UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2018 Pavel Petkevich

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Androidová aplikace pro sestavení plánu a rozpočtu cesty

Petkevich Pavel

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavel Petkevich**
Osobní číslo:	**I14157**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro sestavení plánu a rozpočtu cesty**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je usnadnit plánování cest a také sestavení rozpočtů pro zamýšlené cesty s využitím individuální automobilové dopravy se zaměřením na území Evropské unie. Aplikace bude umožňovat plánování cest se zaměřením na plánování nákladů v podobě dálničních poplatků v jednotlivých zemích na cestě a také provede vyhodnocení nákladů za čerpání paliva s uvažování zadané průměrné spotřeby paliva a odjezdem z výchozího místa s plnou nádrží.
Předpokladem je vytvoření a otestování aplikace pro Android OS.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30-40 normostran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. SIERRA, Kathy. a Bert. BATES. Head first Java. 2nd ed. Sebastopol, CA: O'Reilly, 2005. ISBN 0596009208.
2. CLIFTON I. G. Android User Interface Design: Implementing Material Design for Developers (2nd Edition). Addison-Wesley Professional, 2015. ISBN 978-0134191409.

Vedoucí bakalářské práce: **Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2017**

Termín odevzdání bakalářské práce: **12. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 01. 04. 2018 Pavel Petkevich

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu své práce panu Ing. Michaelu Bažantovi, Ph.D. za cenné rady a velmi přátelský a vstřícný přístup. Dále bych rád poděkoval své rodině. Především své manželce, Anně Petkevich, za každodenní podporu a motivaci.

ANOTACE

Cílem bakalářské práce je usnadnit plánování cest a také sestavení rozpočtů pro zamýšlené cesty s využitím individuální automobilové dopravy se zaměřením na území Evropské unie. Aplikace bude umožňovat plánování cest se zaměřením na plánování nákladů v podobě dálničních poplatků v jednotlivých zemích na cestě a také provede vyhodnocení nákladů za čerpání paliva s uvažování zadané průměrné spotřeby paliva a odjezdem z výchozího místa s plnou nádrží. Předpokladem je vytvoření a otestování aplikace pro Android OS.

KLÍČOVÁ SLOVA

Android OS, Aplikace pro mobilní platformu Android, Databáze, Cestovaní, Cestovaní autem, XML, JSON, Java, Material Design.

TITLE

Application for building and planing budget for a trip.

ANNOTATION

The purpose of this thesis is to make it easier for people to plan automobile trips and to forecast needed budget for a journey in the European Union. The application will allow planning of trips with focusing on forecasting budget that could include such things as highway tolls. Also by using this application, a final user will be able to track and then evaluate costs of fuel consumption during the road based on the entered average fuel consumption, and starting point. To meet facts stated above there is a plan to create and test application for Android OS.

KEYWORDS

Android OS, Android application, Database, Travelling, Travelling by car, XML, JSON, Java, Material Design

SEZNAM ILUSTRACÍ A TABULEK

SEZAM ZKRATEK A ZNAČEK

API	Application Programming Interface
CSS	Cascading Style Sheets
GUI	Graphical User Interface
JDBC	Java Database Connectivity
JEE	Java Enterprise Edition
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MD	Material Design
MVC	Model-view-controller
MVP	Model-view-presenter
NFC	Near Field Communication
ODBC	Open Database Connectivity
OS	Operating System
REST	Representational State Transfer
SQL	Structured Query Language
SQLITE	Structured Query Language Lite
XSQL	Combining XML and SQL
XML	Extensible Markup Language
VDM	Virtual Dalvik Machine

0 ÚVOD

Traap neboli má vlastní aplikace pro mobilní platformu Android, která je zároveň i mojí bakalářskou prací, by měla usnadnit život lidem při sestavování cesty v rámci EU autem. V současné době existují podobné aplikace, a to jak webové, tak mobilní. Po prozkoumání trhu bylo vidět, že tato aplikace ještě neexistuje na trhu v České republice a ani ve většině zemí EU. Zároveň je těžké nalézt aplikaci, která by byla multiplatformní, lehká při používání, uživatelsky přívětivá a za několik dotyků by měla výsledek pro uživatele. Tato aplikace není jednoduchá záležitost. Proto součástí této práce bude i nastínění hlavních problémů spojených s tvorbou aplikace.

Jelikož se nejedná o jednoduchou aplikace a stále pokračuji ve vývoji, stihl jsem vytvořit vlastní algoritmus pro vytvoření a rozpočtování cesty. Rozpočet je v některých částech přibližný, proto při prvním spuštění aplikace bude uživatel upozorněn, že rozpočet cesty není zcela přesný.

1 UVEDENÍ DO PROBLEMATIKY

V současné době se často cestuje po světě. Každý z nás alespoň jednou někam cestoval. Ať už do zahraničí, nebo jen ve své vlastní zemi. Prvním krokem při plánovaní výletu je vybrat cílový bod. Druhým krokem je jakým způsobem se tam dostaneme. Máme různé způsoby dopravy: letadlem, vlakem, autobusem, na kole, pěšky a v neposlední řadě autem. Aplikace v aktuálním stavu řeší jen jeden z těchto způsobů dopravy, a to autem. Tím se můžeme dostat kamkoliv, pokud samozřejmě máme peníze a auto. Ale předtím než někam pojedeme, by většina z nás chtěla vědět kolik by to alespoň přibližně stálo. Přesně tento problém řeší tato aplikace.

1.1 ZPŮSOBY DOPRAVY

Jak již bylo zmíněno, tato aplikace se zabývá jen automobilovou dopravou. Proto bych se v této části podíval na klady a zápory tohoto typu dopravy.

Auto není nejlevnější typ dopravy, což je asi první a poslední zápor. Ale k záporům se ještě dostaneme. Dle vlastních zkušeností považuji za první klad fakt, že se jedná o nejflexibilnější ze všech druhů dopravy. Dalším kladem je nezávislost na nikom jiném a zároveň absence jakýchkoli omezení na velikosti a hmotnosti zavazadla, kromě rozměrů auta. Dle potřeby je možné i za běhu změnit směr cesty, samozřejmě za využití této aplikace.

Teď bych se vrátil k záporům. Nejedná se o úplně levný typ dopravy, jenže například letadlo nebude o moc levnější. Představte si, kolik by Vás stálo proletět 10 různých míst. Vsadil bych se, že auto v tomhle případě vyhraje oproti letadlu v ceně, ale ne v rychlosti. Je dost možné, že se Vám přímo na cestě objeví město, které byste ani náhodou nenašli při cestování letadlem. Další zápor, který bych nemohl vynechat je parkování. V některých městech nebo zemích je velmi drahé parkování nebo dokonce nemožné, protože počet těch aut je větší než kapacity parkovišť.Ale to všechno spadá do jednoho velkého záporu – peníze.

1.2 MÝTNÉ

1.2.1 Co to je mýtné?

Mýtné či mýto je poplatek, který se vybírá za použití cesty, silnice, dálnice, tunelu nebo mostu.

V každé zemi jsou ohledně těchto poplatků různá pravidla. Můžeme platit za ujeté kilometry, nebo za období, po kterém můžeme danou komunikaci využívat (například denní, týdenní, desetidenní nebo roční). Určitě nezapomeneme na průjezd tunelem a přejezd přes most.

Některé věci v této práci budou vyřešeny jinak, než v distribuované verzi. Například poplatek za mosty a tunely nepočítám, jelikož k počítání je potřeba připojení aplikace k placenému API a čerpat z něj data.

V některých zemích tento poplatek vůbec nemají, například v Německu, Andoře a Finsku.

Dokonce jsou i takové země, ve kterých se můžete obejít bez poplatku za podmínky, že v zemi budete maximálně jeden den. To platí například v Bulharsku a Rumunsku.

Mýtné je přímo závislé na váze Vašeho auta. Tato aplikace momentálně řeší problém jenom pro auta do 3,5 tuny. Proto se nezabývám cenami pro větší auta.

Konkrétní řešení vysvětlím až v praktické časti.

1.3 PALIVO

Palivo je asi nejhlavnější bod, bez něho auto nepojede. Máme různé druhy paliv. V současné době jsou stále nejpoužívanější benzín a nafta, za kterými pokulhávají plyny (LPG a CNG). To se ale podle mého názoru brzy změní na jiný typ, a sice na elektřinu. Tato práce se zabývá pouze benzínem a naftou. Ceny paliv se v jednotlivých zemích liší, proto aplikace počítá s průměrnou cenou dle země.

1.3.1 SPOTŘEBA

Průměrnou spotřebu si uživatel uvede sám. Jedná se o průměrnou spotřeba paliva v litrech na 100 kilometrů. Uživatel tento údaj může dohledat buď ve velkém technickém průkazu nebo v palubním počítači auta (pokud auto tento počítač má). Dokonce zkušenější řidiči znají průměrnou spotřebu svého auta, nebo jsou schopni ji odhadnout.

Spotřeba je různá, jelikož je závislá na mnoha faktorech. Mezi tyto patří například: počasí, tlak v pneumatikách, kvalita cesty, stáří auta, typ motoru, síla motoru, váha auta a styl jízdy řidiče.

1.4 APLIKACE PRO OPERAČNÍ SYSTÉM ANDROID

Začneme asi pojmem Android, který označuje mobilní operační systém založený na jádře Linuxu, který je dostupný jako otevřený software. Je používán na chytrých telefonech, tabletech a mnohých dalších zařízeních.

Vývojem operačního systému Android se zabývá jedna z nejznámějších firem ve světě - Google. Zajímavostí je, že Google nezakazuje výrobcům telefonů upravovat operační systém, ovšem pouze za dodržení smluvních podmínek.

Android jako operační systém má největší podíl prodaných telefonu na trhu po celém světě [1].

Proč zrovna androidová aplikace?

Mobilní technologie se stále rozvíjejí. Prakticky každý z nás už má chytrý telefon ve své kapse. Nechtěl bych se tady věnovat porovnání Android App vs. Mobile Web, na tohle budu mít vyhrazenou část. Zkráceně bych řekl, že oproti Mobile Web, androidová aplikace určitě vyhrává.

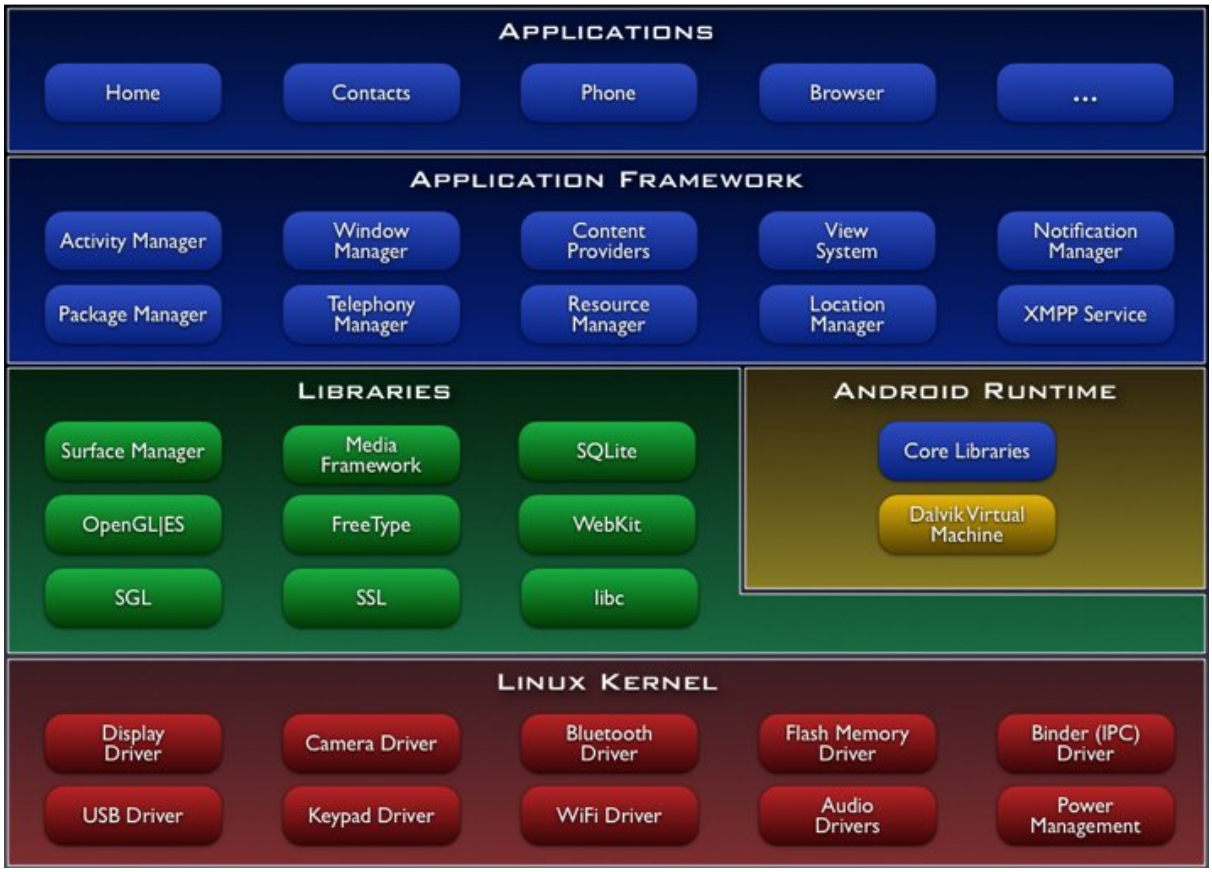
Tyto telefony pracují na nějakém operačním systému, kterých je hodně. Nejpoužívanější však jsou Android, IOS a Windows. Rozdíly těch systémů budou popsány v jiné kapitole. Většinou se potkáváme buď s Android nebo IOS, proto bude jedna kapitola o porovnaní těchto dvou operačních systému.

Začal jsem vytvářet tuto aplikace pro Android OS, protože většina lidi využívá tento operační systém. Androidová aplikace obsahuje takové benefity, jako například ukládaní naplánované cesty do historie (což webová aplikace také umí, ale přístup k této historii bez připojení k internetu udělá jen mobilní aplikace), nebo responzivní design (mobilní webová aplikace je něco jiného, než design v mobilních aplikacích).

2 ANDROID DO HLOUBKY

2.1 Androidová architektura

Architektura androidu je rozdělená do pěti vrstev. Každá vrstva má svůj účel a nemusí být přímo oddělena od ostatních vrstev. Níže je uveden Obrázek 1 pomocí kterého lze androidovou architekturu pochopit rychleji. Vrstvy budou popsány od nejnižší.



Obrázek 1 – Architektura Android. Zdroj: [2].

2.1.1 Linux kernel

Linux kernel je nejnižší vrstva ze všech. Jádro přímo komunikuje s hardwarem zařízení. Tato vrstva provozuje základní systémové funkcionality: správu procesů a paměti, ovladače, síťovou komunikaci, kamery, klávesnici, zobrazení a tak dále. Z důvodu spuštění aplikace a služby v oddělených procesech se používá Binder pro zajištění komunikace mezi nimi. Binder mapuje a počítá veškerou komunikaci a zabezpečuje přístupy mezi nimi [3].

2.1.2 Libraries (knihovny)

Je to takzvaná mezivrstva mezi jádrem (Linux kernel) a vyššími vrstvami. Knihovny jsou napsány v programovacích jazycích jako C a C++. Projedeme alespoň pár knihoven [3].

Surface Manager

Zabezpečuje funkcionalitu multidotykového displeje a směruje grafický výstup z více aplikací do souvislého toku, který je předán grafické vykreslovací paměti [3].

SQLite

Označuje vnitřní databázi androidu, která se běžně využívá při vývoji. SQLite je lehká při využití, nezabírá moc místa a zároveň umožňuje velmi rychlou komunikaci s aplikací oproti MS SQL databázi, jelikož se nepotřebuje obracet na server (veškerá data jsou na lokálu). Posledním velkým bonusem u SQLite je absence nutnosti internetového připojení. Za pomocí SQLite lze postavit libovolnou databázi, pouze u obtížnějších aplikací bychom měli využit MS SQ, ale dle mých zkušenosti ve většině případu SQLite stačí [4].

WebKit

Vykresluje a zobrazuje webové stránky [3].

2.1.3 Android Runtime

Třetí vrstva androidové architektury, která je přístupná z druhé vrstvy. Tato vrstva obsahuje sadu základních knihoven.

DVM – je něco podobného jako JVM (Java Virtual Machine), akorát pro android. Při překladu aplikace napsané v jazyce Java/Kotlin kompilátor přeloží zdrojové kódy do binárních souborů, a ty, spolu s jinými zdroji, která aplikace využívá, transformuje za pomocí DX do jednoho souboru ve formátu DEX. Až pak virtuální stroj začne tento soubor vykonávat [3].

2.1.4 Application Framework (Aplikační Framework)

Tento Framework je celý napsán v jazyce Java, je to nejdůležitější vrstva pro vývojáře androidových aplikací. Tato vrstva poskytuje aplikacím spoustu různých systémových služeb z vyšších úrovní [3].

1. **Activity Manager**

   Tento modul se zabývá aktivitou, sleduje informace o aktivitě a životní cyklus [2].

2. **Fragment Manager**

   Rozhraní pro interakci s objekty Fragmentu uvnitř aktivity. Takže tento modul sleduje životní cyklus fragmentu [2].

3. **Package Manager**

   Modul pro spravování balíčků, který udržuje aktuální seznam všech nainstalovaných aplikací. Uživatel jej vidí jako pracovní plochu s ikony, kde každá ikona aplikace reprezentuje jeden balíček [2].

4. **Window Manager**

   Tento modul se zabývá spravováním oken tvořících aplikaci. Většina aplikací současně může využívat dvě a více oken. Vyskakovací dialogové nabídky jsou samostatná okna, která vyvolává aplikace [2].

5. **View System**

   Spravuje uživatelské rozhraní – texty, ikony, editory textu, tlačítka, spinnery a tak dále [2].

2.1.5 Application (Aplikace)

Jedná se o nejvyšší vrstvu androidové architektury. Najdeme tu celou aplikaci pro Android, kterou zapíšeme pouze na tuto vrstvu. Příklady takových aplikací jsou seznam kontaktů, webový prohlížeč, pošta (email), zprávy a jiné. Pro přístup aplikací k některým komponentám a údajům, potřebuje aplikace mít potřebné oprávnění (permission). Oprávnění jsou zavedeny pro to, aby aplikace bez povolení uživatele nemohla využívat například polohu nebo seznam kontaktů. Těchto oprávnění je spousta. Podrobněji se o tom po bavíme až u praktické časti. Veškerá oprávnění musí být deklarována v manifestu [2] [3].

Manifest (AndroidManifest.xml) – je soubor, který popisuje veškery prostředky, které aplikace bude využívat [2].

2.2 Historie verzí Android

Historie mobilního operačního systému Android započala v roce 2003, tehdy vlastněné firmou Android Inc. Ale v roce 2005 firma Google koupila firmu Android Inc. A už roku 2007 byl vydán první mobilní operační systém Android – Android Alpha, trochu později následoval Android Beta. Ale tyhle dvě verze byly nekomerční verzí operačního sytému Android. Později v roce 2008 byla vydána první komerční verze Android OS, tzv. Android 1.0 [3].

Obrázek 2 – Verzí Android platforem. Zdroj: [2].

Android 1.5 Cupcake

Verze byla představená v dubnu 2009 a podporovala virtuální klávesnice třetích stran se slovníkem, animací přechodů a nabízela možnost widgetů. Nahrává a přehrává videa ve formátu MPEG-4 a 3GP [2].

Android 1.6 Donut

Kobliha byla vypuštěna do světa v září 2009, tedy jen několik měsíců po spuštění Muffinu. Verze dostala vyhledávaní Google, bezplatnou navigaci od Google, vylepšil se market s aplikacemi, VPN připojení a podporování displejů s větším rozlišením [2].

Android 2.0 – 2.1 Eclair

Krémový rohlíček byl přípraven v roce 2010. Android dostal nový design, optimalizaci výkonu, podporu Bluetooth 2.1, Google Mapy a živé tapety [2].

Android 2.2 – 2.2.3 Froyo

Šlehačková špička přináší rychlost práce. Objevila se nová funkce hlasového ovládaní telefonu a možnost vytvoření WI-FI hot-spotu. Za pomocí hlasu mohl uživatel vytvořit poznámku, nastavit budík a mnoho dalších úkolů. Virtuální mašina Davlik (VDM) dostala dynamický kompilátor, což zrychlilo kompilace kódu až pětkrát oproti minulým verzím [2].

Android 2.3 – 2.3.7 Gingerbread

Až na konci roku 2010 se objevil perníček. Na tuto verzi čekali především vývojáři her pro Android OS. Nově mohli vytvářet dynamické hry s výbornou 3D grafikou. Ale Google nezapomněl ani na uživatele Android OS, proto přidal podporu NFC, která dovoluje uživateli předávat informace z jednoho zařízení na jiné za pár vteřin a provádět platby. Za několik let tato technologie bude základem pro Android Pay [2].

Android 3.0 – 3.2.6 Honeycomb

S touto verzí uživatelského rozhraní se tablet stal takovým, jakým ho známe dnes. Velmi vítaným způsobem byl přizpůsoben k rychlé navigaci a práci s velkými obrázky [2].

Android 4.0 – 4.0.4 Ice Cream Sandwich

Ruská zmrzlina přináší uživatelům vytváření složek a widgetů na hlavní ploše. Ještě se nám objevila kontrola trafiku mobilních dát. Nově si uživatel může zapnout statistiku za období, navíc si může přidat limit na spotřebu mobilních dát a jakmile spotřeba dosáhne limitované hranice, uživatel bude informován. Tato verze obsahuje takzvaný Android Beam – je to technologie, která umožňuje uživateli za pomoci NFC rychle přehrávat jak aplikaci, tak i hudbu, videa a tak dále [2].

Android 4.1 – 4.3.1 Jelly Bean

Tato verze přidala interaktivní notifikace, za pomocí kterých uživateli stačí jen klepnout na notifikace a bude nasměrován tam odkud je notifikace původem. Nyní uživatel může využívat několik různých účtů [2].

Android 4.4 – 4.4.4 KitKat

Zlepšilo se hlasové ovládaní telefonu. Za pomocí nového příkazu „Ok, Google“ se uživatelé mohou rychleji dozvědět nějakou informace, posílat někomu zprávy a ovládat jinou funkcionalitou android telefonu [2].

Android 5.0 – 5.1.1 Lollipop

Od hodinek a tabletu se přesouváme k televizím a autům. Tahle verze se liší od ostatních verzí prostým a barevným uživatelským prostředím s využitím Material Design [2].

Android 6.0 – 6.0.1 Marshmallow

Největší přínos této verze je ekonomie baterie pro nejdůležitější úkoly. Ještě přináší podporu USB typu C a rozpoznávání otisku prstů [2].

Android 7.0 – 7.1.1 Nugat

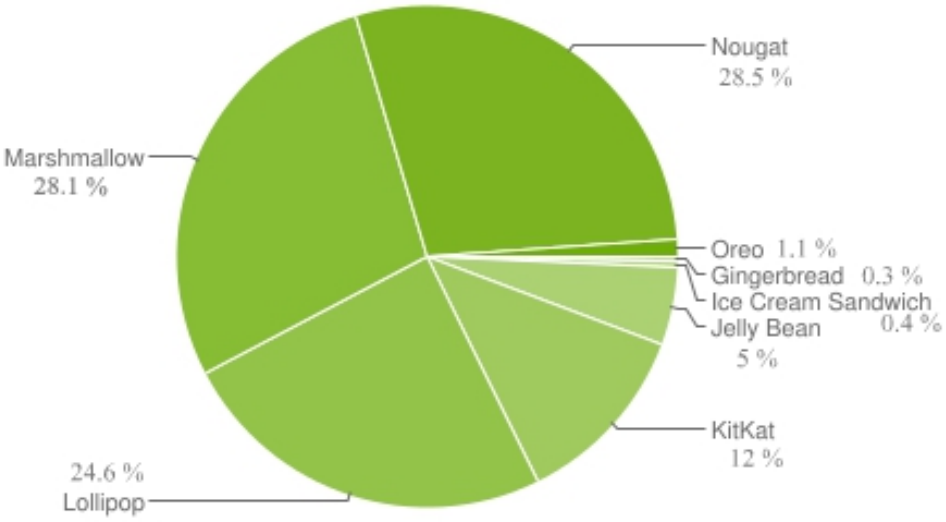
Před dvěma lety se nám objevila verze Nugát, která nám přinesla podporu API pro virtuální realitu, možnost práce s více okny, rychlé přepínaní mezi aplikacemi a vlastní uspořádaní dlaždic v rychlém nastavení [2].

Android 8.0 Oreo

Chytrý, rychlý, silný a sladší než všichni předchozí verze operačního systému android. Co je nového: znovu zlepšené notifikace, jak designově, tak i funkčně, obraz v obraze jako u televize, adaptivní ikony v celém systému, výběr textu s umělou inteligencí a lepší výdrž baterie [2].

2.2.1 Nejpoužívanější verze platformy android

Dle statistik Google k roku 2018, lidé nejvíc využívají verze Maršmeloun.



Obrázek 3 – Zastoupení verzí platformy android na trhu. Zdroj: [2].

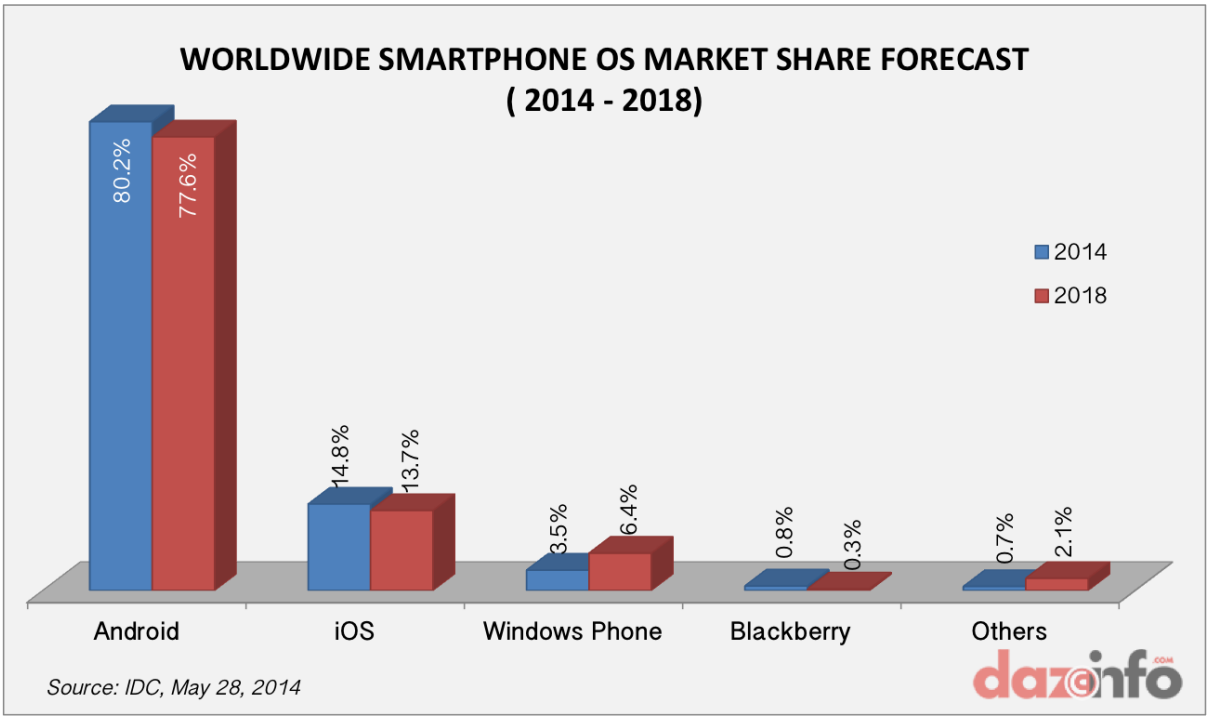
3 Proč jsem si zvolil Android aplikace oproti IOS?

Osobně mám raději práci s Apple technikou, a to nekončí jen IPhonem. Všechny počítače, telefony, chytré hodinky, tablety a televizi mám od Apple, zatímco jsem vývojář pro platformu Android.

Těch důvodů, proč jsem si vybral Android, je dost, ale nechtěl bych procházet všechny, proto bych uvedl jen pár základních důvodů, za pomocí nichž jsem se rozhodoval:

1. Podnikatelský pohled na využití platformy Android oproti IOS. K roku 2018 používá platformu Android více než 77.6 % lidí v celém světě a jenom 13.7 % lidí využívá platformu IOS. Níže je uveden Obrázek 4, na kterém je přesně vidět kolik lidí co využívá.

2. Začínal jsem s programovacím jazykem Java, proto jsem v ní měl více zkušeností než ve Swiftu. Navíc když jsem začínal s Androidem, IOS se vyvíjel v Objective C, který je o moc komplikovanější než Java.



Obrázek 4 - Celosvětová prognóza podílů OS smartphonů na trhu. Zdroj: [10].

4 Technologie pro tvorbu androidové aplikace

4.1 Použitá technologie

4.1.1 Java

V roce 1995 byl světu představen objektově orientovaný programovací jazyk – Java, který byl vyvinutý firmou Sun Microsystems. Tato firma zanikla v roce 2010, poté, co ji koupil Oracle. Java je nejpoužívanější programovací jazyk na světě [8].

1. **Základní vlastnosti Javy:**
    a. Jednoduchá syntaxe oproti C a C++ [5].
    b. Objektově orientovaný přístup s výjimkou osmi jednoduchých datových typů (boolean, char, byte, short, int, long, float, double) [5].
    c. Distribuovaný (umožňuje vytvářet distribuované klientské aplikace) [5].
    d. Interpretovaný – místo kódu se vytváří takzvaný bajtkód, který je nezávislý na architektuře počítače. Program pak může pracovat na libovolném počítači, který má interpret Javy (JVM) [5].
    e. Automatická správa paměti – správa paměti je realizovaná pomocí automatického garbage collectoru, který automaticky vyhledává již nepoužívané časti paměti a uvolňuje je pro opětovné použití [5].
    f. Bezpečný – chrání počítač v síťovém prostředí [5].
    g. Nezávislý na architektuře a přenositelný mezi různé druhy platformy v Javě. [5].
    h. Výkonný – překladače mohou pracovat v režimu Just In Time (JIT) a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí [5].
    i. Podporuje zpracování vícevláknových aplikací [5].
    j. Dynamický a elegantní [5].

Tento programovací jazyk jde použít všude: v čipových kartách (JavaCard), v mobilním vývoji (Java ME), v desktopových aplikací pro počítače (Java SE), tedy pro podnikové aplikace i informační systémy (Java EE) [6].

2. **Javové platformy**
    a. JavaCard – umožňuje bezpečný běh aplikací nazývaných applety (softwarová komponenta, která běží v kontextu jiného programu a nejčeštěji je orientovaná na plnění konkrétní funkce). JavaCard je využívána v SIM pro mobilní telefony a ATM kartách [6].

b. Java ME – tato platforma je podmnožina platformy Javy SE a je orientovaná na malá zařízení, jako jsou mobilní telefony. Java ME jsou často klienty služeb Java EE [6].
c. Java SE – je to vlastně Java, tak jak byla vyvíjena od první verze a postupně se rozšiřovala. Tato platforma poskytuje základní funkce programovacího jazyka Java až po třídy vysoké úrovně, které se používají pro vytváření sítě, zabezpečení, přístupu k databází, vývoj GUI a analýzu XML [6].
d. Java EE – tato platforma je postavena na Java SE, poskytuje rozhraní API a runtime pro vývoj a provoz rozsáhlých, vícevrstvých, spolehlivých a zabezpečených síťových aplikací [6].
e. Java FX – je platforma pro vytváření bohatých internetových aplikací. Java FX Script je staticky napsaný deklarující jazyk, který je kompilován s Java technologií bajtkód, který pak může být spuštěn za pomocí JVM [6].

4.1.2 XML

XML (Extensible Markup Language) je rozšiřitelný značkovací jazyk podobně jako HTML nebo JSON, který se používá k popisu dat. Samotný XML je dobře čitelný jak člověkem, tak i strojem. V Androidu používáme XML pro návrh našich rozvržení, ukládání dat, ukládání zdrojů (res/color nebo res/string) a tak dále [5] [6].

4.1.3 JSON

JavaScript Object Notation (JSON) je způsob zápisu dat nezávislý na platformě, určený pro přenos dat, která mohou být organizovaná jak v polích, tak i v objektech. JSON je využívaný prakticky všude. Google dokonce ve svých API využívá jak JSON, tak i XML. Ale doporučuji k použiti JSON místo XML. Například pří posílaní požadavku na Google Maps API, se nám vrátí JSON soubor, dle kterého bychom měli správně vytvořit strukturu objektu a teprve s ním začít něco dělat. Pro vytvoření takových struktur existuje hodně různých nástrojů, jako třeba JsonSchemaPojo[1], ale pouze pokud se orientujete ve struktuře JSON a víte jak z něj navrhnout strukturu objektů pro Javu, což by většina z nás měla umět. Rychlejší a lepší pro vývojáře bude navrhnout vlastní strukturu, než pak probírat a odstraňovat nepotřebné věci po vygenerování a nejhlavnější, pokud to udělá sám pak to pochopí mnohem lépe [6].

[1] http://www.jsonschema2pojo.org

4.1.4 API

V informatice je chápáno jako rozhraní pro programovací aplikací. Jak funguje API? Je to buď externí stránka nebo aplikace, se kterou se chcete propojit, poskytující kompletní seznam funkcí a popis dat, s nímž lze pracovat. S API většinou se komunikuje za pomocí speciálních formátů pro výměnu dat jako XML nebo JSON. Google nám poskytuje hodně různých druhů API: Google Maps Android API, Google Maps Directions API, Google Places API, Google Drive API a tak dále. Je jich opravdu mnoho. Využití API v mé aplikaci se budeme věnovat v praktické části [5].

4.1.5 Material Design

Material design byl zaveden firmou Google a představen na Google I/O v roce 2014. Je to standartní design, které by vývojáři měli dodržovat. MD obsahuje víc animací a jejich hlavní vizitkou byly karty (cards) a využití efektů stínu. Tento design byl představen pro systémy Android i Chrome OS [11] [7].

Zásady Material Design:

1. **Material je metafora:** MD je založen na hratelné realitě, inspirovaný naší studií papíru a inkoustu, který je však otevřen fantazii a kouzlu [11] [7].
2. **Povrchy jsou intuitivní a přirozené:** Povrchy a hrany poskytují vizuální prvky, které jsou zakotveny v našich zkušenostech s realitou. Použití známých hmatových atributů hovoří o primárních částech našeho mozku a pomáhá nám rychle porozumět dostupnostem [11] [7].
3. **Rozměrnost umožňuje interakci:** Základy světla, povrchu a pohybu jsou klíčem k informování o tom, jak objekty interagují. Realistické osvětlení ukazuje švy, dělí prostor a označuje pohyblivé části [11] [7].
4. **Jeden adaptivní design:** Jeden základní systém designu organizuje interakce a prostor. Každé zařízení odráží jiný pohled na stejný podkladový systém. Každý pohled je přizpůsoben velikosti a interakci vhodné pro toto zařízení. Barvy, ikonografie, hierarchie a prostorové vztahy zůstávají konstantní [11] [7].
5. **Obsah je tučný, grafický a záměrný:** Tučná konstrukce vytváří hierarchii, význam a zaměření. Předběžná volba barev, snímky od hrany k okrajům, velká typografie a záměrný bílý prostor vytvářejí ponoření a jasnost [11] [7].

6. **Barva, povrch a ikonografie zdůrazňují akce:** Uživatelská akce je podstatou návrhu zážitku. Primární akce jsou inflexní body, které transformují celý návrh. Jejich důraz zjednodušuje základní funkcionalitu a poskytuje uživatelům body [11] [7].
7. **Uživatel iniciuje změnu:** Změny v rozhraní odvozují svou energii z akcí uživatele. Pohyb, který kaskáduje od dotyku, respektuje a posiluje uživatele jako hlavní hnací sílu [11] [7].
8. **Animace je choreografována na společné scéně:** Všechna akce probíhají v jediném prostředí. Objekty jsou prezentovány uživateli bez přerušení nepřetržité zkušenosti, a to i při transformaci a reorganizaci [11] [7].
9. **Pohyb poskytuje význam:** Pohyb je smysluplný a vhodný, sloužící k soustředění pozornosti a udržení kontinuity. Zpětná vazba je jemná, ale jasná. Přechody jsou účinné, avšak souvislé [11] [7].

4.2 Alternativní technologie

4.2.1 Kotlin

Kotlin je zcela nový programovací jazyk, který se nám poprvé objevil v roce 2011, ale verze 1.0 Kotlinu byla představena až roku 2016, tedy před dvěma lety. Kotlin běží nad JVM, jeho hlavním vývojářem je tým programátorů JetBrains. Podle vedoucího vývoje Andreye Breslava je Kotlin navržen jako průmyslově spolehlivý objektově orientovaný programovací jazyk, který je lepší než Java. Pro lidi, kteří znají Javu a C# není žádný problém začít pracovat v Kotlinu. Větší podobu bychom mohli najit mezi Kotlin a C# / Scala, podobně jako ve Scale středníky jsou volitelné a používají se jen v případech, kdy jsou reálně zapotřebí [2].

Začátkem února roku 2018 se svět androidového vývoje v jazyce Kotlin znovu zlepšil. Google 5.02.2018 nám představil Android KTX, což je sada rozšíření pro Kotlin pro vývoj aplikací pod Android. Tato knihovna vývojářům přináší do projektu – méně kódu, více zábavy a jednoduššího pochopeni projektu [2].

Dále uvádím Obrázek 5, který jasně ukazuje, jak KTX zlepšuje náš kód.

Kotlin	Kotlin with Android KTX

`sharedPreferences.edit()`
`  .putBoolean(key, value)`
`  .apply()`
	`sharedPreferences.edit {`
	`  putBoolean(key, value)`
	`}`

Obrázek 5 – Před a po využiti Kotlin extensions. Zdroj: [2].

4.2.2 Xamarin

Xamarin je multiplatformní Framework, který umožnuje vytvářet aplikace jak pro Android, tak pro IOS a Windows. Používáme stejný jazyk, API a datové struktury pro všechny 3 platformy, pouze je potřeba mít rozdílné uživatelské rozhraní pro každou platformu, proto kryjeme jenom 75 % kódu, dle potřeby můžeme vytvořit uživatelské rozhraní s formáty Xamarin.Forms a sdílíme téměř 100 %. Xamarin byl založen na open-source projektu Mono. Mono je založen na ECMA standardech pro C# a Common Language Infrastructure a umožňuje psaní crossplatformních aplikací [9].

4.3 Shrnutí

K výše zmíněnému popisu všech použitých technologií pro tvorbu Androidové mobilní aplikace je zjevné, že i v alternativních technologiích je potřeba mít alespoň základy těch, které jsem používal.

Vzhledem k tomu, že jsem jeden z těch vývojářů, který v dnešní době stálé programuje pro každou platformu jejím nativním jazykem, nevybral jsem si Xamarin místo Javy. Kotlin jsem si nevybral proto, že produkovanou verzi této aplikace pro podnikání budu mít v Kotlinu a proto jsem si vybral Javu pro bakalářskou práce.

5 Technologie pro návrh a tvorbu databáze pro platformu android

Uchovat nějaké maličkosti můžeme i v Shared Preferences, JSON, XML a dokonce i v CSV, ale pro trochu složitější případy bychom mohli využit databázi. Android vývojáři většinou využívají SQLite pro dlouhodobé uchovávaní dat v aplikaci.

5.1 Použitá technologie

5.1.1 SQLite

SQLite je relační databázový systém obsažený v relativně malé knihovně napsané v jazyce C. Na rozdíl od databází založených na principu klient-server, kde je databázový server spuštěn jako samostatný proces, je SQLite pouze nevelká knihovna, která, po přilinkování k aplikaci, je k dispozice pomocí jednoduchého rozhraní. Každá databáze je uložena v samostatném souboru *.dbm [11].

SQLite Database obsahuje základní metody: create (vytvoř), delete (vymaž) a execute SQL commands (proveď SQL příkaz) [11].

Pro přistup k SQLite databázi není potřeba vytvářet žádné připojení, jako JDBC, ODBC a tak dále [11].

Oproti databázím klient-server má jak výhody, tak i nevýhody [12].

Jaké jsou výhody:

1. **Lepší výkon:** Čtení dat z SQLite databáze je mnohem rychlejší, než ze serveru [12].
2. **Snížené náklady na aplikaci a složitost:** Není potřeba pronajímat místo na serveru, starat se o server a tak dále [11].
3. **Přenosnost:** Aplikační soubor je přenosný ve všech operačních systémech, 32bitové i 64bitové architektuře [11].
4. **Spolehlivost:** Obsah může být aktualizován nepřetržitě a atomicky tak, aby při ztrátě nebo zhroucení elektrické energie nedošlo k žádné ztrátě práce [11].
5. **Přístupnost:** Databázový obsah SQLite lze prohlížet pomocí široké škály nástrojů třetích stran [11].

Určitě jsou zde i nevýhody:

1. SQLite se používá pro zpracování žádostí HTTP s nízkou až střední návštěvností.
2. Velikost databáze je ve většině případů omezena na 2 GB.

Ale z osobních zkušeností pro androidovou aplikaci ve většině případech SQLite bohatě vystačuje.

5.2 Alternativní technologie

Někdy se stane, že nám SQLite nestačí, proto můžeme využit alternativní technologie pro návrh a tvorbu databáze.

Pro začátek bych chtěl uvést příklad, kde by nám mohlo nestačit SQLite. Ve druhém ročníku jsem dostal nápad udělat Skladovací systém pro maloobchodní a velkoobchodní firmy. Skladovací systém byl postaven na Oracle MS SQL z důvodu jeho náročnosti, a zřejmě bylo pochopitelné, že SQLite není pro tento případ vhodná volba. Androidová aplikace by měla komunikovat s MS SQL databází, která leží někde na serveru. A pro ukládání stavu aplikace bychom mohli využit jak Shared Preferences tak i SQLite databázi.

5.2.1 Oracle MS SQL

Oracle je největší a nejznámější firma ve světě, která poskytuje databázový server pro strukturovanou správu dat. V multiuživatelském prostředí může více uživatelů přistupovat ke stejným datům. Databázový server je zabezpečený proti neautorizovaným přístupům. Navíc je multiplatformní s velice pokročilými možnosti zpracování dat a vysokým výkonem. Proto je databázový server Oracle velice výkonný oproti jiným serverům. Oracle databáze je soubor dat uvnitř databázového serveru [13].

Pro komunikaci aplikace s databázovým serverem se využívá standardní API ODBC (Open Database Communication) [13].

Systém Oracle podporuje nejen standardní relační dotazovací jazyk SQL, ale také i rozšíření Oracle: hierarchické dotazy, PL/SQL rozšiřující možnosti SQL, dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat (XML databáze, jazyk XSQL, JSON) [13].

1. **SQL** – je nástroj pro organizování a manipulaci s daty v databázi. SQL je adaptovatelný pro jakékoliv prostředí [14].

   SQL není pouze dotazovací jazyk, pomocí něhož můžeme definovat data, strukturu tabulky, naplňovat sloupce tabulky, daty a definovat vztahy mezi položkami dat. Můžeme pomocí něj rovněž řídit přístup k datům pomocí různých úrovních přístupových oprávnění, čímž zachráníme data před náhodným nebo úmyslným zničením nebo neautorizovaným čtením a manipulaci s nimi [13].

   Nemůžeme říct, že SQL je plnohodnotný samostatný programovací jazyk, protože se v jeho implementaci nenachází řídící programové konstrukce, které by měl obsahovat každý programovací jazyk [13].

   SQL je interaktivní dotazovací jazyk, který umožňuje získat odpovědi i na velmi komplikované dotazy. Je srozumitelný, protože chápe data v podobě tabulek, což je snadno pochopitelné i uživatelům [14].

   Tabulka je množina dat, která je uspořádaná v řádcích (záznamech) a sloupcích (položkách). Uživatel se odkazuje na hodnotu dat jako na prvek v matici [13].

2. **PL/SQL** – je procedurální rozšíření jazyka SQL implementovaný v databázovém systému Oracle. Díky němu je možné napsat část logiky aplikace přímo v databázovým systému, čímž jak zmenšíme kód, tak i zrychlíme aplikaci. PL/SQL nám umožňuje vytvářet [15].

   a. **Procedury** – skupina příkazů sdružených pod jedním názvem, může mít vstupní a výstupní parametry a dovoluje měnit tabulky (přidávat, odebírat a mazat). Procedury jsou vhodné pří provedení více příkazů najednou v případech, které se často opakuji [15].

```
create or replace PROCEDURE aktualizaceSKJ(
  OPERACE IN VARCHAR2,
  maxKAP IN NUMERIC,
  nazevSKSJ IN VARCHAR2,
  skladID IN NUMERIC,
  idSKJ IN NUMERIC)
IS
BEGIN
  IF OPERACE = 'I' THEN
      INSERT INTO SKLADOVACI_JEDNOTKA(ID_S_J,MAX_SJ_KAP,NAZEV_S_J,ID_SKLAD) VALUES(SEQ_Skladovaci_Jednotka.NEXTVAL, maxKAP, nazevSKSJ, skladID);
    ELSE IF  OPERACE = 'U' THEN
       UPDATE SKLADOVACI_JEDNOTKA SET MAX_SJ_KAP = maxKAP, NAZEV_S_J = nazevSKSJ, ID_SKLAD = skladID WHERE ID_S_J = idSKJ;
      ELSE IF OPERACE = 'D' THEN
         DELETE FROM SKLADOVACI_JEDNOTKA WHERE ID_S_J = idSKJ;
      END IF;
    END IF;
  END IF;
END aktualizaceSKJ;
```

Obrázek 6 - Ukázka procedury z mého vlastního projektu.

b. **Funkce** – skupina příkazů sdružených pod jedním názvem, nedovoluje měnit tabulky, vrací jenom jednu hodnotu a vyplatí se používat ve stejných případech jako u procedur [15].

```
create or replace function SUM_PRIJMU_VYDEJKY
(CENA IN VARCHAR2, nazevT IN VARCHAR2)
return number
as
summa number;
begin
EXECUTE IMMEDIATE 'SELECT SUM('||CENA||') FROM '|| nazevT INTO summa;
return summa;
end SUM_PRIJMU_VYDEJKY;
```

Obrázek 7 - Ukázka funkce z mého vlastního projektu.

c. **Triggery** - skupina příkazů, které jsou zavolány buď před určitou akcí nebo těsně po ní na konkrétní tabulce. Triggery mohou měnit tabulky jako i procedury, vhodné k použití místo procedur nebo funkcí u akcí, které jsou přímo spojené se změnou v tabulce [15].

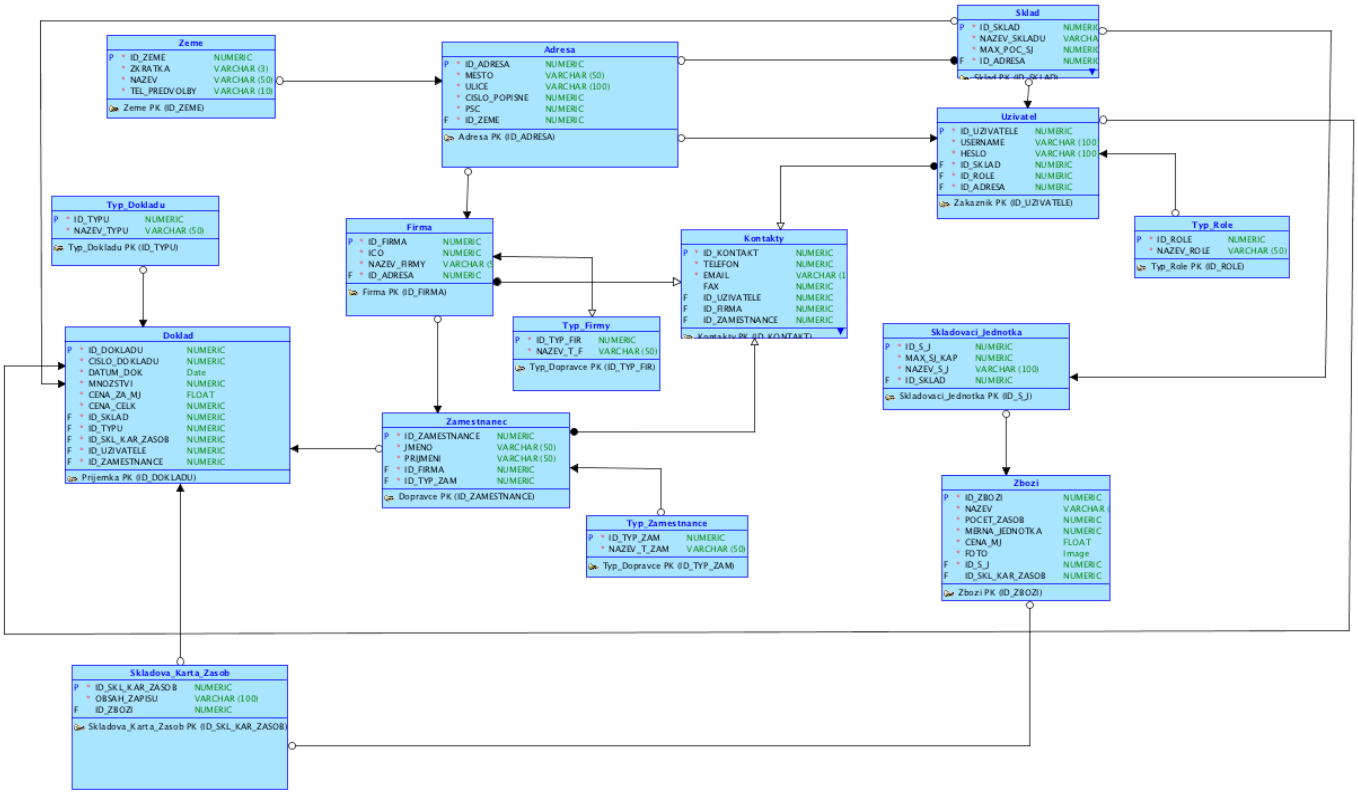
```
CREATE SEQUENCE SEQ_ADMIN_CONTROL START WITH 1 INCREMENT BY 1;

execute reset_seq('SEQ_ADMIN_CONTROL');

CREATE TABLE ADMIN_CONTROL(
  ID_ZPRAVY NUMERIC NOT NULL,
  OBSAH_ZPRAVY NVARCHAR2(50),
  USER__DB NVARCHAR2(50),
  DATE_UPD DATE,
  OPERATION NVARCHAR2(50),
  OLD_USERNAME NVARCHAR2(50),
  NEW_USERNAME NVARCHAR2(50),
  OLD_HESLO NVARCHAR2(50),
  NEW_HESLO NVARCHAR2(50),
  PRIMARY KEY(ID_ZPRAVY)
);
/
CREATE OR REPLACE TRIGGER uzvatele_username_last
  BEFORE UPDATE ON UZIVATEL
  FOR EACH ROW
BEGIN
  INSERT INTO ADMIN_CONTROL(ID_ZPRAVY,OBSAH_ZPRAVY,USER__DB, DATE_UPD,OPERATION,OLD_USERNAME,NEW_USERNAME,OLD_HESLO,NEW_HESLO)
    VALUES(SEQ_ADMIN_CONTROL.NEXTVAL, 'Update username, by user:' ,USER, SYSDATE, 'UPDATE',
    nvl(:old.username, 'null'), nvl(:new.username,'null'),nvl(:old.heslo, 'null'), nvl(:new.heslo,'null'));
END;
```

Obrázek 8 - Ukázka triggeru z mého vlastního projektu

Oracle poskytuje vývojářům databázových systémů vlastní aplikace, které jsou velmi snadné a uživatelské přívětivé. Pro návrh a tvorbu spolu s řízením databáze jsem používal SQL Developer a Oracle SQL Developer Data Modeler. Začal bych tou aplikací, kterou bych začínal pří návrhu databáze. Oracle SQL Developer Data Modeler, v němž budeme vytvářet logický model databáze, pak z něho vygenerujeme relační model a z relačního modelu se dostaneme k DDL skriptu, který bude potřeba naimportovat do Oracle SQL Developer [13].

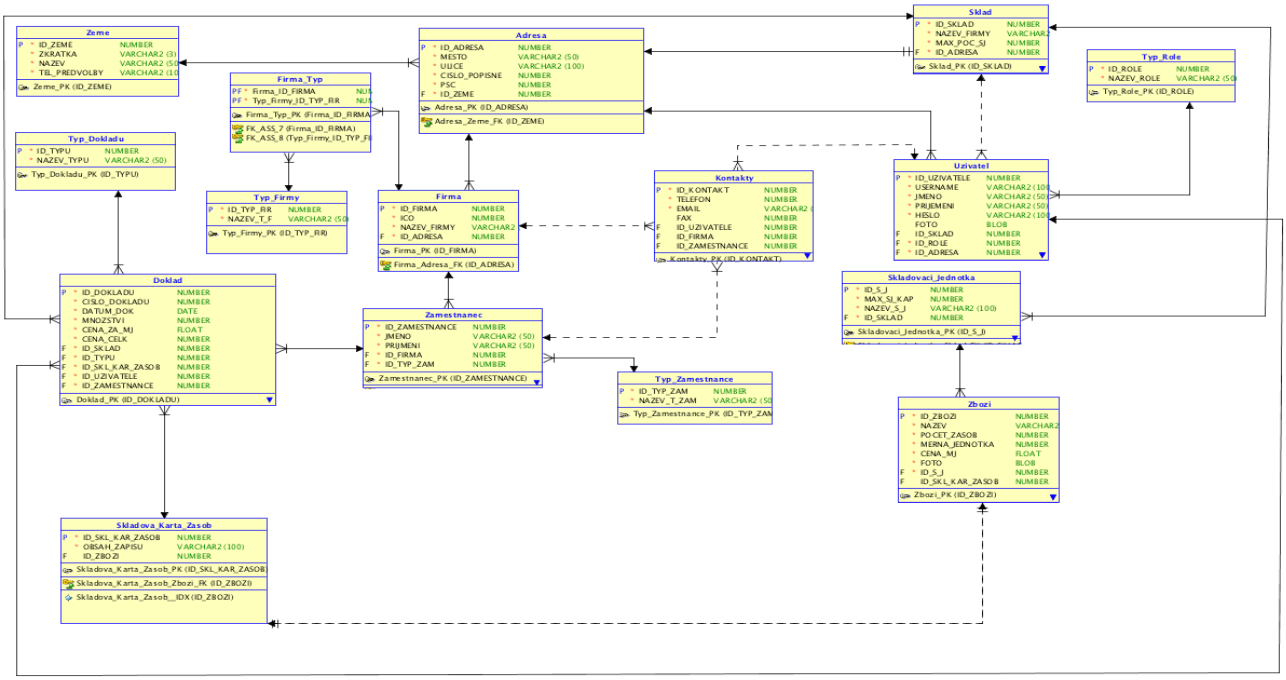
Dole uvádím několik obrázků, které by popsaly postup při vytváření databáze za pomocí nástrojů od Oracle. Obrázky jsou z mého vlastního vytvořeného skladovacího systému.

Obrázek 9 nám ukazuje, jak by měl vypadat logický model naší databáze, je to první krok při návrhu libovolné databáze.



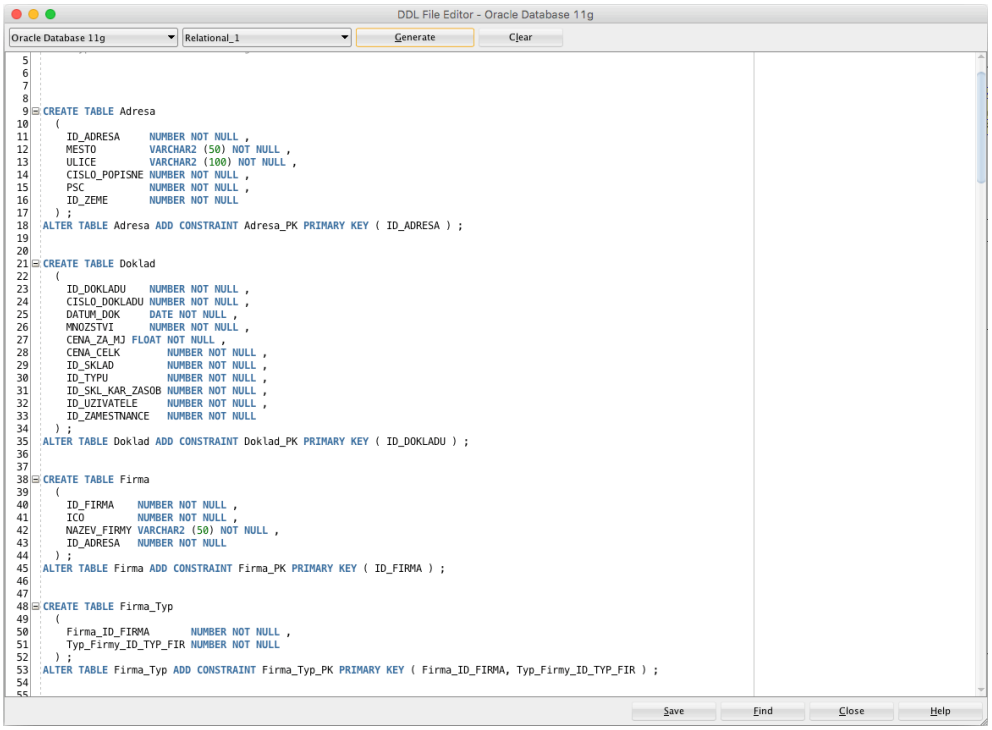
Obrázek 9 – Oracle SQL Developer Data Modeler logický model.

Obrázek 10 je vygenerovaný z logického modelu, relační databázový model, ze kterého se nám pak vygeneruje DDL skript.



Obrázek 10 – Oracle SQL Developer Data Modeler relační model.

Na Obrázek 11 je poslední krok před finálním vytvořením databáze na serveru. Zde máme vygenerovaný DDL skript, který pak importujeme v našem případě do Oracle SQL Developer a vytvoří se nám naše databáze, se kterou se můžeme začít pracovat.



Obrázek 11 – Oracle SQL Developer Data Modeler vygenerovány DDL skript.

Určitě pokud jste šikovný databázový programátor můžete vynechat první dva kroky a začít rovnou ručně psát DDL skript. Ale z mého pohledu je dobře umět jak za pomoci Oracle SQL Developer Data Modeler tak i bez něj.

5.2.2 MySQL

MySQL je systém pro řízení databází. V podstatě princip MySQL je naprosto stejný jako u MS SQL, protože databázový server MySQL je založený na jazyce SQL. Nejčastěji se s použitím MySQL můžeme potkat u webových stránek, ale jde to použit i jinde. Nejlepšími ukázkami užití MySQL databáze jsou: Facebook, YouTube, Twitter a tak dále. Tato open source databáze je nejoblíbenější ze všech, kvůli své spolehlivosti, výkonnosti a nejhlavnější jednoduchosti na použití [16].

5.3 Shrnutí

Abychom mohli postavit libovolnou databázi, je zapotřebí znát alespoň základy SQL, jelikož všechny systémy jako MySQL, SQLite, PostgreSQL jsou postaveny na SQL. Je potřeba využívat každý systém tam kde je opravdu zapotřebí.

6 Zkoumání trhu existujících aplikací

Spolu se svou manželkou často cestujeme autem, proto mně napadlo nějak si usnadnit život a hlavně ušetřit čas, který nelze vrátit zpět. Takže jsem se dostal k nápadu udělat aplikaci, která by mohla vypočítat všechno, co je potřeba a to za pár vteřin. Určitě jsem v této době nemyslel na zkoumání trhu aplikací, ale začal jsem rozvíjet svůj nápad na papír. Celé léto jsem přemýšlel, jak bych jej mohl realizovat a jak by mohl vypadat algoritmus pro nejhlavnější část aplikace, odkud bych měl čerpat data a tak dále. Pomalu, ale jistě jsem začal vymýšlet tento algoritmus a připravovat kostru androidové aplikace. Jakmile jsem pochopil, jak by mohl algoritmus vypadat, začal jsem zkoumat trh existujících aplikací. Dlouho jsem nemohl najit nic podobného, proto jsem došel k závěru, že jsem průkopník v tomto nápadu, ale pokazila mi náladu manželka, která večer omylem narazila na aplikaci ViaMichelin, která se zabývá stejnou problematikou.

Každý řidič zná Francouzkou firmu Michelin, tato firma má odvětví jako ViaMichelin. Určitě je vidět, že jejich aplikace, jak pro Androidovou, tak i IOS platformu, využívá hodně lidi po celé Evropě a dokonce i celém světě. Tak jsem byl trochu naštvaný, ale pak jsem pochopil, že v této době je velmi těžké vymyslet něco nového, co ještě neexistuje nebo není ve vývoji a jestli se to podaří, je třeba vědět, že někdo jiný už asi na tom rovněž pracuje.

Proto jsem se nezastavil a naopak jako torpédo vletěl do kódu, za jeden měsíc už jsem měl připravenou beta verze, pro testování mezi známými. Pro bakalářskou práci používám jinou verzi než kterou budu produkovat, s úplně jiným kódem a v jiném programovacím jazyce.

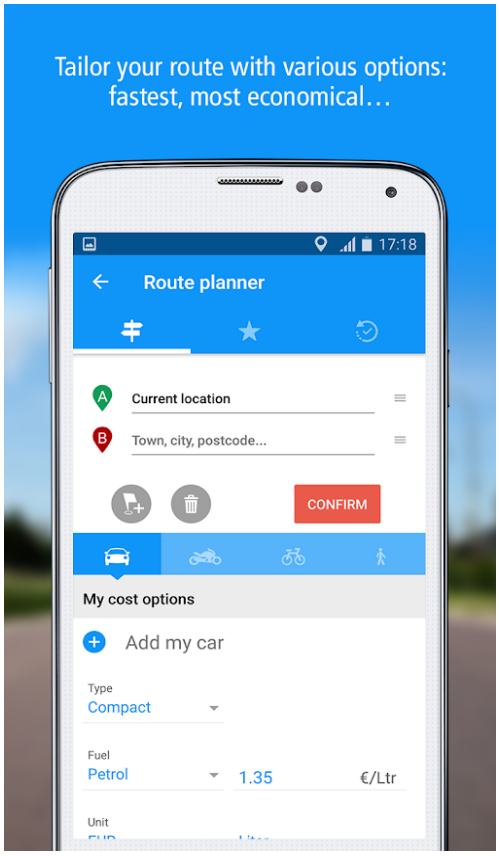
6.1 ViaMichelin

Je to dceřiná společnost skupiny Michelin, která se zabývá různými aplikacemi v kategorii cestovaní. Michelin má za sebou hodně různých druhů aplikací v teto kategorii, takové jako: Michelin Hotels- Booking, Michelin guide Europe 2018, Michelin Travel a nakonec GPS Traffic Speedcam Route Planner by ViaMichelin [18].

GPS Traffic Speedcam Route Planner by ViaMichelin aplikace řeší víc než jen jeden problém. Řeší jak plánování, tak i navigování, včetně upozorňování na rychlostní kamery. Aplikace je docela dobrá, jak designově, tak i funkčně. Proto je to zcela silný konkurent [18].

Ano tento konkurent je velmi silný a už má na trhu aplikace své zlaté místo, ale pro teď bych mohl zabojovat aspoň o bronzové nebo stříbrné místo.

Dále uvádím několik obrázků, jak teď jejich Androidová aplikace vypadá.



Obrázek 12 – ViaMichelin Start Route Planner GUI. Zdroj: [17].

Obrázek 13 – ViaMichelin Choose Route For Planning GUI. Zdroj: [17].

Uvádím obrázky dle nového designu, který realizovali v nové verzi od 9. 02. 2018. Design se úplně změnil a vypadá mnohem lepé než v minulých verzích. Aplikace ke dnešnímu datu má 5 000 000 – 10 000 000 instalací, což ukazuje že tato aplikace a hlavně tato problematika má poptávku na trhu.

Po prozkoumání komentářů k této aplikaci na Google Play Store jsem pochopil, že tato aplikace má malé drobnosti, které se nelibí uživatelům, například Čechům vádí, že tam není Čeština nebo nelze zadat typ auta a spotřebu.

Tyhle drobnosti se mi hodí a určitě už mám v produkované verzi a pomalu se připravuji k vydaní aplikace na Google Play Store a někdy na konci leta i do AppStore pro platformu IOS.

7 Analýza a návrh aplikace

Vytváření jak androidové tak i libovolné aplikace pro různé platformy není úplně lehká věc. Než se pustíme do implementace aplikace je potřeba projit několika standardními kroky.

Ve většině případu prvním krokem je sběr a specifikace požadavků. Spolu s tím je potřeba rozvíjet svůj nápad, který byste měl rozložit na části a řešit každou část samostatně, kvůli tomuhle byste mohl dorazit k cíli mnohem rychleji.

Teď začínáme se dvěma nejdůležitějšími fázemi, které budou náplní této kapitoly. Jedná se o analýzu a návrh aplikace. Projedeme každou fázi jak teoreticky tak i prakticky.

Jak bychom měli správně začít s analýzou aplikace? Je důležité analyzování aplikace? K čemu to je potřeba?

Takové otázky jsem také potkal pří vytváření své první aplikace. Rozhodně to jsou asi nejsložitější body při vývoji aplikace, ale pokud budete mít za sebou dobře vypracovanou analýzu a návrh aplikace, implementování u Vás zabere méně času. Proto jsem vložil hodně sil a svého času do analyzování problematiky a návrhu aplikace.

Traap je aplikace pro plánování a rozpočet cesty autem někam v rámci Evropské Unie. Tato aplikace nepotřebuje mít víceuživatelské rozhraní, ale přihlášení do aplikace v produkované verzi bude zajištěno za pomocí: Facebook, Gmail nebo Google +, Instagram, LinkedIn tedy za pomocí telefonního čísla. Tato aplikace by měla být nejenom funkční, ale i uživatelsky přívětivá, ale o tom bych promluvil až v návrhové časti.

Celá tato kapitola bude čerpat informace ze zdroje [19].

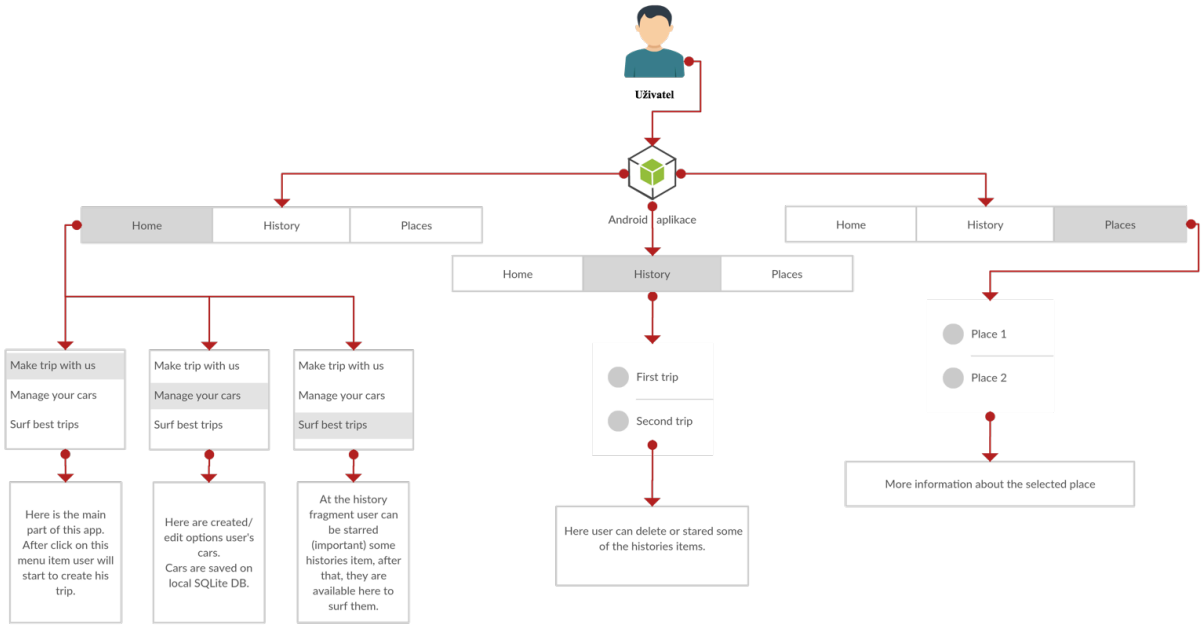
7.1 Analýza

Analýza je velmi důležitý bod, která se provádí za účelem tvorby analytického modelu, který je zaměřen na požadovanou funkčnost. Analytický model nepopisuje jak a jakým způsobem funkce bude provedena, přesně tohle má na starosti fáze navrhování. Model by měl být přehledný, srozumitelný všem kategoriím lidí a jednoduchý. V podstatě modeluje typy objektů a vztahy mezi nimi.

Prvním krokem je vytvoření analytického metamodelu, za pomocí jazyku UML.

UML (Unified Modeling Language) je jazyk, který byl navržen na propojení nejlepších existujících postupů modelovacích technik. UML není omezen jenom na modelování ve softwarovém inženýrství, ale najde své využití praktické všude.

Níže je uvedený Obrázek 14, který ukazuje jak by se měla aplikace chovat, jaké cesty bude mít uživatel, ale nezacházíme v tomhle modelu moc do detailů.



Obrázek 14 - Analytický model vytvořený za pomocí Creately.

Uživatel by měl mít telefon s platformou Android a nainstalovanou aplikaci Traap, pak ji může spustit a objeví se mu rozhraní s menu.

Menu bude obsahovat položky jako:

1. **Home (Domovská)**

   Na domovské části se nám objeví několik různých karet:

   a. **Make trip with us (Vytvořte výlet s námi)**

      Tady je nejhlavnější část aplikace Po klepnutí na tuto kartu bude uživatel nasměrován do aktivity, kde může začít vytvářet buď první nebo další výlet.

   b. **Create your car (Vytvořte své auto)**

      Tady uživatel může vytvářet, mazat a editovat svá auta.

   c. **Surf best trips (Zkoumejte své nejlepší výlety)**

      Tady se uživatelovi ukládají nejoblíbenější výlety.

2. **History (Historie)**

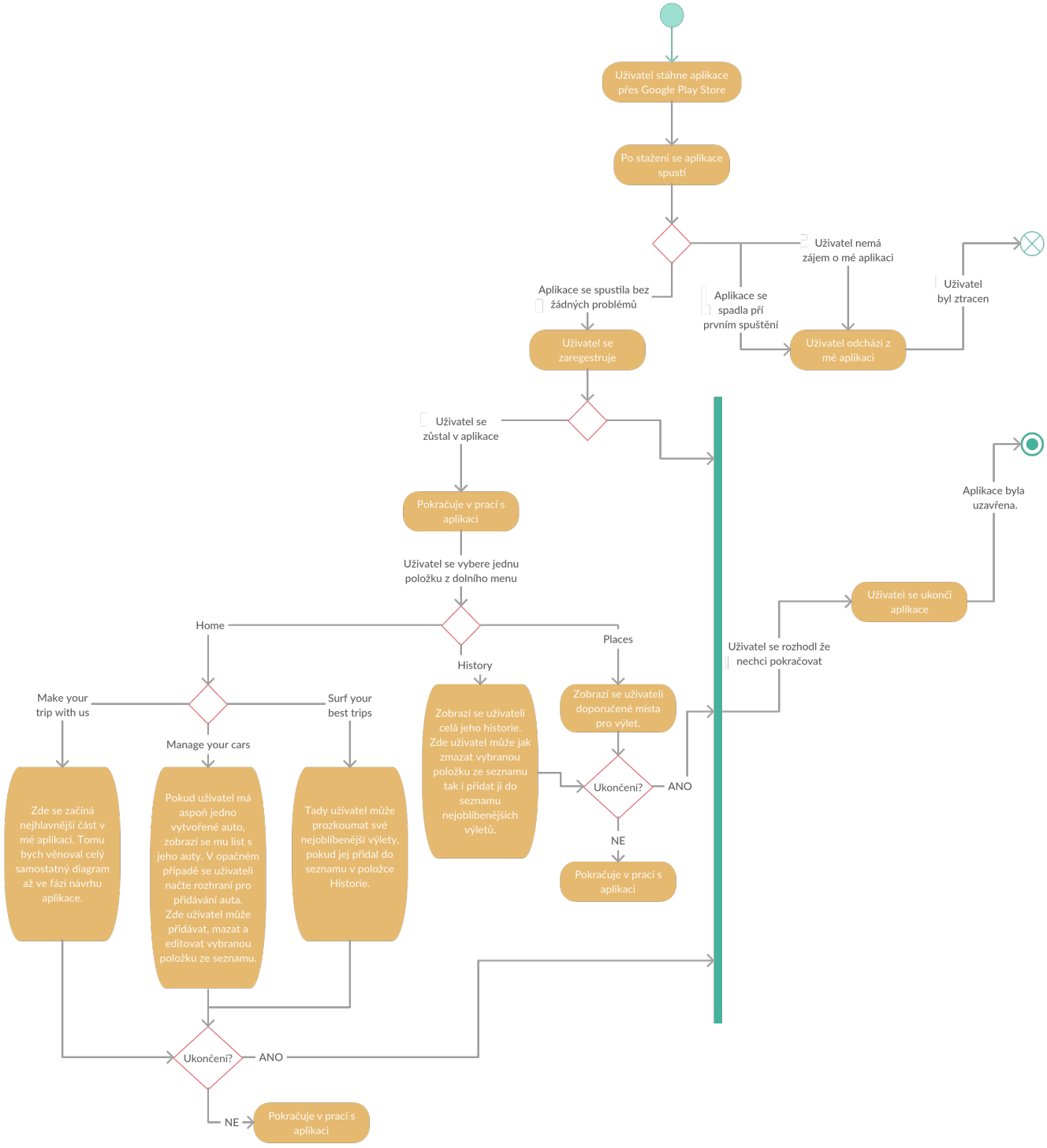
   V historii uživatel najde všechny výlety, které vytvářel.

3. **Places (Místa).**

   Zde se uživateli objevují doporučená místa pro výlet.

Analýza tím nekončí, zbývá nám poslední krok před navrhováním aplikace - diagram aktivit. Diagram aktivit se používá pro popis dynamických aspektů systému, tedy pro řízení aktivit, k modelování pracovních postupů a podnikatelských procesů.

Níže uvádím Obrázek 15, který popisuje aktivity uživatelů od prvního okamžiku, kdy se potkal s mou aplikací.



Obrázek 15 – Diagram aktivit vytvořený za pomocí Creately.

Podle mého názoru model, který jsem si vytvořil je zcela srozumitelný a jde pochopit nejen programátorovi, ale i člověku bez odborné kvalifikace v oboru IT. Čímž splňuje podmínky pro návrh Analytického modelu.

Šlo by tyto modely udělat pokročilejší a více do hloubky, ale tímhle bych porušil zásady pro vytváření analytických modelů.

7.2 Návrh

Co je návrh a jak s tímto démonem bojovat? Tak se podíváme na to spolu, návrh aplikace je poslední fáze a zároveň i nejkomplikovanější ze všech. Jak jsme pochopili z předchozí kapitoly, analýza aplikace je zaměřena jenom na funkce, nikoliv na jejich specifikaci, právě tímhle se bude zabývat návrh. Jinými slovy smyslem návrhu je specifikace různých způsobů implementace těchto funkcí. Základním kamenem návrhového modelu je model analytický, jedná se tedy o jeho rozšíření. Návrhový model obsahuje více podrobností a hlavně konkrétní technické řešení.

Pokud byl návrh správně navržen, výsledkem by měl být model systému, který lze rovnou implementovat.

Každá část toho analytického modelu se zabývá jedním vlastní problémem, proto bych rozdělil ten návrhový model na několik malých modelů, které by ukázaly, jak přesně ten problém bude vyřešen.

Před každou částí návrhu budeme probírat teorii a pak samotný návrh.

Začneme od nejlehčích častí ke složitějším.

7.2.1 Places (místa) Fragment návrh

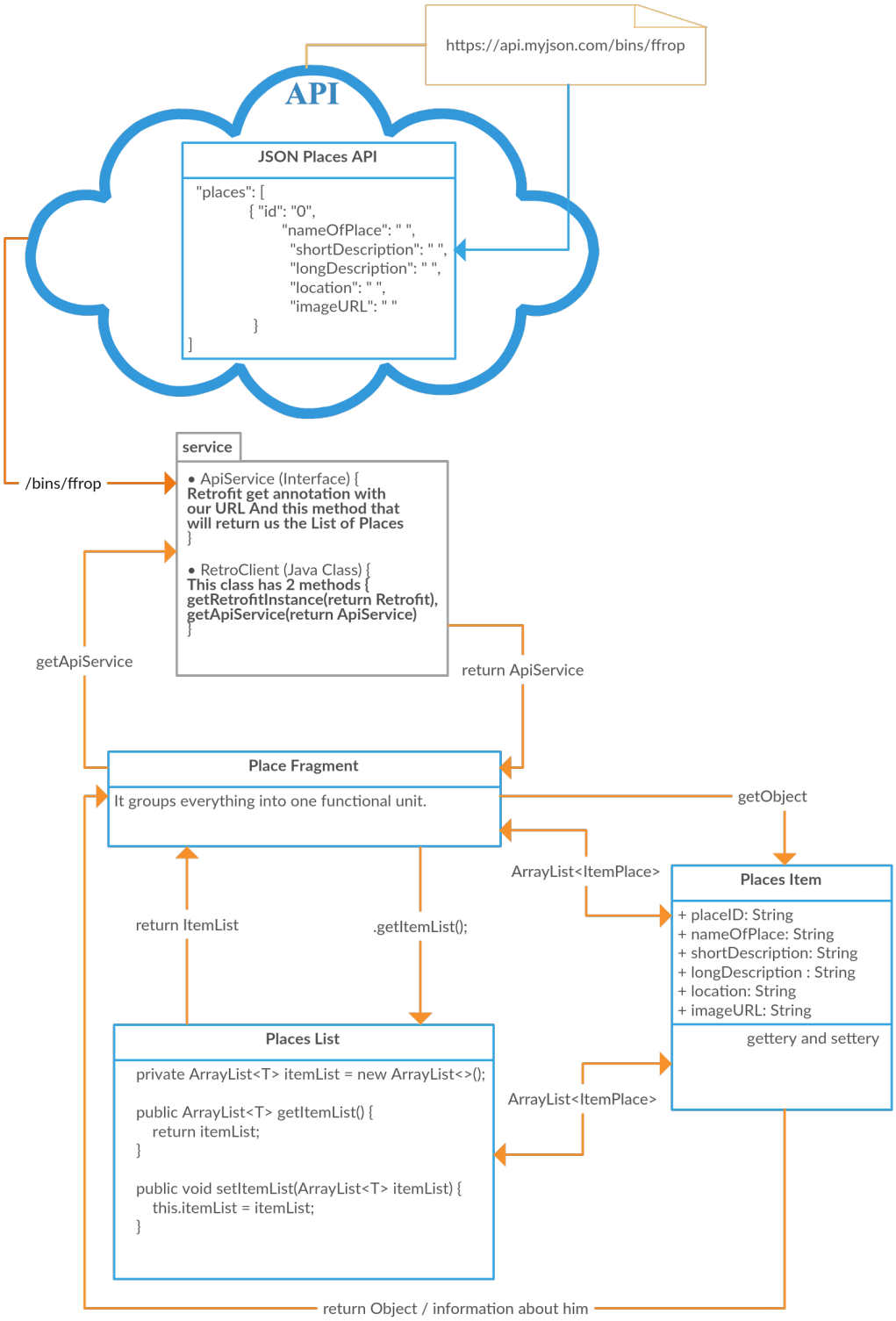
Tato část by měla poradit uživateli nějaká TOP místa v evropské unii. Pro bakalářskou práci jsem vytvořil takzvaný vlastní API pro tato místa. Je to JSON soubor, který je uložený na speciálním úložišti[2], ze kterého pak za pomocí knihovny Retrofit [3]vytahujeme data.

REST (Reprezentační státní přenos) nebo RESTful, je docela široké téma, ale zkráceně zajišťuje interoperabilitu mezi systémy na internetu. Webové služby kompatibilní s protokolem REST umožňují požadujícímu systému přístup k textovým reprezentacím webových zdrojů a manipulaci s nimi. Retrofit je klientem REST pro Android a Java, který byl vytvořen firmou Square. Za pomocí webové služby REST je poměrné snadné, jak načíst, tak i nahrát JSON, případně i jiné strukturované údaje. V nástroje Retrofit můžeme nastavit, který konvertor budeme používat pro serializaci, například pro JSON budeme používat GSon. Pro HTTP požadavky se Retrofit používá OkHttp [20].

[2] https://api.myjson.com

[3] http://square.github.io/retrofit/

Níže uvádím Obrázek 16, který přesně popisuje, jak by měl pracovat PlacesFragment. Tento model je pochopitelný a podle něj jde rychle naimplementovat tuto část aplikace.



Obrázek 16 – Návrhový model pro PlaceFragment vytvořený za pomocí Creately.

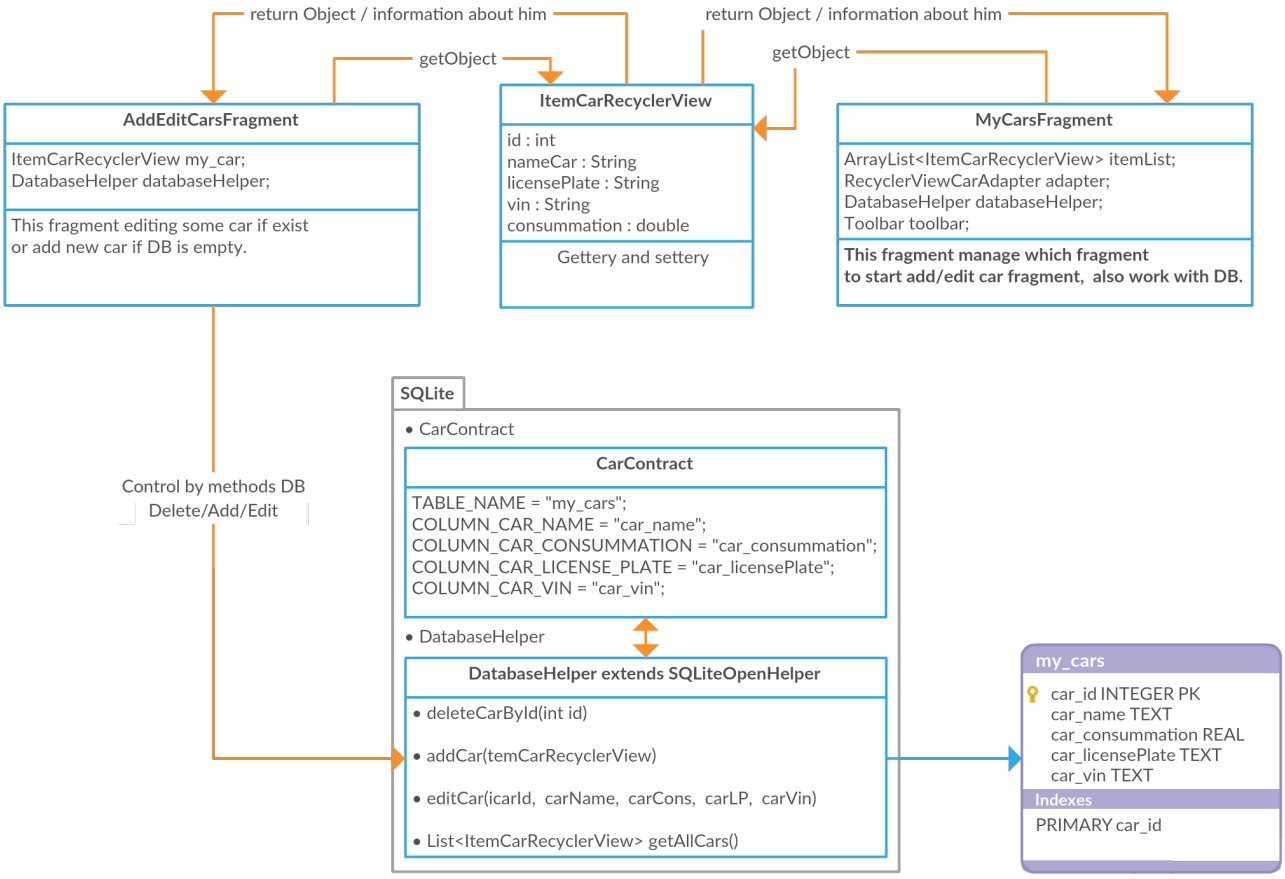
7.2.2 Cars (auta) Fragment návrh

Tato část návrhu se bude zabývat uživatelskými auty. Jelikož tato aplikace je pro sestavování a rozpočtu cesty autem, předtím než začneme počítat, musíme něco vědět o uživatelském autě. Nejdůležitější pro nás je vědět poměrnou spotřebu na 100 kilometrů.

Ale jde to udělat různými způsoby: prvním způsobem je na pevno zadanou průměrnou spotřebu, dalším je například mít bokem API s auty a informace o nich a tak dále těch způsobů jsou mraky. Budeme se zabývat tím způsobem, který jsem zvolil pro svou bakalářskou práci, tedy tím prvním.

Takže budeme potřebovat SPZ a VIN číslo automobilu, do budoucna, přesněji v produkované verzi pokud uživatel bude souhlasit, za pomocí těchto údajů můžeme zakoupit elektronické dálniční známky.

Níže uvádím Obrázek 17, který zjevně bude pochopitelný a hodně pomůže při implementování této časti.

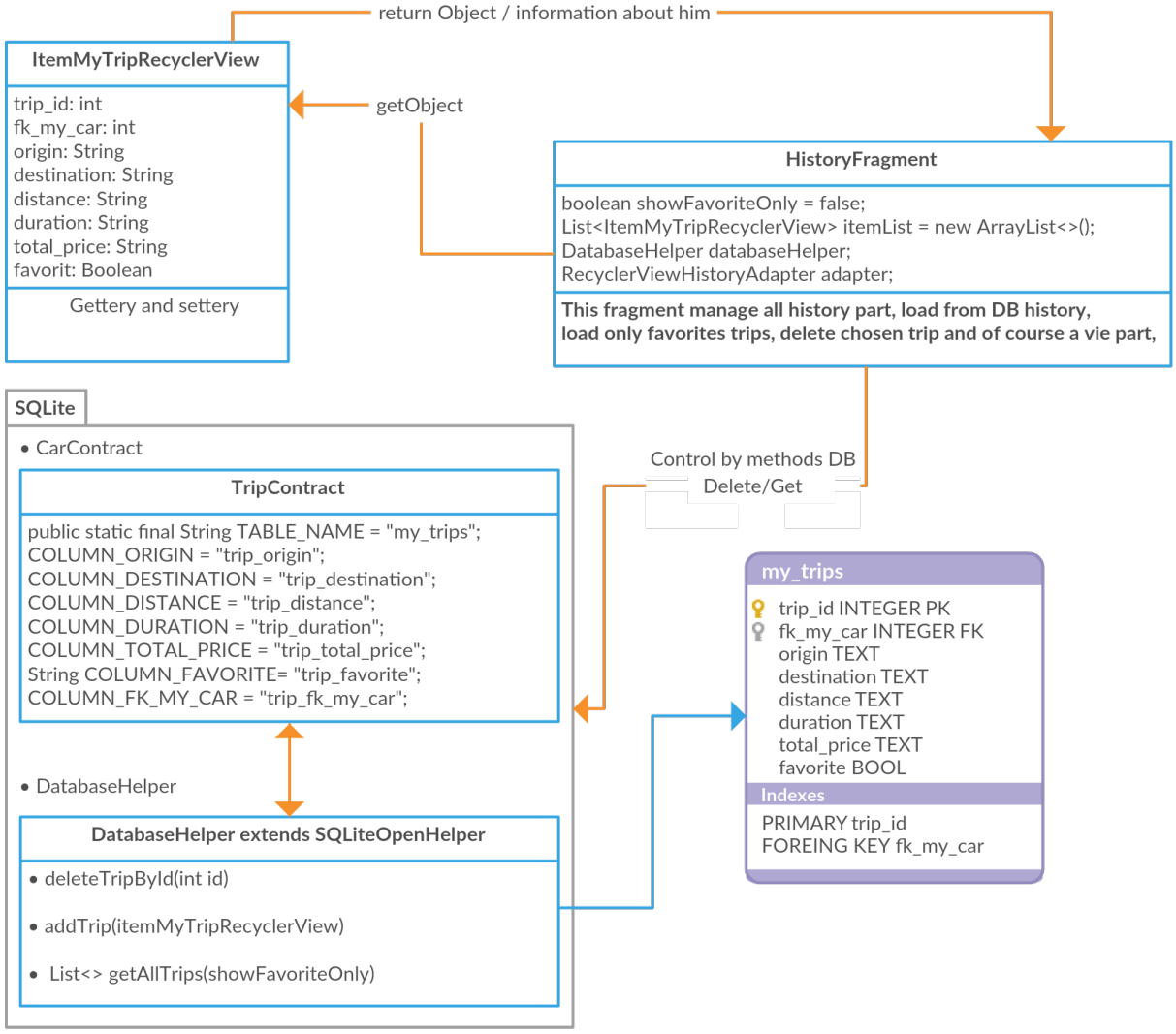


Obrázek 17 – Návrhový model pro MyCarsFragment vytvořený za pomocí Creately.

7.2.3 History (historie) Fragment návrh

Zde potřebujeme nějak vyřešit ukládání výletů do historie. Jako vždy to jde udělat různými způsoby, ale nejlepším bude využití lokálního databázového prostoru na androidu, jako v případě prací s auty.

Níže uvádím Obrázek 18, dle kterému lze začít implementovat tuto část do projektu.



Obrázek 18 - Návrhový model pro HistoryFragment vytvořený za pomocí Creately.

Teď jsme probrali předposlední část návrhu mé aplikace a jak jsem říkal, že nechám tu nejzajímavější, nejsložitější a nejdůležitější část aplikace až na konec, tak jsme se k ní dostali.

7.2.4 Maps (mapa) Activity návrh

Zde se budeme zabývat nejhlavnější částí návrhu mé aplikace, tahle část sestavuje a rozpočítavá uživatelskou cestu a na konci by měla nasměrovat uživatele k jeho výsledkům.

Před tím, než přejdu k návrhu, měl bych probrat pár základních teoretických bodů. V této části budeme využívat Google Maps API, Google Maps Directions API, Google Places API a Own Countries Cost API, ale primárně nás bude zajímat, jak bude aplikace pracovat s Google APIs.

Google APIs – firma Google nám poskytuje různé druhy APIs, které bychom mohli napojit jak na mobilní aplikace, tak i na ty webové. Primárně se budeme zabývat APIs pro mapy. Těch APIs pro mapy je dost, ale pro bakalářskou práci jsem zvolil Google Maps API, Google Maps Directions API a Google Places API. Pojďme se podívat na každou z nich samostatně.

1. **Google Maps API**

Pro přidání mapy, která bude založená na datech z Google Maps, do androidové aplikace využijeme **Google Maps Android API**. Rozhraní API automaticky zpracovává přístup k serverům Google Mapy, zobrazování mapy a reakci na gesta uživatelů, jako jsou například kliknutí a přetahování na mapě [2].

2. **Google Maps Directions API**

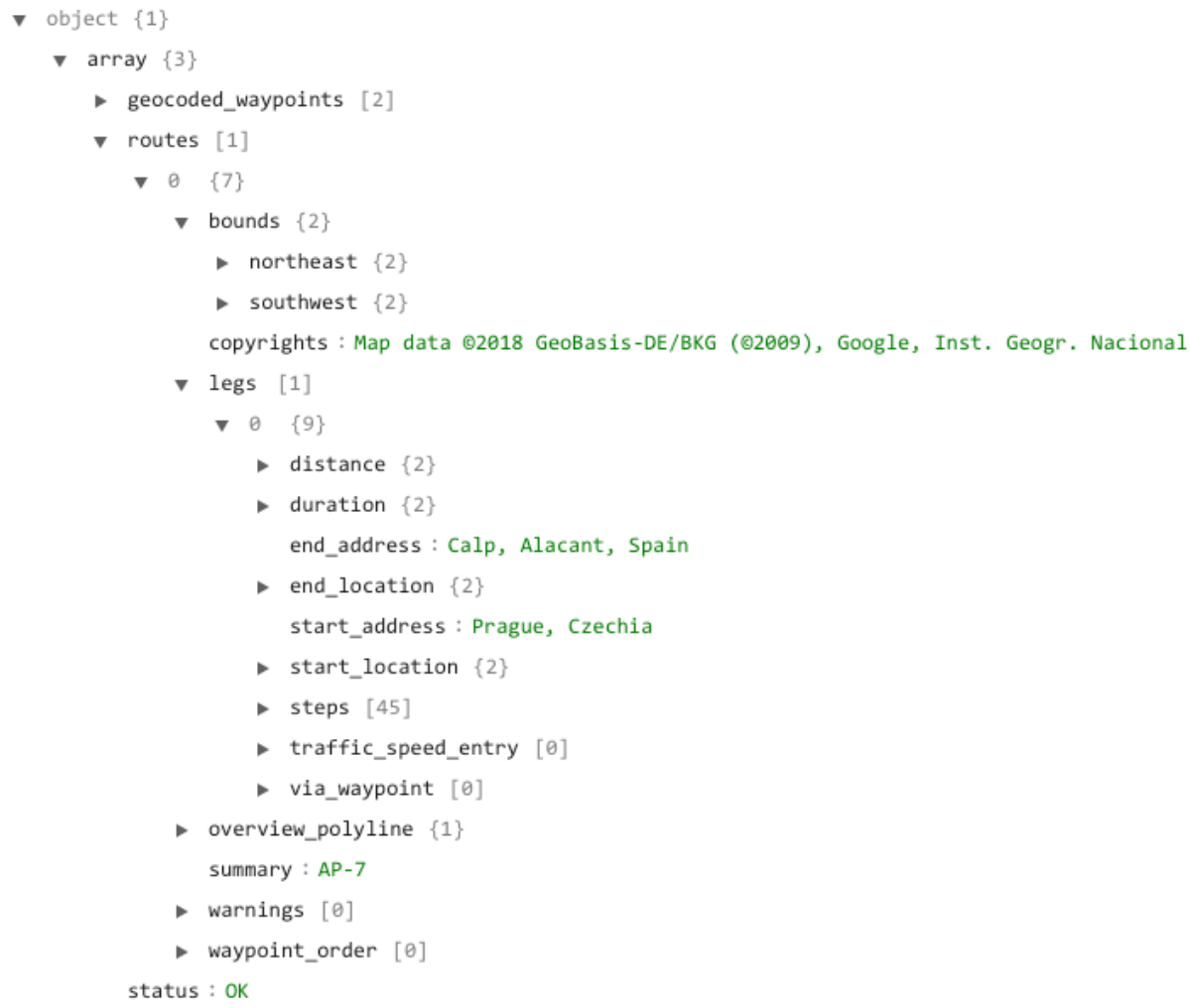
Pro sestavování a hlídání případné cesty se používá **Google Maps Directions API**. Obsahuje trasové body, které nabízejí možnost měnit trasu přes určité místo. Určete počty, cíle a body jako textové řetězce (např. "Praha" nebo "Pardubice, Pardubičky, 530 09 ") nebo za pomocí souřadnic (50.047982, 14.342626) [2].

3. **Google Places API**

Pokud je v aplikaci potřeba stále něco vyhledávat (nějaká místa nebo dokonce i kavárny), můžeme použit **Google Places API**. Dostáváme přístup k jedné velké databázi, používané službou Google Mapy. Tento API v sobě zahrnuje více než 100 milionů podniků a zajímavých bodů, které jsou často aktualizovány uživateli služeb Google. Tento API nám poskytuje: Umístění detailů (bohaté podrobnosti o místě), umístění automatického doplňování (automaticky vyplňuje jméno nebo adresu místa), umístění fotografií (kvalitní fotografický obsah) [2].

Teď bychom mohli přejit k návrhu této částí. Snažím se jít krok za krokem, abyste se v tom neztratili, proto to nejprve popíši slovy. Jakmile uživatel vstoupí do této částí aplikace, měl by vyplnit jenom dvě pozice - odkud a kam pojede, a vybrat auto ze seznamu svých aut, pokud je seznam prázdný, aplikace vynutí uživateli přidat alespoň jedno auto. Odkud a kam přeposílám do Google Maps Directions API a pokud cesta mezi dvěma body existuje, vrátí se do aplikace JSON soubor, se kterým bych mohl začít pracovat za pomocí Retrofitu a konvertoru GSON můžu snadně a rychle pracovat s tím souborem.

Níže uvádím Obrázek, který ukazuje jak vypadá struktura JSON souboru od Google Maps Directions API.



Obrázek 19 - Struktura JSON souboru od Google Maps Directions API

Tento soubor se vrátí do aplikace po zpracovávání požadavku uživatele: najdi cestu mezi dvěma body „Prague, Czechia“ a „Calpe, Alacant, Spain“. Pojďme probrat tento soubor do hloubky. Z tohohle souboru nás bude nejvíc zajímat pole *„routes“*.

Routes (Trasy) je pole objektů, každý objekt reprezentuje jednu možnou trasu. V té trase jsou jak objekty, tak i pole objektů. Uděláme tedy strukturu a popíšeme každou její část.

Routes (Trasy)

- **0 (první trasa)**: objekt
    - **bounds (hranice)**: objekt
      Obsahuje další dva objekty, které reprezentují severovýchodní a jihozápadní hranice v podobě zeměpisné délky a šířky.
    - **copyrights (autorská práva)**: řetězec
      Řetězec, který obsahuje text autorských práv, který se má zobrazit pro tuto trasu.
    - **legs (nohy)**: pole
      Pole, které obsahuje další pole, obsahující informace o úseku trasy mezi dvěma místy v dané trase. Nohám se ještě budu věnovat níže.
    - **overview_polyline (přehled polylinu)**: objekt
      Obsahuje jeden bodový objekt, který obsahuje zakódované znázornění trasy v křivce. Tato křivka je přibližná (vyhlazená) cesta výsledných směrů. Polyline se ještě budeme věnovat níže.
    - **summary (souhrn)**: řetězec
      Je to krátký textový popis trasy, vhodný pro pojmenování a odlišení trasy z alternativ [2].
    - **warnings (upozornění)**: pole
      Pole varování, které se zobrazí při zobrazování těchto pokynů [2].
    - **waypoint_order (pořadí bodů)**: pole
      Pole, které udává pořadí jakýchkoli bodů v vypočtené trase [2].

Legs (Nohy) je pole, které obsahuje další pole a každý prvek v tomto poli určuje jednu etapu cesty od místa původu do cíle ve vypočítané trase. Pro trasy, které neobsahují žádné body trasy, bude trasa tvořena jednou "nohou", ale pro trasy, které definují jeden nebo více trasových bodů, bude sestávat z jedné nebo více noh odpovídajících konkrétním nohám cesty [2].

Legs (Nohy)

- **0 (první noha)**: objekt
    - **distance (vzdálenost)**: objekt
      Obsahuje dvě proměnné vzdálenosti v různých jednotkách, první je v kilometrech a druhá v metrech [2].
    - **duration (trvání)**: objekt
      Obsahuje, stejně jako i předchozí, dvě proměnné doby trvání, v různých jednotkách, první je v hodinách a druhá v minutách [2].
    - **end_address (konečná adresa)**: řetězec
      Řetězec s konečnou adresou [2].
    - **end_location (konečné umístění)**: objekt
      Objekt s konečnou adresou, která je reprezentována v podobě zeměpisné délky a šířky [2].
    - **start_address (počáteční adresa)**: řetězec
      Řetězec s počáteční adresou [2].
    - **start_location (počáteční umístění)**: objekt
      Objekt s počáteční adresou, která je reprezentována v podobě zeměpisné délky a šířky [2].
    - **steps (kroky)**: pole
      Obsahuje řadu kroků, které označují informace o každém jednotlivém kroku cesty [2].
    - **traffic_speed_entry (dopravní rychlost)**: pole
      Pole dopravních rychlostních omezení.
    - **via_waypoint (přes trasový bod)**: pole
      Je pole mezibodů na cestě, pokud body nejsou, pole je prázdné. Bod trasování je určen v podobě zeměpisné délky a šířky [2].

Steps (Kroky) je pole objektů, kde každý objekt je jeden krok. Krok popisuje konkrétní jednotnou instrukci na cestě, například „Odbočte doleva u ulice Jana Pernera“. Tento krok nejen popisuje instrukci, ale také obsahuje informace o vzdálenosti a trvání týkající se toho, jak se tento krok týká následujícího kroku. Například mějme dva kroky, první má distance 0.2 km a druhý bude mít 0.4 km. Jdeme popsat jednotlivé části kroku.

Steps (Kroky)

- **0 (první krok)**: objekt
    - **distance (vzdálenost)**: objekt

      Objekt, který obsahuje vzdálenost pokrytou tímto krokem až do dalšího kroku. Více informací o tomto poli najdeme na předešlé stránce.
    - **duration (trvání)**: objekt

      Objekt, který obsahuje typický čas potřebný k provedení kroku, až do dalšího kroku. Více informací o tomto poli najdeme na předešlé stránce.
    - **end_location (konečné umístění)**: objekt

      Objekt, který obsahuje umístění konečného bodu tohoto kroku, které je reprezentováno v podobě zeměpisné délky a šířky [2].
    - **html_instructions (html pokyny)**: řetězec

      Textový řetězec, který obsahuje formátované pokyny pro tento krok.
    - **polyline (polylin)**: objekt

      Objekt, který obsahuje textový řetězec s kódovanou reprezentací křivky kroku. Tato křivka je přibližná (vyhlazená) cesta kroku.
    - **start_location (počáteční umístění)**: objekt

      Objekt, který obsahuje umístění počátečního bodu tohoto kroku, které je reprezentováno v podobě zeměpisné délky a šířky [2].
    - **travel_mode (způsob dopravy)**: řetězec

      Je to řetězec, který říká, jaký způsob dopravy byl zvolen při této cestě (jízda, pěší, jízda na kole, tranzit). V nových verzích API bylo to změněno na *available_travel_modes,* který obsahuje pole dostupných režimů cestování pro zvolenou trasu, což nám například z Pardubic do Prahy vrátí tyto způsoby dopravy: jízda a pěší.

Předtím než se pustíme do návrhu MapsActivity, zbývá nám projít poslední bod z toho JSON souboru a to je *overview_polyline (přehled polylinu)*. Dříve už jsem psal, že *overview_polyline* je objekt, který v sobě má řetězec zakódovaného znázornění trasy v křivce. Tohle se používá pro vykreslení polylinu na mapě po sestavení uživatelské cesty. Ale abychom mohli ten řetězec použít pro vykreslování, musíme ho dekódovat do *List<LatLng>* za pomoci *PolyUtil.decode (overview_polyline)* - dekóduje kódovaný řetězec cesty do sekvence LatLngs.

Níže uvádím Obrázek 20 a Obrázek 21, první ukazuje, jak vypadá polyline před a druhý po dekódování v mé aplikaci.

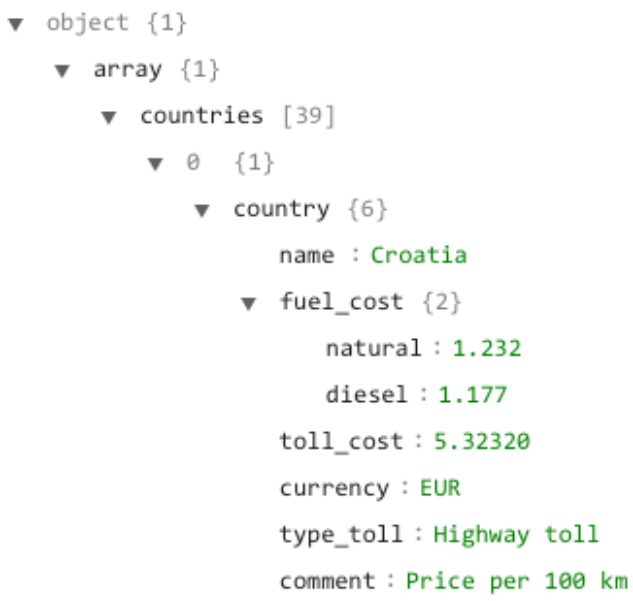
▼ overview_polyline {1}

points : akspH}zbwA`_BlaS`qEzcXvtKt{VhjPvhb@pdGf}VtqHp{NpcJpyk@rqDjsFrbAzgH`WjhNkpDnjHi@`cf@{j@|hY{yEp}S|iMdyf@rzGzjZdjCjaZpfGj|RryIjeL~cExcDvwEbaG~}GpgUjQtr`@etCzsRtfA|zT
}q@pb[piDzbu@riJhfUdpCnoLcb@x|UbvI|ge@n]nec@~tDpjh@fjCp_K_]~yKfvEjp`@okA~ap@quCptg@jkEz{TviCplSaxDdr`@orN`naAge@~lZvyBvgIx}UtlHbuZfoU|`FjeIh|IxoHnvJtaTr{JbvN|kZxo
Vba[rgWpuKfhAfgKj|Oh~^lnJp{W{wJlyFvNtyHxzJxkGtfTdhKnuJ~bMt{Cn~DjiGtuAxxb@xnIxni@~rB~fPd{EvbNziCv}J~hJn_CjpGwKzaFvpGpeIz~NbbDveSngAhb[zqDdlPxnElp[beErbb@pkFxhSdmBh
rItrEdjEngF~e[lwHdzf@f_Crn\\|zKld]tsBjlUezApeOa|An_UhaI`fI|xWdyGjgYisAjwTeeDnsI{}FlzHi{B|qLtiGlqXlsJl~GdpDbpHif@hj_@fwIhaHx_BlcJmFfmW}~CzfKyrDlqDooGlpKakCpkQ`tH~h
KorKvpIpsDdrT|`D|wSsuDrtNmrHdfQhr@tbc@tMzoNtdBjiH`yInrSve@djZuArhUp`LhiLkl@vmFex@f_HvtDlmH{d@t_L}eIlqG{nA`cFhsCheDrGxiY~gj@nhC~_L`qGbwKvyLlvZbdGzwRllH`fZvmMpiPv`I
rnXfyKbrTvjS`{o@lfG`{\\lpLvqUpmUdwd@rtPxpHlrUqsEzsG{gCzcGnjCvdBzbHzqHblDfeQt_EfoSbrA|hGjV`rD~gDzxNodH|xNmvCxqGgoHxaEzBhmT`j@`~LndCbvKhzHzpBt|FbuFpeF`|JtV~dVdcEr_H
d{Df}BvxI`fFpzZ`lKvdT~sPv_b@njMrnd@n`B|`EsT|yD|_@tnPjiNv}\\jfItqIpeGneSt_KdhJzlGloMreGdhSq]nm[ndSlpn@hpCteHlvEnkB~q[jz[hbWzlc@bb_@vjY~dObp@`zL`}GxhQbqXhdKrsBhgGlo
Gn{TlwIngCniK``H~cRtwHdlN~dQdFhg_@vnVjzBnaHhkF`nAlsKzxRlwVd~[lqV}qWleNmsBxoN}f@tqPyuZhuRu{E~vIs~ElpEuxJ~hGwsT~eKyuJtqDwnCzaChrAjkDmVthHtsB|mEgm@

Obrázek 20 – Overview polyline před dekódováním.

Obrázek 21 – Overview polyline po dekódováním v mé aplikaci.

Tento soubor je v podstatě to nejhlavnější v této aplikaci, proto jsem mu věnoval více než ostatním. Ale zbývá nám projít ještě jeden API, který se nazývá CountriesCostsApi. Tohle je můj vlastní vytvořený API pro celou Evropskou Unii s cenami mýtného. Soubor není těžký, obsahuje jedno velké pole *„countries“*, které v sobě má různé státy reprezentované jako objekty.



Obrázek 22 - Countries Cost API

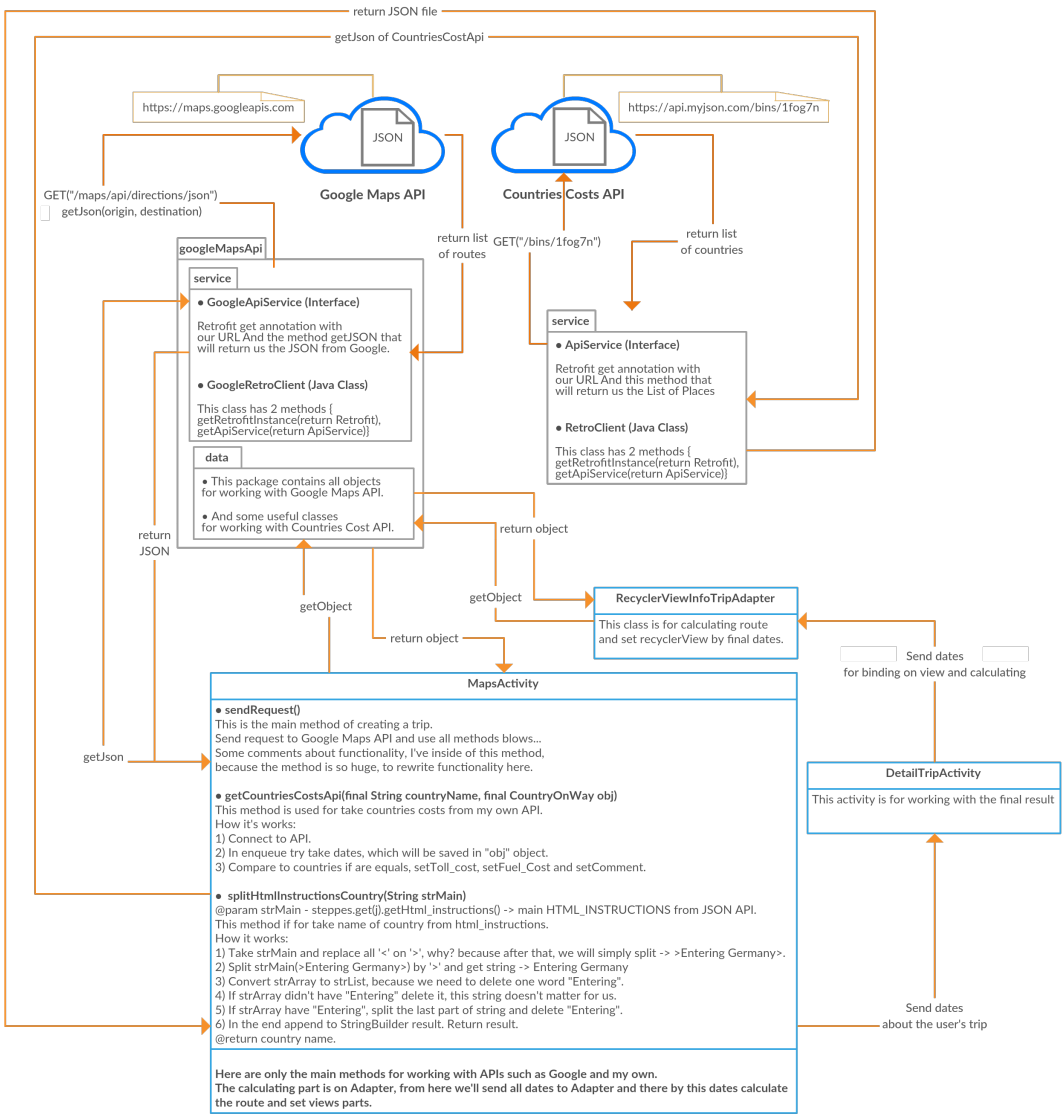
Country (Stát) je objekt, který obsahuje nějakou informaci o daném státu. Výše uvádím Obrázek 22, na kterém je dobře vidět, kterou informaci o státu nám objekt bude poskytovat. Pojďme rychle projít každý parametr v tomto souboru.

Country (Stát)

- **0 (první stát)**: objekt
    - **name (název)**: řetězec – název konkrétního státu.
    - **fuel_cost (cena paliva)**: objekt
        - **natural**: řetězec – cena benzínu.
        - **diesel**: řetězec – cena nafty.
    - **toll_cost**: řetězec – cena mýtného.
    - **currency**: řetězec – měna, pro BP jsem zvolil všude stejnou měnu - eura.
    - **type_toll**: řetězec – typ poplatku (Dálniční poplatky, známky, poplatky za tunely, mosty a tak dále).
    - **comment**: řetězec – komentář, který se zobrazuje uživateli při rozpisu cen k danému státu.

Konečně se můžeme vrhnout na návrh MapsActivity.

Níže uvádím Obrázek 23, který demonstruje, jak by mohl vypadat návrhový model pro MapsActivity, určitě by to šlo udělat podrobněji, ale pak by se sem model ani nevešel.



Obrázek 23 – Návrhový model pro MapsActivity vytvořený za pomocