systému sdílení automobilů v projektu Uniqway. Postupujte v těchto krocích:
1. Společně s vedoucím práce a ostatními účastníky projektu proveďte detailní specifikaci požadavků na aplikaci.
2. Pomocí metod softwarového inženýrství proveďte návrh aplikace včetně návrhu uživatelského rozhraní.
3. Zvolte vhodnou implementační platformu a návrh implementujte.
4. Aplikaci připojte na existující API, které v projektu již vzniklo.
5. Aplikaci vhodným způsobem otestujte a dobře zdokumentujte. 6. Navrhněte budoucí vylepšení aplikace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Wearables - aplikace pro sdílení vozidel Uniqway

Lukáš Norek

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

1. června 2020

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Václavu Jirovskému, Ph.D. za vstřícnost a ochotu při vedení této práce. Také bych rád poděkoval své rodině za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu, 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 31. května 2023.

V Praze dne 1. června 2020 . . . . . .

Odkaz na tuto práci

Norek, Lukáš. *Wearables - aplikace pro sdílení vozidel Uniqway.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace Uniqway pro nositelnou elektroniku, konkrétně pro zařízení s operačním systémem Wear OS. Aplikace slouží jako podpůrný prvek k již existující aplikaci pro operační systém Android. Účelem této aplikace je usnadnění používání některých funkcí z Android aplikace. V první části je prováděna analýza Android aplikace Uniqway a představení základních částí běžné Android aplikace. Dále jsou specifikovány požadavky vycházející z této analýzy. Následně je provedena analýza komunikace mezi Wear OS aplikací a mobilní aplikací. V návrhové části této práce je uveden programovací jazyk, který bude použit pro implementaci. Dále je proveden návrh architektury, používání aplikace a uživatelského rozhraní. V implementační části je pak představena vytvořená aplikace a postup implementace včetně použitých nástrojů. Dále jsou popsány způsoby testování aplikace. V závěru jsou pak zhodnocena možná rozšíření aplikace.

Klíčová slova nositelná elektronika, Wear OS, Android, sdílení vozidel, Uniqway, analýza aplikace, návrh aplikace, implementace aplikace, Java

Abstract

This thesis deals with the design and implementation of an Uniqway application for wearables, specifically for devices with Wear OS operating system. The application serves as a companion application for Android application, which already exists. The purpose of this application is to simplify the usage of some of the features from the Android application. The first part of this thesis is focused on the Analysis of the Uniqway Android application and introduction of the basic parts of the Android application. Follows the specification of requirements which are base on the analysis of the Android application. An analysis of communication between Wear OS application and mobile application is done next. The programming language which will be used is introduced in the design part of this thesis. Follows the design of the software architecture, of the application usage process and of the user interface. The final application and the process of the implementation, including the used tools, is introduced in the implementation part of this thesis. The testing of the application is described next. Possible extensions of the applications are discussed in the conclusion.

Keywords wearables, Wear OS, Android, car sharing, Uniqway, application analysis, application design, application implementation, Java

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Nositelná elektronika se stále více stává součástí našich životů a je o ní více slyšet. Přestože nedosahuje takového rozšíření jako chytré mobilní telefony, přináší svým uživatelům určitou míru pohodlí, jednoduchosti a zrychlení úkonů běžně spojených s mobilním telefonem. Tato práce se nezabývá obecnou prospěšností těchto zařízení ale návrhem a implementací aplikace pro Wear OS, operační systém pro chytré hodinky od firmy Google. Tato aplikace poskytne uživatelům projektu Uniqway, což je systém pro sdílení automobilů, rychlejší a snadnější přístup k funkcím z mobilní aplikace.

Již existující mobilní aplikace slouží uživatelům ke správě výpůjček automobilů. To zahrnuje historii výpůjček, zřízení nové výpůjčky nebo ukončení probíhající výpůjčky. Dále také pro zjištění informací o konkrétních vozidlech nebo zobrazení mapy dostupnosti vozidel. Výsledkem této práce je aplikace pro chytré hodinky, která část těchto funkcí poskytne uživateli bez nutnosti použití mobilního telefonu.

Teoretická části práce je rozdělena na tři části. První část část obsahuje analýzu Android aplikace a jejích současných funkcí. Druhá je zaměřena na specifikaci a analýzu požadavků na aplikaci pro chytré hodinky, která přímo navazuje na část první. Třetí část sestává z rozboru dokumentace a demo programů od firmy Google, který bude využit při implementaci a návrhu uživatelského rozhraní.

Praktická část je složena z návrhu uživatelského rozhraní a přechodů mezi jednotlivými obrazovkami. Následuje návrh základní architektury komunikace mezi aplikací na chytrých hodinkách a aplikací na mobilním zařízení, která vychází z poznatků teoretické části, dále popis implementace a na závěr testování funkčnosti.

Kapitola 1

Cíl práce

Hlavním cílem této práce je vytvoření aplikace pro systém sdílení automobilů Uniqway pro operační systém Wear OS. Prvním cílem je popis a analýza současných funkcí Android aplikace pro systém sdílení automobilů Uniqway, které budou použity v další části pro specifikaci a analýzu funkcí aplikace pro chytré hodinky. Navazujícím cílem je analýza Google dokumentace a demo programů týkajících se komunikace mezi mobilním zařízením a hodinkami.

Dalším cílem je návrh používaní aplikace a jejího uživatelského rozhraní. Navazujícím cílem je návrh softwarové architektury aplikace. Závěrečným cílem je implementace funkčního řešení a jeho následné otestování.

Kapitola 2

Analýza

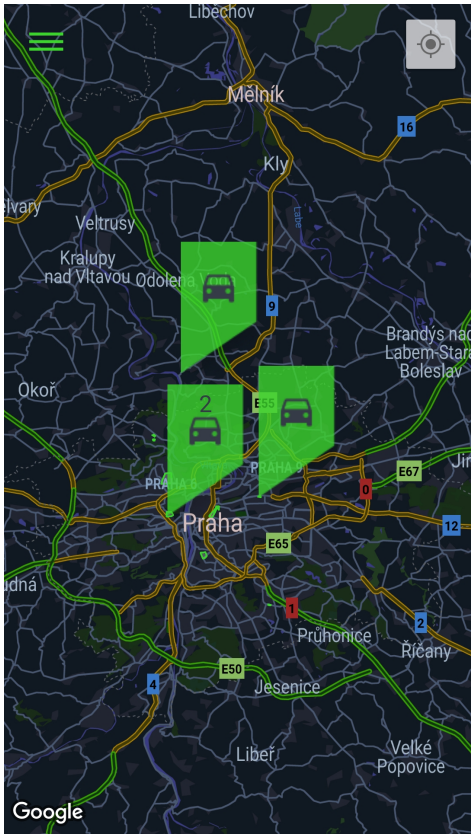
Tato kapitola obsahuje analýzu Android aplikace Uniqway [1] z hlediska uživatele. Na základě této analýzy a diskuze s týmem Uniqway byly specifikovány požadavky uvedené v další části. Následuje analýza kódu aplikace Uniqway, který bude využit při implementaci řešení. V neposlední řadě je provedena analýza dokumentace o komunikaci mezi mobilním zařízením a chytrými hodinkami.

2.1 Android aplikace Uniqway

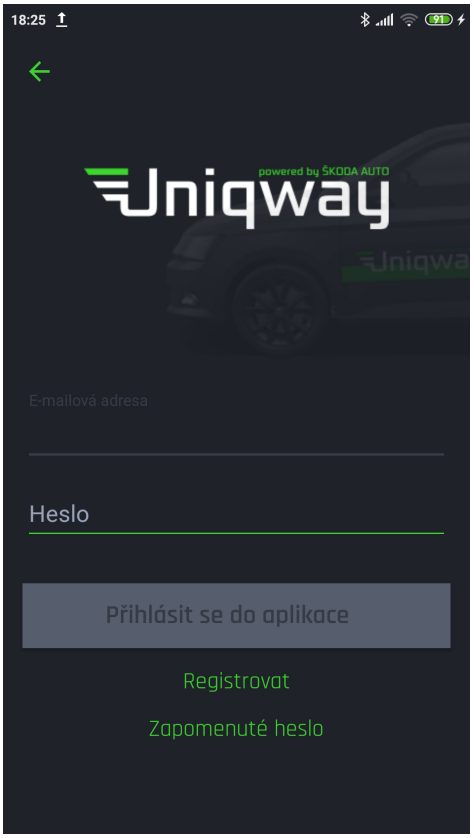
Tato aplikace slouží studentům a zaměstnancům vysokých škol k výpůjčce sdílených vozidel. Vychází z původní bakalářské práce Filipa Ravase [2] a současná verze aplikace [1], na které je analýza postavena, má mnoho vylepšení a úprav.

Po zapnutí aplikace je uživateli zobrazena obrazovka s mapou a dostupnými vozidly (obrázek 2.1). Po kliknutí na značku symbolizující určité vozidlo je zobrazen ve spodní části obrazovky detail vozidla. V horní části obrazovky je možno vidět mapu s pozicí vozidla. Stejný detail je zobrazen i v případě přihlášení. Rozdíl je ale v tom, že je uživateli již umožněno si vozidlo zarezervovat (obrázek 2.7). Bez přihlášení si lze pouze prohlížet dostupná vozidla, ale nelze si žádné zarezervovat. Po stisknutí tlačítka „přihlášení“ se zobrazí přihlašovací obrazovka s políčkem pro e-mail a heslo. V případě, kdy uživatel nevlastní účet, je možné se zaregistrovat přes webové stránky, na které se uživatel dostane po kliknutí na tlačítko „registrovat“ (obrázek 2.2).

Přihlášený uživatel si již může vypůjčit libovolné vozidlo. Záleží na něm, zda si vybere ze seznamu dostupných vozidel nebo podle pozice na mapě. Na seznam všech dostupných vozidel (obrázek 2.4) se uživatel dostane přes menu, které si lze otevřít pomocí tlačítka v levém horním rohu obrazovky. V menu si uživatel také může zobrazit historii svých jízd, obrazovku s podporou a dodatečnými informacemi, nastavení nebo se může odhlásit. Dále je zde vidět



Obrázek 2.1: Aplikace po spuštění

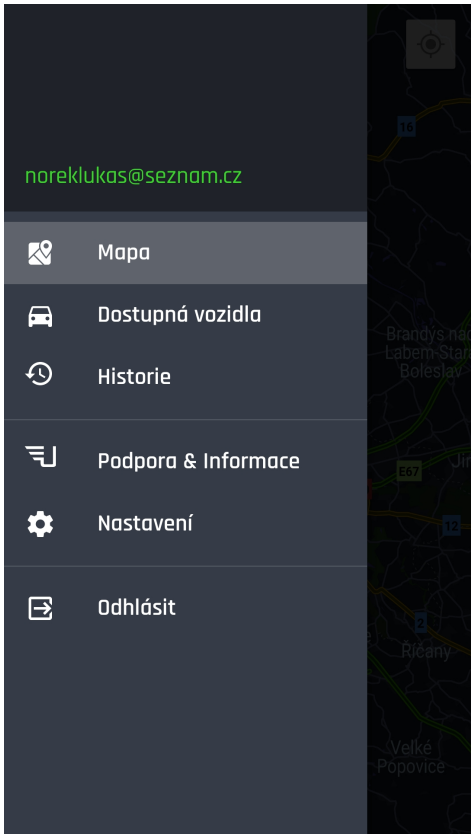


Obrázek 2.2: Přihlašovací obrazovka

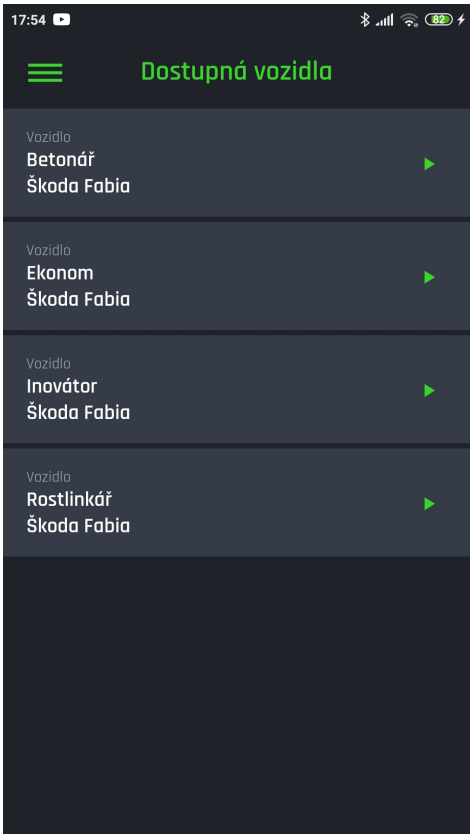
aktuálně přihlášený uživatel a je možné se z menu dostat přímo na příslušnou obrazovku v případě, kdy má uživatel aktivní jízdu (obrázek 2.3).

Po vypůjčení vozidla je uživateli zobrazena podobná obrazovka jako v případě náhledu na vozidlo. Rozdíl je ale v tom, že hlavní tlačítko slouží místo k rezervaci vozidla k jeho vrácení. Také přibylo tlačítko zámku (obrázek 2.8). Uživatel si vybere auto, které si chce vypůjčit, ze seznamu dostupných vozidel nebo z mapy. Toto auto si následně zarezervuje. Uživatel má možnost vozidlo vrátit, odemknout, zobrazit si o něm informace či spustit externí navigační program, který ho dovede k vozidlu.

Po kliknutí na tlačítko „odemknout“ je uživatel přesměrován na obrazovku odemknutí (obrázek 2.6). Po kliknutí na zobrazený přepínač je uživatel vyzván k přiložení karty na čtečku za sklem vozidla a běží časový interval, po kterém proces odemykání skončí. Po přiložení karty je vozidlo odemknuto. Zamknutí probíhá obdobně. Uživatel stiskne přepínač a je upozorněn, aby nezapomněl svou kartu ve vozidle. Poté se vozidlo zamkne a uživatel je o tom srozuměn v aplikaci.



Obrázek 2.3: Menu



Obrázek 2.4: Dostupná vozidla

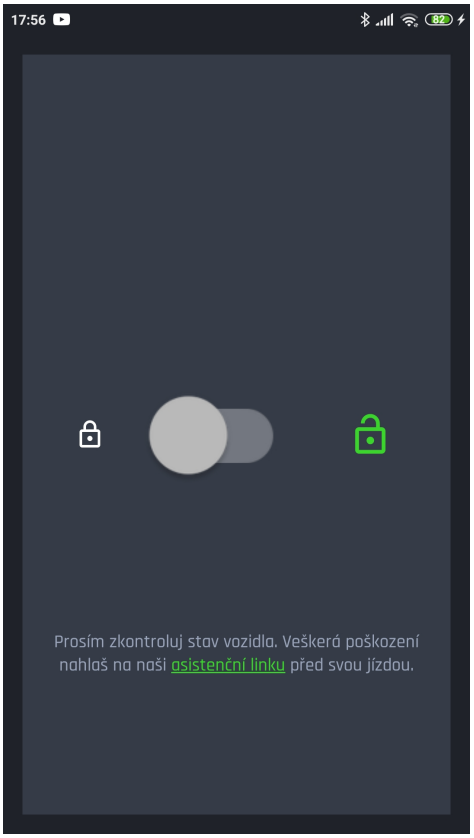
Pro dokončení jízdy uživatel stiskne příslušné tlačítko na obrazovce jízdy, následně je vyzván ke kontrole správného parkovacího místa a zkontrolování, zda si z vozidla vzal všechny své věci (zejména svou RFID kartu). Po úspěšném vrácení vozidla se uživateli zobrazí informace o jeho jízdě (např. ujetá vzdálenost, délka jízdy či cena).

2.1.1 Android aplikace obecně

Android i Wear OS aplikace se skládají ze stejných základních komponent. Základní je *Activity* [3], která reprezentuje jednu obrazovku a její logiku. GUI pro *Activity* je reprezentováno v XML souboru a říká se mu *Layout* [4]. Je v něm definováno, co se zobrazí uživateli a jaký to bude mít vzhled. Každá aktivita má svůj životní cyklus [5], ve kterém figurují metody jako např. *onResume*, *onDelete*. Ty se volají v určité fázi života *Activity*.

Pokud chceme ukládat malé množství dat typu klíč–hodnota je nejlepší použít *API SharedPreferences* [6]. Díky tomuto rozhraní lze ukládat data, která zůstanou uložena i po vypnutí aplikace. Při novém spuštění aplikace lze tato data načíst a dále používat.

Obrázek 2.5: Informace o vozidle



Obrázek 2.6: Přepínač zámku

2.2 Specifikace požadavků

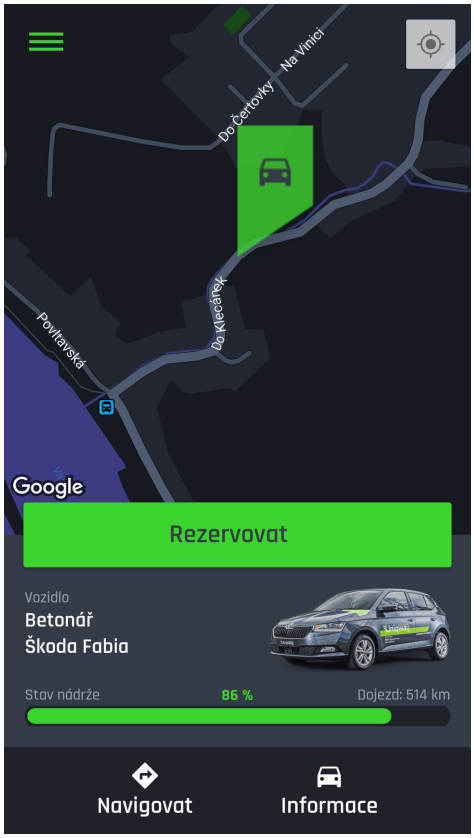
Požadavky byly stanoveny na základě analýzy Android aplikace a diskuze s týmem Uniqway včetně vedoucího práce.

2.2.1 Funkční požadavky

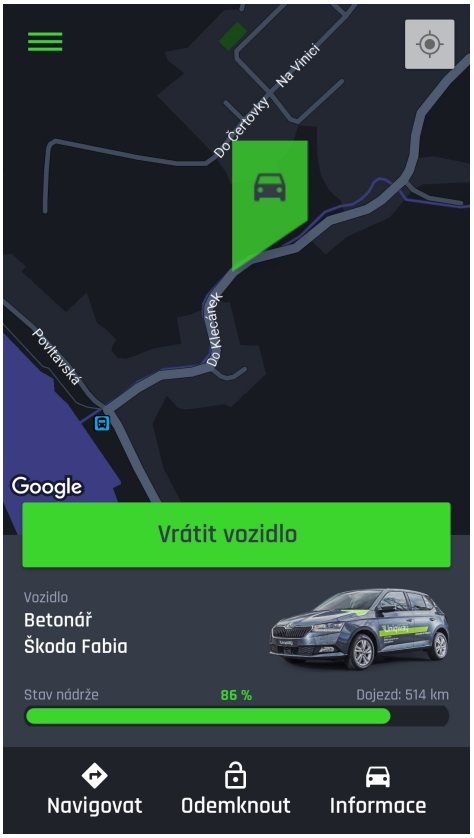
F1 – **Přihlášení:** Přihlašování a odhlašování uživatele bude probíhat na mobilním zařízení a bude následně synchronizováno s hodinkami.

F2 – **Aktuální jízda:** Uživatel si bude moci zobrazit informace o aktuálně probíhající jízdě.

F3 – **Navigace k vozidlu:** Uživatel bude mít možnost zobrazit si polohu auta na mapě a následně se k tomuto místu navigovat. Tato funkce pouze přesměruje na externí aplikaci v hodinkách jako např. Google Mapy.



Obrázek 2.7: Nezarezervované vozidlo



Obrázek 2.8: Zarezervované vozidlo

F4 – Odemykání a zamykání vozidla: V případě probíhající jízdy bude moci uživatel odemknout či zamknout vozidlo.

F5 – Vrácení vozidla: Při probíhající jízdě má uživatel možnost vrátit vozidlo a tím ukončit jízdu.

F6 – Zobrazení informací o ukončené jízdě Po úspěšném vrácení vozidla budou uživateli zobrazeny základní údaje o ukončené jízdě (např. cena, ujeté kilometry, ...).

2.2.2 Nefunkční požadavky

N1 – Jednoduchost ovládání: Ovládání bude intuitivní a zpříjemní uživateli interakci s aplikací a používání jednotlivých funkcí.

N2 – Lokalizace: Bude zachována stejná lokalizace jako má Android aplikace.

2.3 Wear OS

Wear OS je operační systém optimalizovaný pro hodinky, který je založený na operačním systému Android [7]. Pro správné fungování aplikace vyvíjené v této práci je nutná komunikace mezi hodinkami a mobilním zařízením, respektive mezi mobilní aplikací a aplikací na hodinkách. Je potřeba nejen synchronizovat datové položky, jako jsou například informace o vozidle, ale i posílat informační zprávy typu chybových hlášení či příkazů. Pro tyto účely slouží *Wearable Data Layer API* [8].

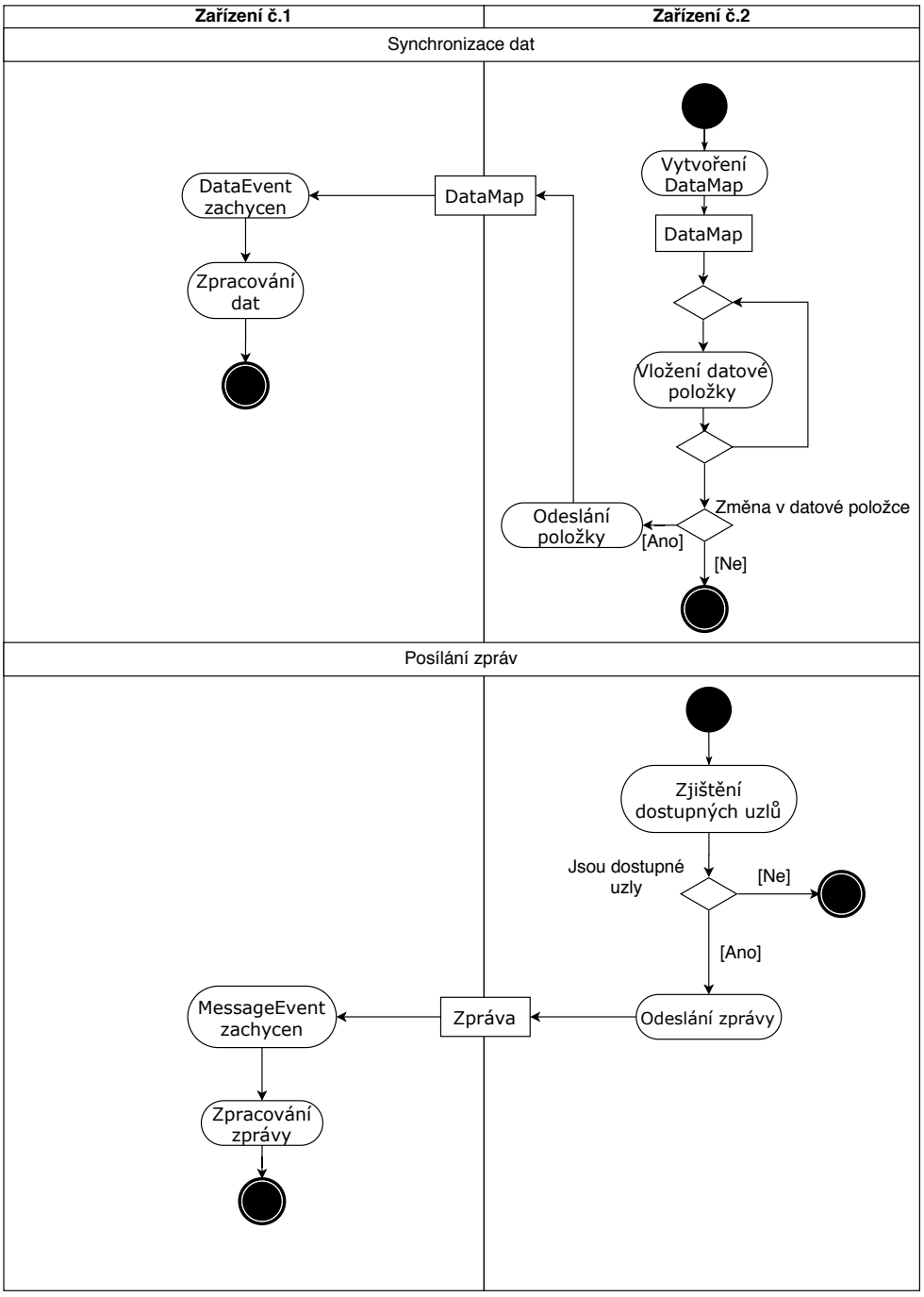
2.3.1 Synchronizace datových položek

K synchronizaci dat je doporučována třída *DataMap*. Nejprve je vytvořen *PutDataMapRequest* objekt, ve kterém se definuje cesta. Cesta je unikátní textový řetězec začínající zpětným lomítkem, který identifikuje daný *DataMap* objekt. *PutDataMapRequest* objekt obsahuje zmíněnou *DataMap*, do které lze vkládat jednu a více položek (každou s unikátním klíčem), které jsou určeny k synchronizaci. Položky mohou být různých typů, tedy například číslo nebo textový řetězec. Následně je použito *DataClient API* k vyřízení požadavku synchronizace. Toto API je nastaveno tak, aby šetřilo co nejvíce baterii zařízení. To znamená, že se položky nemusí synchronizovat ihned. Pokud to ale situace vyžaduje, lze nastavit synchronizaci jako urgentní a tím pádem k ní dojde ihned. V opačném případě může zpoždění dosahovat až třiceti minut. Pokud se datové položky od poslední synchronizace nezměnily, k synchronizaci nedochází.

Na druhé straně spojení chceme zachytit, že se datová položka změnila. K tomu je nutné, aby *Activity*, ve které chceme změny detekovat, implementovala *DataClient.OnDataChangedListener* rozhraní a v její *onResume* metodě ji musíme přidat jako tzv. listener v *DataClient API*. To, že je aktivita zaregistrována jako *listener* nám zajistí, že bude upozorněna pomocí API na změny datových položek. Také je nutné implementovat metodu *onDataChanged*. Ta dostává jako vstupní parametr *DataEventBuffer*, který obsahuje již zmíněné *DataMap* objekty. Na základě cesty lze určit, o jaký *DataMap* objekt se jedná a z něj pak můžeme získat datové položky na základě klíčů. V metodě *onPause* je pak nutné *listener* odregistrovat [9, 10]. Digram komunikace si lze prohlédnout v horní části obrázku 2.9.

2.3.2 Posílání zpráv

Často chceme jen poslat zprávu, ať už se jedná a chybové hlášení či příkaz ke spuštění *Activity*. Takovýmto zprávám se říká RPC (Remote Procedure Calls). K tomu lze využít *MessageClient API*. Samotná zpráva pak obsahuje cestu, která jednoznačně definuje akci zprávy, případně dodatečná data. Oproti před-



Obrázek 2.9: Diagram synchronizace dat a posílání zpráv

chozímu případu jsou tato data malá, neslouží k synchronizaci a jsou vždy odeslána společně se zprávou.

K poslání zprávy je nejprve nutné zjistit dostupná zařízení, která jsou považována za uzly (Nodes). Lze buď získat všechny dostupné uzly nebo pouze uzly, které mají určité vlastnosti (tzv. *Capabilities*). To jsou textové řetězce, které si sami definujeme a jsou unikátní v rámci aplikace. V obou případech tedy získáme seznam dostupných uzlů, na které je možné poslat zprávu pomocí metody *sendMessage*.

Detekce zpráv funguje podobně jako detekce změn dat v předchozí části. *Activity* musí implementovat *MessageClient.OnMessageReceivedListener* rozhraní což znamená implementovat metodu *onMessageReceived*. Jako vstupní parametr metody dostáváme *MessageEvent*, z kterého lze získat cestu a na základě ní pak provést příslušnou akci. Také je nutné *listener* zaregistrovat a odregistrovat obdobně jako v případě synchronizace dat [10, 11]. Digram komunikace si lze prohlédnout ve spodní části obrázku 2.9.

2.3.3 Zachytávání změn dat a zpráv na pozadí

Jsou případy, kdy chceme vědět o změnách v datech nebo o zprávách i když zrovna není naše aplikace v popředí. K tomu slouží třída *WearableListenerService*. Z této třídy bude dědit naše vlastní třída, ve které pak můžeme stejným způsobem, který je popsán v sekcích 2.3.1 a 2.3.2, přijímat zprávy a reagovat na změny dat [10, 12].

2.3.4 Moduly

Aby komunikace mezi aplikací na hodinkách a mobilní aplikací správně fungovala je nutné, aby obě tyto části vznikaly v rámci jednoho projektu Android Studia. Těmto částem se říká moduly [13]. Máme tedy jeden modul pro Android aplikaci a druhý modul pro Wear OS aplikaci. Oba moduly musí mít stejný název balíčku např. „com.bakalarsky.projekt“. Pokud by byly názvy balíčku u modulů rozdílné nebude komunikace fungovat. Nový Wear OS modul lze přidat i do již existujícího projektu [14]. Je také možné vytvářet další moduly, které lze například využít pro dodatečné knihovny [15].

2.3.5 Třídy specifické pro Wear OS

BoxInsetLayout Wear OS používá podobné principy pro UI jako Android a to jsou *Layout* třídy. Velký rozdíl je ale v tom, že zařízení s Wear OS mohou mít být hranatá, kulatá či kulaté s výkrojem v dolní části obrazovky. Této skutečnosti musí být uzpůsobeno i uživatelské rozhraní. *BoxInsetLayout* slouží přesné k tomuto účelu, jelikož umí rozpoznat tvar zařízeni a uzpůsobit tak UI, aby nepřesahovalo přes okraje [16].

WearableRecyclerView Tato třída slouží k zobrazení více položek pod sebou v posouvatelném seznamu. To může být využito k zobrazení seznamu položek či zobrazení delší posouvatelné obrazovky [17, 10].

AcceptDenyDialog Tuto třídu lze použít k zobrazení dialogu. Dialog může obsahovat dvě tlačítka, jedno negativní a druhé pozitivní, ale může být také pouze s jedním či úplně bez tlačítek [18, 19].

2.4 Části Uniqway aplikace potřebné v této práci

K implementaci této práce je nutné navázat na již existující kód Android aplikace [1]. V této části jsou popisovány použité třídy a API, která souvisí s funkcemi popsanými v sekci 2.2.

2.4.1 Datové třídy

Tato sekce obsahuje popis tříd, které patří do doménového modelu systému a souvisí přímo či nepřímo s funkcemi definovanými v sekci 2.2. Část doménového modelu, založená na existujícím API, je zobrazena na obrázku 2.10.

CarModel Obsahuje informace o konkrétním modelu jednotlivých vozidel.

CarReservedDetail Je členskou proměnnou třídy *Reservation*. Obsahuje detaily o zarezervovaném vozidle jako je název, pozice, model auta, stav zamknutí či informace o nádrži.

Error Obsahuje informace o chybovém hlášení. Skládá se z textu chybového hlášení a identifikačního kódu.

FuelCard Obsahuje číslo a PIN kód karty.

LoggedUser Reprezentuje přihlášeného uživatele. Obsahuje údaje jako je email, jméno, příjmení a role. Bude využita pro přenos dat o přihlášeném uživateli mezi hodinkami a mobilním zařízením.

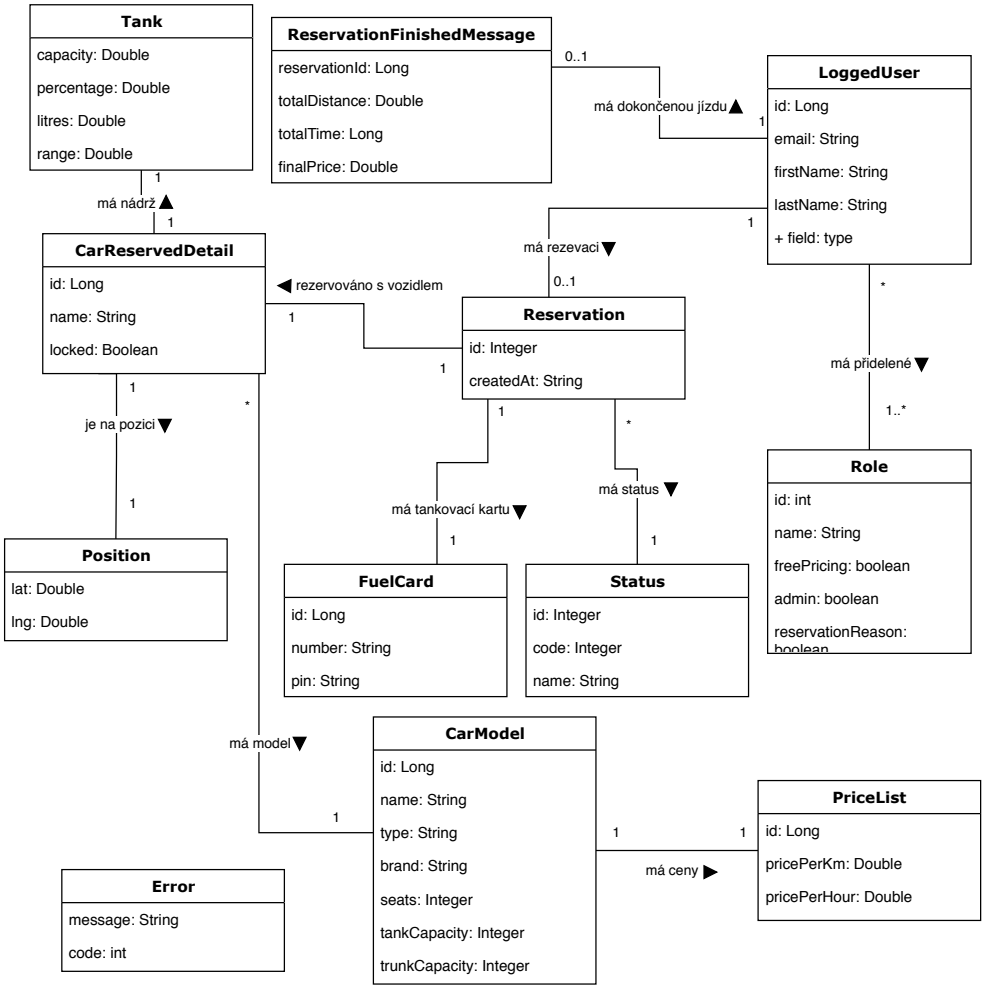
Position Reprezentuje pozici auta ve formě zeměpisné šířky a délky.

PriceList Představuje cenu za ujetý kilometr a cenu za hodinu pro konkrétní model vozidla.

Reservation Obsahuje informace o rezervaci. Konkrétně kdy byla vytvořena, rezervované vozidlo či tankovací kartu.

ReservationFinishedMessage Reprezentuje informace o dokončené jízdě jako je ujetá vzdálenost, čas jízdy či celková cena.

Role Představuje roli uživatele v systému.



Obrázek 2.10: Doménový model

Status Reprezentuje stav rezervace.

Tank Představuje nádrž vozidla. Obsahuje informace jako je kapacita, procento naplnění, počet litrů v nádrži či dojezd vozidla.

2.4.2 Aktivity a fragmenty

Aktivity a fragmenty, které přímo souvisí s funkcemi uvedenými v sekci 2.2 a analýza jejich kódu je nutná k implementaci této práce.

LoginActivity Aktivita určená k přihlášení uživatele.

MyRideFragment V případě, že má uživatel aktivní jízdu je použit tento fragment. Z tohoto fragmentu lze odemknout/zamknout vozidlo, spustit navigaci či vrátit vozidlo.

CarInfoFragment Obsahuje informace o vozidle.

LockActivity Aktivita, která je spuštěna, pokud chce uživatel zamknout vozidlo.

UnlockActivity Aktivita, která je spuštěna, pokud chce uživatel odemknout vozidlo.

2.4.3 RestApi

V aplikaci je naimplementováno rozhraní využívané ke komunikaci se serverem, k čemuž je používáno REST API. V této části jsou rozebrány funkce tohoto rozhraní související s funkcemi specifikovanými v kapitole 2.2. Metody posílají *request* na server a server vrací *response*, kterou je pak nutné v kódu zpracovat a provést na jejím základě příslušné akce.

finishReservation Tato metoda slouží k ukončení rezervace.

unlock Metoda sloužící k odemknutí vozidla.

lock Metoda sloužící k uzamknutí vozidla.

2.4.4 Ostatní

ParsistDataUtil Obsluhuje zápis a čtení dat z *SharedPreferences*.

ActivityStarter Umožňuje přehledně spouštět aktivity.

JWTtoken Způsob zabezpečení komunikace se serverem [20].

JSON Formát pro přenos dat [21].

Gson Java knihovna používaná k serializaci a deserializaci Java objektů na JSON objekty a zpátky [22].

Kapitola 3

Návrh

3.1 Programovací jazyk

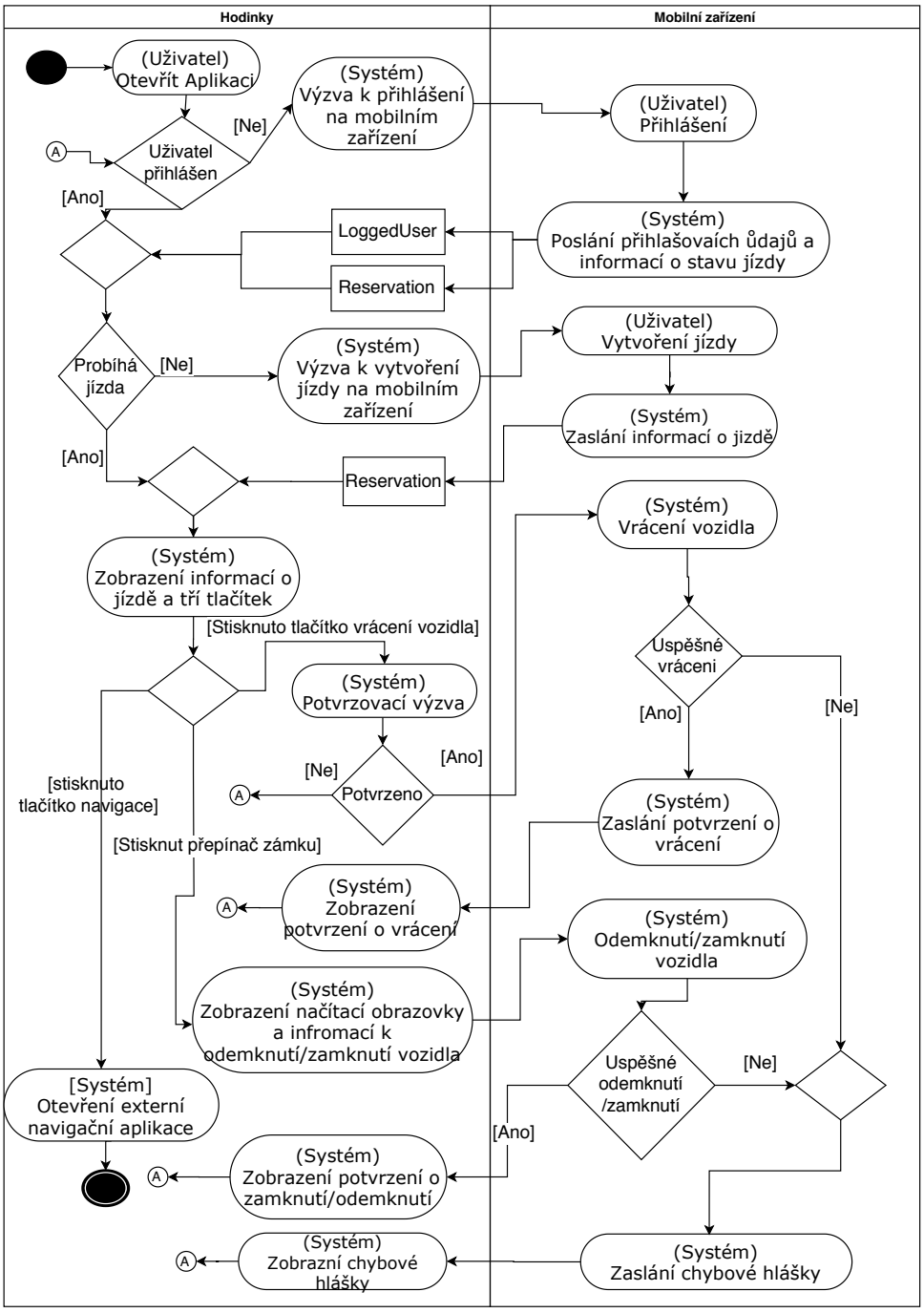
Aplikace pro Android lze psát ve více programovacích jazycích jako jsou Java, Kotlin či C++. V této práci bude k implementaci řešení použita Java hlavně z důvodu, že původní aplikace je napsána taktéž v tomto jazyce.

3.2 Používání aplikace

V této části práce je navrženo přibližné fungovaní aplikace. Po zapnutí aplikace bude zkontrolováno, zda je uživatel přihlášen. Pokud tomu tak není, je uživatel vyzván k přihlášení na mobilním zařízení. Po přihlášení jsou data o přihlášeném uživateli synchronizována s hodinkami. Když je uživatel přihlášen je zkontrolováno, zda probíhá jízda. V případě, že neprobíhá, je uživatel vyzván k rezervaci na mobilním zařízení.

Pokud jízda probíhá, jsou uživateli zobrazeny základní informace o jízdě. Dále má uživatel na výběr ze tří tlačítek a to navigace, vrácení vozidla a odemknutí vozidla. Stisknutí tlačítka navigace uživatele přesměruje do externí navigační aplikace. Stisknutí tlačítka vrácení nejprve upozorní uživatele, zda chce opravdu vozidlo vrátit. Po odsouhlasení je poslána zpráva mobilnímu zařízení, kde dojde k vrácení vozidla s využitím REST API. Uživateli je zatím na hodinkách prezentována načítací obrazovka a poté, co je doručena zpráva z mobilního zařízení s úspěšným vrácením, je o této skutečnosti zpraven.

Odemknutí vozidla probíhá obdobně jako vrácení, s tím rozdílem, že uživatel je obeznámen se stavem zamčení vozidla pomocí přepínače ve správné poloze a slovního označení. Uživatel stiskne tento přepínač a následně je upozorněn aby zkontroloval vozidlo a potvrdil odemčení/zamčení vozidla. Poté je poslána zpráva mobilnímu zařízení, které příslušný požadavek deleguje na REST API. Uživateli je zobrazena načítací obrazovka a v případě úspěchu je následně zobrazeno potvrzení.



Obrázek 3.1: Stavový diagram používání aplikace

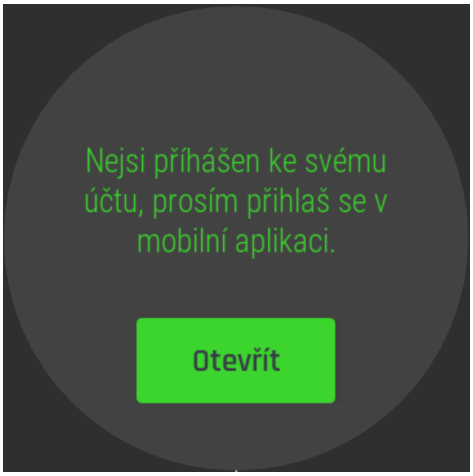
V průběhu procesu odemknutí, zamknutí a vracení může na mobilním zařízení dojít k chybě. V takové situaci je namísto potvrzení o úspěchu posláno chybové hlášení, která se zobrazí uživateli. Celý tento proces si lze přehledně prohlédnout na obrázku 3.1.

3.3 Uživatelské rozhraní

Uživatelé Androidu i Wear OS očekávají, že nové aplikace mají určitý vzhled a používají se stejným způsobem jako ty, které již znají a používají. Od toho existují pravidla nebo spíše rady, kterými by se měli vývojáři řídit [23]. Ty doporučují řídit se pravidly materiálního designu ale i pravidly pro kvalitu, kompatibilitu, bezpečnost či výkon. V této části je představen design jednotlivých obrazovek aplikace, který se drží těchto pravidel. Zároveň byl dodržen podobný grafický design jako má původní Android aplikace Uniqway.

3.3.1 Nepřihlášený uživatel

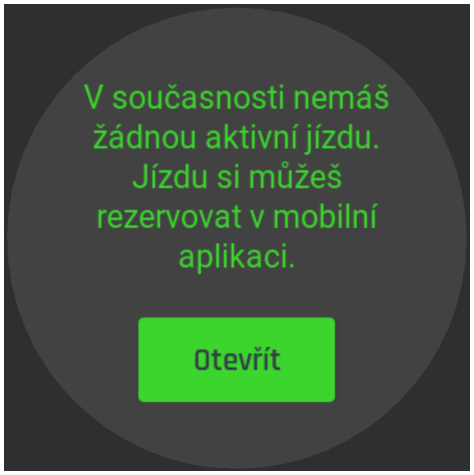
Pokud uživatel spustí aplikaci a není přihlášený, je vyzván k přihlášení na mobilním zařízení. Po stisknutí tlačítka se otevře aplikace na mobilním zařízení s přihlašovací obrazovkou. Návrh obrazovky nepřihlášeného uživatele je na obrázku 3.2.



Obrázek 3.2: Návrh nepřihlášený uživatel

3.3.2 Žádná aktivní jízda

Pokud je uživatel přihlášený ale nemá žádnou aktivní jízdu, je vyzván k zarezervování jízdy na mobilním zařízení. Jako v případě přihlášení se na mobilním zařízení otevře aplikace Uniqway, kde si uživatel může zarezervovat novou jízdu. Návrh této obrazovky je na obrázku 3.3.



Obrázek 3.3: Návrh žádná aktivní jízda

3.3.3 Aktivní jízda

V případě aktivní jízdy je uživateli zobrazena obrazovka se základními informacemi jako je název auta, model auta, stav nádrže či dojezdová vzdálenost. Příklad je na obrázku 3.4a. Pokud uživatel přejede prstem po obrazovce směrem nahoru je posunut na další obrazovku, která obsahuje tlačítka akcí. Tato obrazovka je znázorněna na obrázku 3.4b.

(a) Návrh informace o jízdě

(b) Návrh tlačítka k akcím

Obrázek 3.4: Návrh obrazovek aktivní jízdy

3.3.4 Načítací obrazovka

Pokud je prováděna akce, která by mohla trvat déle, je uživateli zobrazena načítací obrazovka znázorněná na obrázku 3.5. Tato obrazovka obsahuje kolečko znázorňující načítaní a text s krátkým popisem typu načítání.

Obrázek 3.5: Návrh načítací obrazovky

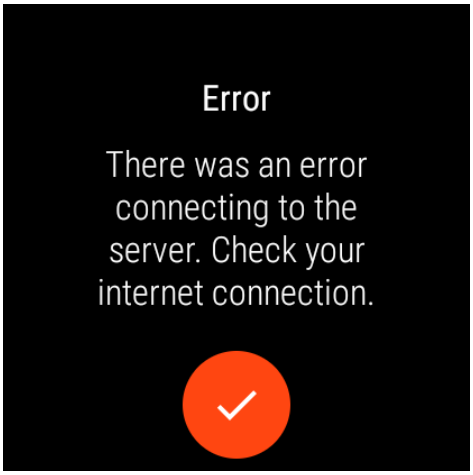
3.3.5 Upozornění a chybová hlášení

V několika případech je potřeba od uživatele potvrzení, jestli chce danou akci opravdu vykonat. V takovém případě je zobrazena výzva znázorněná na obrázku 3.7, která obsahuje stručný obsah potvrzení a dvě tlačítka. Jedno slouží k odmítnutí potvrzení a druhé k potvrzení. Po potvrzení je provedena příslušná akce.

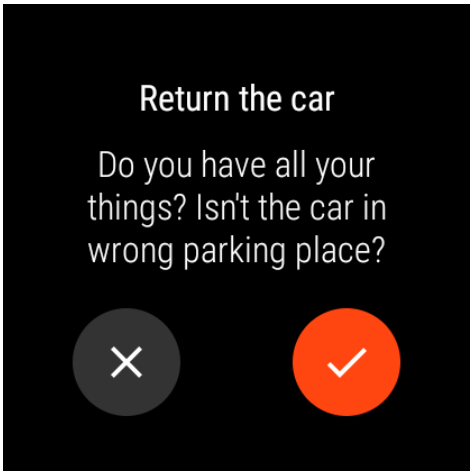
Obrazovka chybového hlášení může být zobrazena kdykoliv za běhu aplikace. Vztahuje se jak k chybám v aplikaci samotné, tak i k chybám delegovaným z mobilního zařízení. Oproti potvrzení má jen jedno tlačítko, které slouží k potvrzení přečtení chybového hlášení. Tuto skutečnost lze vidět na obrázku 3.6. V obou případech bude použita třída *AcceptDenyDialog* popsaná v sekci 2.3.5.

3.4 Architektura

Pro tuto práci byla zvolena architektura MVP, což znamená Model–View–Presenter. Tato architektura odděluje business logiku a datový model od prezentační vrstvy. Dále poskytuje možnost jednoduššího jednotkového testování a rozšiřitelnosti kódu. Skládá se ze tří částí:



Obrázek 3.6: Chybové hlášení



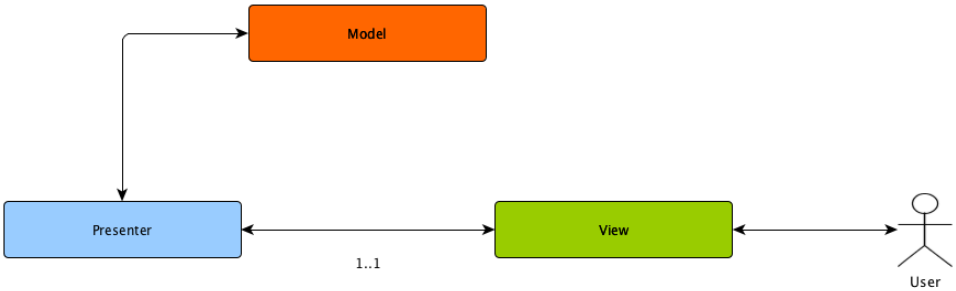
Obrázek 3.7: Výzva

Model Sestává se ze tříd reprezentujících data. V této práci jsou to třídy *LoggedUser*, *Reservation* či *CarReservedDetail*. Tato apliace není přímo napojena na databázi, ale používá Wearable Data Layer API popsané v části 2.3, ze které budou následně data uložena v *SharedPreferences*. Odtud pak budou získávána a ukládána data pomocí *Presenter* tříd.

View Slouží k zobrazení informací uživateli a detekci interakce od uživatele. Do této kategorie patří třídy dědící ze tříd *Activity* či *Fragment*, a také XML soubory s definovanými *Layouty* aplikace.

Presenter Slouží jako prostředník mezi *View* a *Modelem*. Zpracovává uživatelské požadavky od *View* a zpracovává data s pomocí *Modelu*. Tato data pak vrátí *View* třídám, které je zobrazí uživateli.

Na obrázku 3.8 lze vidět, že mezi *Modelem* a *View* není přímé spojení a vše je delegováno přes *Presenter*. *Presenter* a *View* mívají zpravidla vazbu jedna ku jedné [24, 25].



Obrázek 3.8: Architektura MVP

Kapitola 4

Realizace

4.1 Prostředí pro vývoj

Při vývoji bylo použito Android studio IDE ve verzi 3.6.3 [26], které je oficiálně podporované, a tedy nejvhodnější pro vývoj aplikací. Nabízí mnoho funkcí, usnadňujících práci programátorovi, mezi které patří emulátor zařízení, napovídání při psaní kódu, nástroje pro sestavení (build) projektu či editor uživatelského rozhraní (layout editor).

4.2 Správa verzí

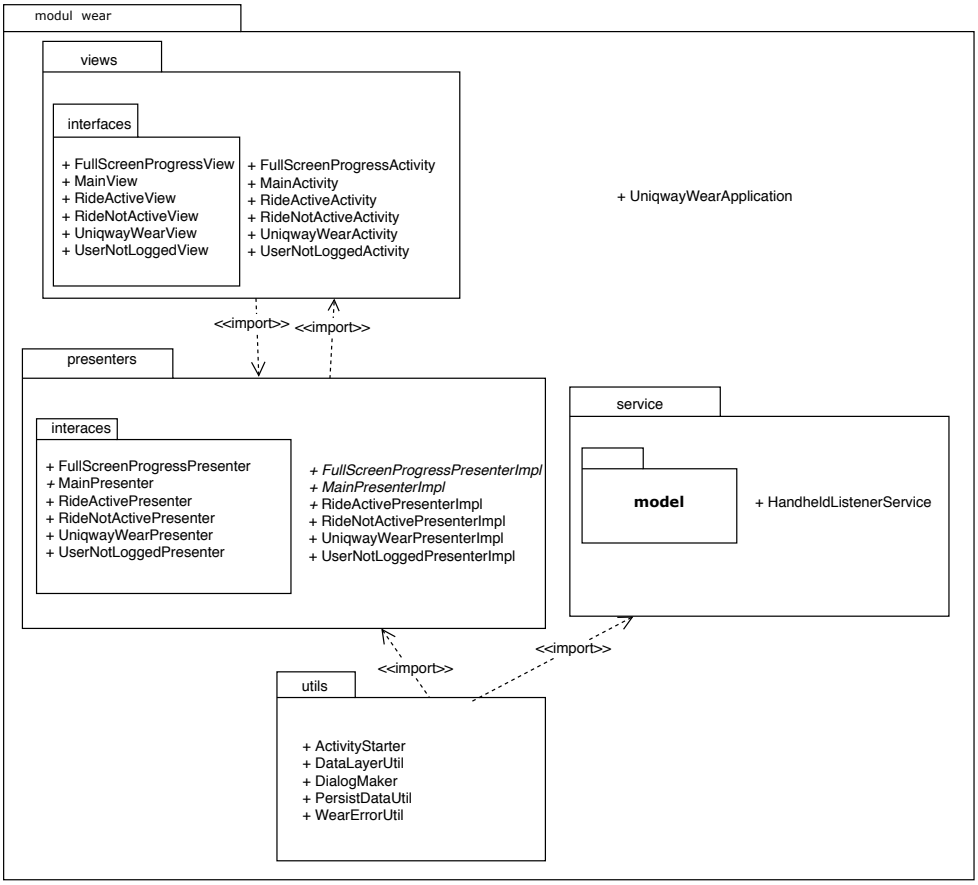
Na spravování verzí při vývoji byl použit nástroj Git a jako úložiště byl použit GitLab, který již byl v rámci projektu používán. Bylo tak velmi jednoduché navázat na současnou verzi Android aplikace a zároveň to přispěje k usnadnění nasazení Wear OS aplikace do provozu.

4.3 Struktura projektu

Projekt je rozdělen do tří modulů. Jeden modul je původní Android aplikace, další je Wear OS aplikace a poslední je modul nazvaný *shared*. K třídám v tomto modulu mají přistup oba předchozí moduly a dochází tak ke sdílení kódu.

4.3.1 Modul s Wear OS aplikací

Modul obsahující Wear OS aplikaci, nazvaný *wear*, byl na základě architektury zvolené v části 3.4 rozdělen do balíčků, které zajistí přehlednost a snazší orientaci v kódu. Níže jsou popsány jednotlivé balíčky, které patří do modulu *wear*, a jejich rozdělení lze vidět na obrázku 4.1.



Obrázek 4.1: Diagram balíčků modulu wear

Třída *UniqwayWearApplication* se nachází v kořenovém balíčku nazvaném „cz.uniqway.uniqway“ a slouží k uchování globálního stavu celé aplikace, lze díky ní například získat aktuálně běžící aktivitu, či zjistit, zda nějaká aktivita běží. Tato třída dědí od třídy *Application* [27]. Balíčky popsané níže se také nacházejí v kořenovém balíčku „cz.uniqway.uniqway“, takže jejich plnohodnotný název je vždy „cz.uniqway.uniqway.[název balíčku]“.

views Zde se nachází *activity* třídy, které se starají o zobrazení uživatelského rozhraní uživateli a patří tedy do View části architektury MVP. Tento balíček obsahuje podbalíček *interfaces*, ve kterém se nachází rozhraní pro implementaci funkčností této vrstvy a která jsou používána v rámci presenter tříd. Každá aktivita má jednu třídu z tohoto balíčku, která jí přísluší.

presenters Obsahuje presenter třídy, které se starají o business logiku aplikace a fungují jako prostředník mezi modelem a view. Jako v případě view obsahuje ještě podbalíček *interface* s rozhraním pro presentery. Každá aktivita má svou vlastní presenter třídu.

service Obsahuje *HandheldListenerService* což je služba, která na pozadí poslouchá na podněty z Data Layer API. Dále také obsahuje podbalíček *model* s třídami reprezentujícími data.

utils Obsahuje pomocné třídy, které zaobalují funkcionalitu využitou na více místech aplikace. Mezi takové třídy patří například *ActivityStarter*, který slouží ke spouštění aktivit, či *PersistDataUtil* používaný k přístupu do SharedPreferences

4.3.2 Sdílený modul

Modul *shared* obsahuje tyto čtyři třídy.

CommonConstants Tato třída obsahující konstanty využívané při komunikaci mezi aplikacemi. Mezi tyto konstanty patří všechny cesty definující typ zprávy ale i klíče jednotlivých datových položek či další konstanty využívané na obou stranách komunikace.

CommonDataLayerUtils Nalezení dostupných uzlů, příprava a odeslání zprávy či příprava a synchronizace datové položky je pro obě strany komunikace stejná, a tak je implementace spojena v této třídě.

NoNodesException V případě, že nejsou nalezeny žádné dostupné uzly, je vyhozena tato výjimka.

InternalMessageEvent Tato třída je používána k reprezentaci interních zpráv posílaných v rámci aplikace pomocí knihovny *EventBus*. [28] Další informace o posílání zpráv jsou v sekci 4.4.1.

4.3.3 Úpravy v modulu Android aplikace

Do modulu s Android aplikací bylo přidáno pět nových tříd. Všechny tyto třídy jsou spjaty s komunikací mezi mobilní aplikací a aplikací v hodinkách. Třída *WearDataLayerUtil* slouží k sjednocení posílání zpráv a synchronizace datových položek a více je popsána v sekci 4.4.1.

Naopak ke zpracovávání zpráv slouží třída *WatchListenerService*, která je rozebírána v sekci 4.4.2. Zde je také rozebráno zpracování požadavků týkajících se zamknutí a odemknutí vozidla, ke kterému se přímo vážou třídy *WaitForLockCheckServiceRunnable*, *WaitForUnlockCheckServiceRunnable* či *RunnerFromService*. Slouží ke komunikaci se serverem při procesu zamykání a odemykání a běží na pozadí. Tyto třídy jsou převzaty ze tříd *Runner*,

WaitForLockCheckRunnable a *WaitForUnlockCheckRunnable* z původní implementace Android aplikace a sloužily ke stejnému účelu s tím rozdílem, že byly vázány na aktivitu. Nově přidané třídy plní stejnou funkci, ale jsou spouštěny a vázány na *WatchListenerService*.

Dále byly provedeny změny v hlavní aktivitě mobilní aplikace s názvem *MainActivity* a třídě *ActivityStarter*. Později jmenovaná třída slouží ke spouštění aktivit. Zde byla přidána možnost spuštění *MainActivity* ze služby. Také bylo upraveno spouštění *LoginActivity*, sloužící k přihlašování, tak aby po ukončení vracela výsledek. To pak slouží v aktivitě *MainActivity* k detekování přihlášení a následné synchronizaci přihlášeného uživatele do aplikace v hodinkách.

MainActivity je hlavní třídou mobilní aplikace a je tedy první která se spustí při zapnutí mobilní aplikace. Metoda *onResume* je spuštěna při každém zapnutí aplikace či přechodu aplikace do popředí [5]. Z tohoto důvodu je v metodě provedena synchronizace přihlášeného uživatele a zarezervované jízdy do aplikace v hodinkách, a to pomocí metod *syncLoginWithWearable* a *syncReservationWithWearable* třídy *WatchDataLayerService*. Stejnými metodami jsou pak synchronizovány údaje o přihlášeném uživateli po dokončení přihlášení a údaje o rezervace v případě jejího vytvoření či ukončení.

4.4 Komunikace

Stěžejní částí aplikace na hodinkách je komunikace s mobilním zařízením. Tato sekce obsahuje popis implementace této komunikace. Její základní principy jsou popsány v sekci 2.3. V úvodu této sekce je popsáno odesílání zpráv a synchronizace datových položek v konkrétních třídách. Dále jsou pak rozebrány způsoby příjmu zpráv.

Pro správné fungování aplikace je nutné, aby na obou zařízeních byly přijímány zprávy i když právě neběží. Třída *WearableListenerService* slouží právě k tomuto účelu a spouští se pouze když dostane podnět od Android API [29]. V této práci byla vytvořena třída *HandheldListenerService* na straně hodinek a *WatchListenerService* na straně mobilního zařízení. Obě tyto třídy rozšiřují již zmíněnou třídu *WearableListenerService* a implementují její metody *onDataChanged* a *onMessageReceived*. Tuto třídu nemusíme registrovat jako listener, jako tomu bylo u aktivit, protože si tuto funkcionalitu zajišťuje sama.

4.4.1 Posílání zpráv a synchronizace dat

Aplikace musí být schopná přijímat zprávy a změny dat ale v první řadě je musí umět posílat. Pro tyto účely slouží třída *DataLayerUtil* na straně hodinek a *WearDataLayerUtil* na straně mobilního zařízení. Obě tyto třídy dědí od třídy *IntentService* [30], což jim umožňuje posílat zprávy asynchronně v pozadí a nezávisle na uživatelském rozhraní.

Část kódu, týkající se nalezení uzlů v síti, odeslání zprávy nebo synchronizace dat, je totožná pro obě strany komunikace. Z toho důvodu byla vytvořena sdílená knihovna nazvaná *shared*, která umožní mít tento kód jen jednou a sdílet ho mezi aplikací pro hodinky i mobilní aplikací. O společnou implementaci kódu k odesílaní a synchronizaci datových položek, popsanou v sekcích 2.3.2 a 2.3.1, se stará třída *CommonDataLayerUtils*. K odesílání zpráv je nejprve nutné zjistit dostupné uzly.

```
public static Collection<String> getNodesWithCapability(
                                        Context context,
                                        String capability)
{
    HashSet<String> nodeResults = new HashSet<>();

    Task<CapabilityInfo> nodeListTask = Wearable.
                        getCapabilityClient(context).
                        getCapability(capability,
                        CapabilityClient.FILTER_REACHABLE);

    try {
        Set<Node> nodes = Tasks.await(nodeListTask).
                                        getNodes();
        for(Node node : nodes){
            nodeResults.add(node.getId());
        }
    }
    catch (ExecutionException exception) {
        Log.e(TAG, "Task failed: " + exception);
    }
    catch (InterruptedException exception) {
        Log.e(TAG, "Interrupt occurred: " + exception);
    }
    return nodeResults;
}
```

Výpis kódu 4.1: Získání dostupných uzlů s definovanou „capability“

Pro účely této aplikace ale nepotřebujeme odesílat data na všechny dostupné uzly, a tak jsou vyhledané uzly omezeny pouze na ty, které mají mezi svými *Capabilities* řetězec, který je definovaný ve třídě *CommonConstants* jako CAPABILITY_UNIQWAY. Získání uzlů si lze prohlédnout ve výpisu kódu 4.1. Pokud neexistují žádné uzly vyhovujícím těmto požadavků je vytvořena výjimka *NoNodesException*, na základě které je uživatel obeznámen s nepři-

```
public static void sendMessage(Context context,
                               final String path,
                               byte[] payload)
                               throws NoNodesException
{
    Collection<String> nodes = getNodesWithCapability(
                        context,
                        CommonConstants.CAPABILITY_UNIQWAY);
    if(nodes.isEmpty()) {
        throw new NoNodesException(
                        new Throwable(context.getClass().
                        getSimpleName() + ": sendMessage"));
    }
    for (final String node : nodes) {
        Task<Integer> sendMessageTask =
                        Wearable.getMessageClient(context).
                        sendMessage(node, path, payload);
    }
}
```

Výpis kódu 4.2: Odeslání zprávy do všech dostupných uzlů

pojenými uzly a je mu doporučeno zkontrolovat připojení. Odesílání zpráv je prováděno asynchronně a nelze tedy přímo z této třídy vytvářet chybové dialogy pro uživatele. Z tohoto důvodu jsou zprávy posílány v rámci aplikace pomocí knihovny *EventBus* [28], která umožňuje komunikaci mezi službou a uživatelským rozhraním. Kód použitý k odesílání zpráv si lze prohlédnout ve výpisu 4.2. Získání uzlů a posílání zpráv je založeno na kódu z dokumentace popsaném sekci 2.3.2.

DataLayerUtil na straně hodinek obsahuje pouze obecné metody k posílání zpráv. Oproti tomu *WearDataLayerUtil* obsahuje metody, které usnadňují některé nejčastěji posílané zprávy. Mezi takové patří posílaní chybových zpráv, které má vždy stejnou cestu a není nutné ji vždy znovu zadávat. Mezi další patří synchronizace přihlášení či rezervace. Rezervace navíc vyžaduje před odesláním další přípravu. V základu třída *Reservation* obsahuje členskou proměnnou *CarReservedDetail*, která by měla obsahovat model vozidla. Model vozidla je ale null, a tak je nutné nejprve model doplnit a poté synchronizovat rezervaci.

4.4.2 WatchListenerService

Tato třída se stará o obsluhu dotazů z hodinek na straně mobilního zařízení. Z hodinek přijde zpráva a tato služba na ní příslušně reaguje. Zprávy jsou rozlišené unikátní cestou, která určuje jejich účel. Některé zprávy vyřídí služba na pozadí a pouze nazpět pošle zprávu s výsledkem. Další ale od uživatele vyžadují interakci s mobilní aplikací a ta je tedy spuštěna.

Zjištění přihlášení obsahuje obě tyto možnosti. V případě, že na hodinkách není žádný uživatel přihlášen, je možnost pomocí tlačítka poslat dotaz na mobilní aplikaci. Pokud mobilní aplikace má přihlášeného uživatele, tak pouze tyto data synchronizuje s hodinkami. Tato data jsou ve formě třídy *LoggedUser* serializované do JSON objektu. V opačném případě se spustí mobilní aplikace s přihlašovací aktivitou a uživatel se může přihlásit. Po přihlášení jsou data synchronizována s hodinkami.

Malý displej hodinek není vhodný k rezervaci vozidla, proto je tato akce delegována do mobilní aplikace. Spustí se hlavní aktivita, kde už si uživatel může mnohem jednodušeji vozidlo vybrat a zarezervovat. Po zarezervování jsou data o jízdě synchronizována s hodinkami.

K vrácení auta není potřeba spuštění aplikace a vše se děje v rámci služby. Pomocí metod třídy *RestApi* je poslán požadavek na server. Pokud je odpověď od serveru kladná, je hodinkám poslána zpráva o úspěšném vrácení společně s údaji o ukončené jízdě. V opačném případě je posláno chybové hlášení.

Odemknutí a zamknutí auta probíhá také pouze na pozadí v rámci služby. Po přijetí požadavku od hodinek je poslán požadavek na server metodou *lock*, respektive *unlock* třídy RestApi. Ta může vrátit buď pozitivní nebo negativní odpověď. Negativní odpověď je delegována hodinkám. V případě pozitivní odpovědi u odemknutí se spustí proces dotazů v určitých časových intervalech, které se dotazují serveru, zda již byla přiložena RFID karta k senzoru za sklem vozidla. Pokud do uplynutí časového intervalu na straně serveru nebo počtu dotazů na straně služby není karta přiložena ke čtečce na okně je na hodinky poslána zpráva oznamující tuto skutečnost. V případě zamykání je situace podobná co se týče dotazů na server. Jen se nečeká na přiložení karty ale po uplynutí určitého intervalu je poslaná zpráva, že zamknutí bylo úspěšné.

Komunikace se serverem při vrácení, odemknutí a zamknutí auta byla převzata z aktivit *LockActivity, UnlockActivity* a fragmentu *MyRideFragment* mobilní aplikace a byla upravena pro běh ve službě na pozadí a komunikaci s hodinkami. Současná implementace těchto funkci v mobilní aplikaci je svázána s uživatelským rozhraním a nedovoluje přímé použití kódu, vztahujícího se k těmto funkcím a byla by nutná refaktorizace kódu mobilní aplikace, což ale není součástí této práce. Z tohoto důvodu byl kód zkopírován a upraven. Metodu *onMessageReceived*, která zpracovává zprávy si lze prohlédnout na výpisu 4.3.

```
public void onMessageReceived(MessageEvent messageEvent)
{
    Log.d(TAG, "message received: " + messageEvent);
    String path = messageEvent.getPath();
    switch (path){
        case CommonConstants.PATH_LOGIN_REQUEST:
            if(!PersistDataUtil.isLoggedIn()) {
                ActivityStarter.startMainActivity(this);
                ActivityStarter.startLoginActivity(this);
            } else {
                WearDataLayerUtil.syncLoginWithWearable(
                    getContextWearListenerService());
                WearDataLayerUtil.startWearMainActivity(
                    getContextWearListenerService());
            }
            break;
        case CommonConstants.
                PATH_START_HANDHELD_MAIN_ACTIVITY:
            ActivityStarter.startMainActivity(this);
            break;
        case CommonConstants.PATH_RESERVATION_FINISH:
            finishReservation();
            break;
        case CommonConstants.PATH_UNLOCK_REQUEST:
            unlock();
            break;
        case CommonConstants.PATH_LOCK_REQUEST:
            lock();
            break;
        default:
            Log.d(TAG, "no action for path: " + path);
    }
}
```

Výpis kódu 4.3: Metoda *onMessageReceived* ve službě *WatchListenerService*

4.4.3 HandheldListenerService

Tato třída, na rozdíl od předchozí, nepřijímá jen zprávy, na které pak reaguje, ale zároveň přijímá i informace o změně dat na straně mobilního zařízení. Jak funguje synchronizace dat je popsáno v sekci 2.3.1. Mezi tyto datové položky patří *LoggedUser* s informacemi o přihlášeném uživateli a *Reservation* s informacemi o stavu rezervace. V obou případech platí, že data jsou uložena do *SharedPreferences* pomocí *PersistDataUtil*.

Pokud je aplikace na hodinkách v popředí je provedena změna uživatelského rozhraní tak, aby odpovídalo aktuálnímu stavu. Tedy v případě, že je synchronizováno, že je uživatel nepřihlášen na hodinkách, je uživatel odhlášen a je mu zobrazena výzva k přihlášení. V případě, že je přihlášen a je přijata nová rezervace, jsou uživateli zobrazeny informace o jízdě a možnost další interakce s aplikací. V opačném případě je uživatel vyzván k rezervaci vozidla v mobilní aplikaci.

O chybové zprávy se tato služba nestará, protože ty jsou nutné zobrazovat pouze pokud aplikace běží a uživatel s ní interaguje. O ně se tedy starají konkrétní aktivity, z kterých je chybové hlášení následně zobrazeno.

4.5 Aktivity a nové uživatelské rozhraní

Aktivity, společně s XML layout soubory, reprezentují to, co se uživateli zobrazuje. V této sekci je popsáno, ke kterým funkcím popsaným v analytické části práce se jednotlivé aktivity vztahují, jak fungují a v neposlední řadě jejich finální vzhled. Každá aktivita, s výjimkou *CormfirmationActivity* a *Dialog*, implementuje rozhraní definované v příslušném *View* a umožňuje tak třídám z balíčku *presenters* interagovat s jejich obsahem. Tyto dvě třídy také nemají, na rozdíl od ostatních, odpovídající presenter. Ten se stará o logiku konkrétní aktivity a poskytuje jí data.

4.5.1 MainActivity

Tato aktivita slouží jako spouštěcí aktivita celé aplikace a je tedy první, která se spustí. Při jejím vytváření je pomocí *PersistDataUtil* zjištěn stav přihlášení či rezervace a na základě toho je spuštěna příslušející aktivita. Na výpisu kódu 4.4 z presenteru příslušejícímu k *MainActivity* lze vidět výběr aktivity ke spuštění na základě přihlášení uživatele a aktivní jízdy. To zajišťuje, že pokud se uživatel odhlásil či změnil stav rezervace na mobilním zařízení, když aplikace neběžela, tak při zapnutí se tento stav přečte ze *SharedPreferences*. Tam byl zapsán při běhu služby *HandheldListenerService*, která změnu zaznamenala.

```
public void onResume() {
    int activityCode;
    if (PersistDataUtil.isLoggedIn()) {
        if(PersistDataUtil.hasReservation()){
           activityCode = RIDE_ACTIVE_ACTIVITY;
        }
        else {
            activityCode = RIDE_NOT_ACTIVE_ACTIVITY;
        }
    } else {
       activityCode = USER_NOT_LOGGED_ACTIVTY;
    }
    mainView.startActivityByCode(activityCode);
}
```

Výpis kódu 4.4: Spuštění aktivity na základě obsahu SharedPreferences ve třídě *MainPresenterImpl*

4.5.2 UniqwayWearActivity

Z této aktivity dědí všechny dále zmíněné aktivity. Jejím hlavním účelem je zajistit jednu implementaci zachytávaní zpráv o chybových hlášeních a jejich zobrazování uživateli. Také se stará o zobrazení chyb, které mohou nastat při posílání zpráv, což je popsáno v sekci 4.4.1.

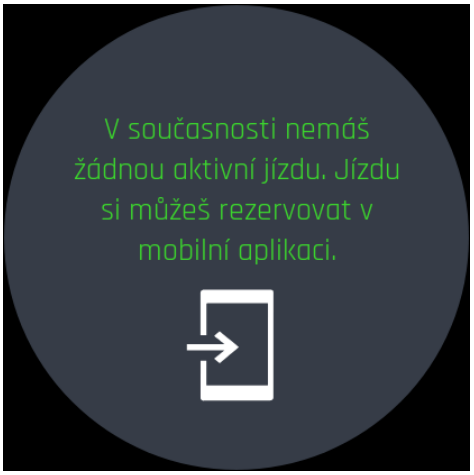
4.5.3 FullScreenProgressActivity

Slouží k oznámení uživateli, že musí počkat na výsledek nějaké své akce. Obsahuje stručný text s informacemi o důvodu čekání a otáčející se kruh symbolizující načítání. Konečný vzhled lze nalézt na obrázku 4.6. Tato aktivita se ukončuje po tom, co přijme zprávu z mobilního zařízení, že akce na kterou je čekáno je dokončena nebo pokud je doručeno chybové hlášení. V případě chybového hlášení je zobrazen dialog s popisem chyby a po stisknutí tlačítka na tomto dialogu je aktivita ukončena.

4.5.4 UserNotLoggedActivity

Tato aktivita je použita v případě, že uživatel není přihlášen. Zobrazuje uživateli výzvu k přihlášení na mobilním zařízení a tlačítko k otevření aplikace na mobilním zařízení. Výsledný vzhled obrazovky si lze prohlédnout na obrázku 4.2.

Obrázek 4.2: Nepřihlášený uživatel



Obrázek 4.3: Neaktivní jízda

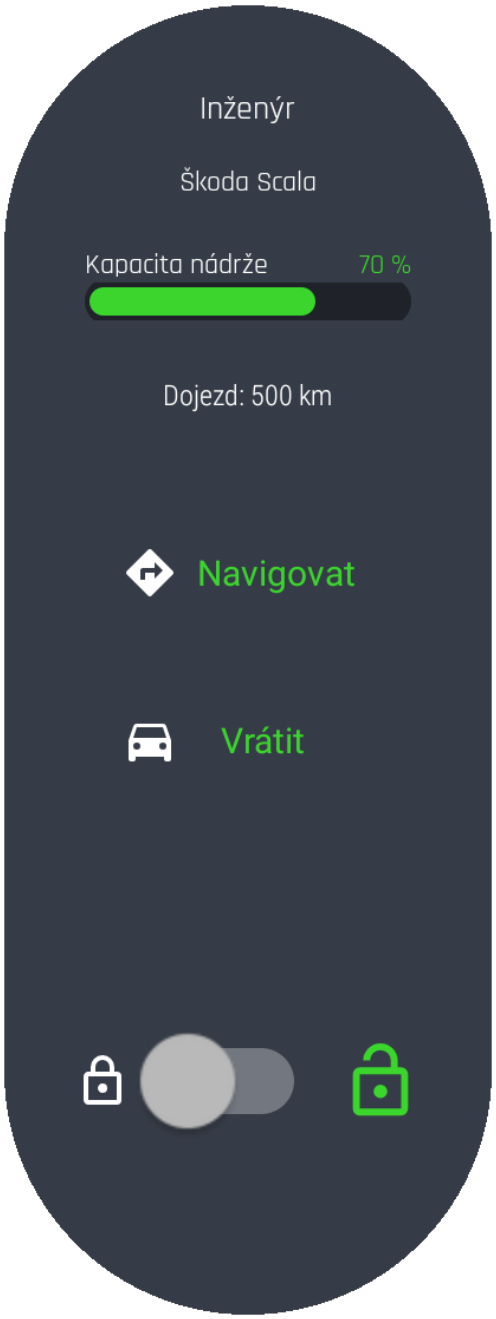
4.5.5 RideNotActiveActivity

Pokud už je uživatel přihlášený, ale nemá žádnou aktivní jízdu, je zobrazena tato aktivita. Zde je uživatel požádán o vytvoření jízdy v mobilní aplikaci a je mu k dispozici tlačítko, kterým mobilní aplikaci spustí. Na obrázku 4.3 si lze prohlédnout výsledný vzhled obrazovky.

4.5.6 RideActiveActivity

Tato aktivita zobrazuje uživateli údaje o právě probíhající jízdě. Její obrazovka je posouvatelná a náhled si lze prohlédnout na obrázku 4.4. Po posunutí se uživateli zobrazí možnosti interakce s jeho jízdou. Najdeme zde tlačítko navigace, které uživatele přesměruje do externí navigační aplikace. Dále se zde nachází tlačítko vrácení vozidla. Po stisknutí tohoto tlačítka se uživateli zobrazí dialog, kde je uživatel vyzván ke kontrole vozidla a má možnost vrácení potvrdit či odmítnout. Dialog je na obrázku 4.8. Po potvrzení je zobrazena *FullScreenProgressActivity* se stručným popisem na co se čeká. Po úspěšném vrácení skončí *FullScreenProgressActivity* a je nahrazena dialogem se stručným shrnutím jízdy, které lze vidět na obrázku 4.5.

Po dalším posunutí je zobrazen přepínač zámku vozidla, který funguje obdobně. Podle stavu přepínače ví uživatel, zda je vozidlo zamčené či odemčené. Po stisknutí přepínače je, jako v případě vrácení, zobrazena *FullScreenActivity* se stručným popisem, který koresponduje se stavem zamknutí. Po úspěšném zamknutí je upraven stav přepínače a stav zamknutí je zapsán do rezervace v SharedPreferences. V případě neúspěchu je o tom uživatel zpraven společně s příčinou a je vrácen zpět do *RideActiveActivity*.



Obrázek 4.4: Posouvatelná obrazovka aktivní jízdy

Obrázek 4.5: Úspěšně ukončená jízda

Obrázek 4.6: Načítací obrazovka

4.5.7 ConfirmationActivity

Tato aktivita je interní třídou Android API a v aplikaci je použita k upozornění uživatele na fakt, že byla zaslána zpráva k otevření mobilní aplikace a danou akci tam tedy musí dokončit. Toto oznámení je uživateli zobrazeno po stisknutí tlačítek u aktivit *UserNotLoggedActivity* a *RideNotActiveActivity* a lze si ho prohlédnout na obrázku 4.7.

Obrázek 4.7: Potvrzení

Obrázek 4.8: Úspěšně ukončená jízda

4.5.8 Dialog

Dialog je interní třída Android API, která v aplikaci nahradila původně zamýšlený *AcceptDenyDialog*. V této třídě lze použit vlastní *layout* soubor, takže

dialog má jednotný design s celou aplikace. Slouží k zobrazení chybových hlášení či upozornění a může mít jedno či dvě tlačítka. Dialog s jedním tlačítkem je zobrazen na obrázku 4.5 a dialog se dvěma tlačítky je na obrázku 4.8.

Kapitola 5

Testovaní

Aplikace byla testována standardními postupy pro testování. Nejprve byly testovány malé části aplikace a následně byly provedeny testy uživatelského rozhraní.

5.1 Jednotkové testovaní

Tato část testů je nejvíce zaměřená na fungování tříd z balíčku *presenter*, které se starají o logiku fungování aplikace. Při testování byl použit framework Mockito [31], který umožňuje vytvářet jednoduché instance tříd takzvané „mock“ objekty. Ty umožňují simulovat funkcionalitu tříd a je možné díky nim snadněji testovat kód.

Mockito ale neumí vytvářet „mock“ objekty, tak aby se daly využívat statické metody tříd. Proto byl také použit framework PowerMock [32], který tuto funkcionalitu umožňuje.

5.2 Automatické testy UI

K automatickému testování uživatelského rozhraní byl použit framework *Espresso* [33], díky kterému lze testovat, zda po akci provedené uživatelem následuje změna v uživatelském rozhraní či zda je spuštěna nějaká aktivita. Toto testování běží buď na fyzickém zařízení nebo na emulátoru.

5.3 Uživatelské testování

Důležitým aspektem testování je také zjistit, jak se používá potencionálním uživatelům. Aplikace je určena zejména pro studenty proto vetší část testerů byla z této skupiny. V testovací skupině byli však i lidé z vyšší věkové skupiny, kteří zastupovali zaměstnance univerzit. Všem testerů, kteří neměli žádné zkušenosti se službou Uniqway, byla nejdříve představena Android aplikace, její

funkce a jak se používá. Testeři si zkusili přihlášení, půjčení a vrácení vozidla či zamykání a odemykání vozidla. Následně přešli k testování aplikace na hodinkách podle následujícího scénáře.

5.3.1 Scénář testování

5.3.1.1 Přihlášení

1. Zapnutí aplikace jako nepřihlášený uživatel.
2. Otevření mobilní aplikace pomocí tlačítka na hodinkách.
3. Přihlášení v mobilní aplikaci.
4. Kontrola přihlášení na hodinkách.

5.3.1.2 Rezervace vozidla

1. Zapnutí aplikace jako přihlášený uživatel.
2. Otevření mobilní aplikace pomocí tlačítka na hodinkách.
3. Rezervace vozidla v mobilní aplikaci.
4. Ověření rezervace na hodinkách.

5.3.1.3 Interakce akcemi rezervace

1. Zapnutí aplikace jako přihlášený uživatel s aktivní rezervací.
2. Posunutí na obrazovku s přepínačem zámku.
3. Odemknutí vozu.
4. Zamknutí vozu.
5. Posunutí na obrazovku s akcemi navigace a vrácení.
6. Otevření navigace.
7. Zavření navigace.
8. Vrácení vozidla.

5.3.2 Výsledky testování

Všichni dotázaní se shodli na tom, že aplikace usnadňuje úkony spojené s aktivní jízdou. Část respondentů navrhla přesunutí přepínače se zámkem před akce navigace a vrácení, jelikož to bude nejčastěji používaná funkce. Respondenti z vyšší věkové skupiny pak doporučili zvětšení písma z důvodu horší čitelnosti.

Na základě těchto výsledků byl přepínač zámku přesunut před tlačítka navigace a vrácení a font byl u všech obrazovek zvětšen.

Závěr

Cílem práce bylo vytvořit aplikaci pro systém sdílení automobilů Uniqway pro operační systém Wear OS. Nejprve byla provedena analýza současné Android aplikace Uniqway na základě které byly stanoveny funkční požadavky. Dále byla prozkoumána dokumentace potřebná pro pochopení a následnou implementaci komunikace mezi hodinkami a mobilním zařízením. Návrhová část práce obsahuje zvolení programovacího jazyka, návrh používání a fungování aplikace, návrh uživatelského rozhraní a v neposlední řadě softwarovou architekturu aplikace. Na základě analýzy a návrhu byla implementována aplikace, která byla následně otestována.

Výsledkem této práce je plně funkční aplikace, která splňuje všechny požadavky stanovené v analytické části práce. Tato aplikace umožňuje uživateli ovládat některé funkce Android aplikace pomocí hodinek bez nutnosti otevíraní mobilní aplikace. Mezi tyto funkce patří zobrazení informací o právě probíhající jízdě, navigace k vozidlu, odemknutí nebo zamknutí vozidla a vrácení vozidla pro ukončení jízdy.

Návrh budoucích zlepšení

Jedním z problému aplikace na hodinkách je nemožnost přerušení zamykacího a odemykacího procesu. Na mobilním zařízení je tento proces svázán s aktivitou a když uživatel tuto aktivity ukončí, tak se ukončí celý proces. U hodinek tento proces probíhá v rámci služby běžící na pozadí v mobilním zařízení. Pokud by tedy uživatel přerušil zamykací, respektive odemykací aktivitu, byla by odeslána zpráva se zrušením do mobilní aplikace. Tato zpráva však spustí novou instanci služby a instance s procesem zamykání tak nemůže být z této nové přerušena. Řešením tohoto problému by mohla být nová zpráva na server, která by pak přerušila komunikaci s první spuštěnou službou tím, že by server vrátil odpověď s příznakem přerušení. Dalším řešením by mohlo být začlenění komunikace hodinek tak, že by každý samostatný dotaz, jestli už

byla přiložena karta nebo vozidlo zamknuto, byl odesílán z hodinek a ne ze služby. Služba by se tak vždy spouštěla jen s jedním dotazem na server.

Dalším rozšířením by mohla být simulace studentské RFID karty přes NFC, které je v mnoha hodinkách přítomno. Tím by uživatel již nemusel vyndávat svou kartu při odemykání vozidla a pohodlně by vše vyřídil přes hodinky.

Aplikace by také mohla být rozšířena o podporu takzvaného *ambient mode*, který umožňuje aplikacím zobrazovat data uživateli, i když jsou hodinky přepnuty do módu šetření baterie.

Bibliografie

1. ŠKODA AUTO DIGILAB S.R.O. *Uniqway* [software]. 2020 [cit. 2020-02-09]. Dostupné z: `https://play.google.com/store/apps/details?id=cz.uniqway.uniqway`.
2. RAVAS, Filip. *Mobilná aplikácia pre užívateľov systému zdielania automobilov viac užívateľmi*. 2017. bathesis. FEL ČVUT. Vedoucí práce doc. ING. DAVID ŠIŠLÁK PH.D.
3. Introduction to Activities. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/guide/components/activities/intro-activities`.
4. Layouts. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/guide/topics/ui/declaring-layout`.
5. Understand the Activity Lifecycle. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/guide/components/activities/activity-lifecycle`.
6. Save key-value data. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/training/data-storage/shared-preferences`.
7. Wear OS overview. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/training/wearables`.
8. Send and sync data on Wear. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/training/wearables/data-layer`.
9. Sync data items on Wear. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: `https://developer.android.com/training/wearables/data-layer/data-items`.

10. GOOGLE, INC. *Android DataLayer Sample* [software]. 2020 [cit. 2020-02-09]. Dostupné z: https://github.com/android/wear-os-samples/tree/master/DataLayer.

11. Send and receive messages on Wear. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: https://developer.android.com/training/wearables/data-layer/messages.

12. Listen for Data Layer events. In: *Android Developers* [online]. 2019 [cit. 2020-02-09]. Dostupné z: https://developer.android.com/training/wearables/data-layer/events#Listen.

13. Modules. In: *Android Developers* [online] [cit. 2020-03-25]. Dostupné z: https://developer.android.com/studio/projects#ApplicationModules.

14. Add a Wear OS module to your project. In: *Android Developers* [online] [cit. 2020-03-25]. Dostupné z: https://developer.android.com/training/wearables/apps/creating#add-wear-os-module.

15. Android Library. In: *Android Developers* [online] [cit. 2020-03-25]. Dostupné z: https://developer.android.com/studio/projects/android-library.

16. Define layouts on Wear. In: *Android Developers* [online] [cit. 2020-02-09]. Dostupné z: https://developer.android.com/training/wearables/ui/layouts.

17. WearableRecyclerView. In: *Android Developers* [online] [cit. 2020-02-09]. Dostupné z: https://developer.android.com/reference/androidx/wear/widget/WearableRecyclerView.

18. THE ANDROID OPEN SOURCE PROJECT. *AcceptDenyDialog* [software]. 2016 [cit. 2020-02-09]. Dostupné z: https://android.googlesource.com/platform/packages/apps/PackageInstaller/+/refs/tags/android-7.1.2_r19/src/android/support/wearable/view/AcceptDenyDialog.java.

19. GOOGLE, INC. *Android WearAccessibilityApp Sample* [software]. 2020 [cit. 2020-02-09]. Dostupné z: https://github.com/android/wear-os-samples/tree/master/WearAccessibilityApp.

20. Introduction to JSON Web Tokens. In: *JWT* [online] [cit. 2020-02-09]. Dostupné z: https://jwt.io/introduction/.

21. *Introducing JSON* [online] [cit. 2020-05-15]. Dostupné z: https://www.json.org/json-en.html.

22. *Gson* [software]. 2020 [cit. 2020-02-09]. Dostupné z: https://github.com/google/gson.

23. Design for Android. In: *Android Developers* [online] [cit. 2020-02-09]. Dostupné z: https://developer.android.com/design.

24. KARPOUZIS, Thanos. Android Architecture. In: *AndroidPub* [online]. 2015 [cit. 2020-02-09]. Dostupné z: https://android.jlelse.eu/android-architecture-2f12e1c7d4db.

25. CHUGH, Anupam. Android MVP. In: *JournalDev* [online] [cit. 2020-02-09]. Dostupné z: https://www.journaldev.com/14886/android-mvp.

26. Android Studio. In: *Android Developers* [online] [cit. 2020-03-09]. Dostupné z: https://developer.android.com/studio.

27. Application. In: *Android Developers* [online] [cit. 2020-03-25]. Dostupné z: https://developer.android.com/reference/android/app/Application.

28. JUNGINGER, Markus. *EventBus* [software]. 2020 [cit. 2020-04-09]. Dostupné z: https://github.com/greenrobot/EventBus.

29. Android Studio. In: *Android Developers* [online] [cit. 2020-03-09]. Dostupné z: https://developer.android.com/training/wearables/data-layer/events#with-a-wearablelistenerservice.

30. IntentService. In: *Android Developers* [online] [cit. 2020-03-25]. Dostupné z: https://developer.android.com/reference/android/app/IntentService.

31. *Mockito framework* [software] [cit. 2020-05-15]. Dostupné z: https://site.mockito.org/.

32. *PowerMock* [software] [cit. 2020-05-17]. Dostupné z: https://github.com/powermock/powermock.

33. Espresso. In: *Android Developers* [online] [cit. 2020-05-15]. Dostupné z: https://developer.android.com/training/testing/espresso.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

GUI Graphical user interface

IDE Integrated Development Environment

NFC Near field communication

REST Representational state transfer

RFID Radio Frequency Identification

RPC Remote Procedure Calls

UI User interface

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
  - `installation_guide.txt` ..... instalační příručka
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `documentation` ..... zdrojová forma dokumantace ve formátu html
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF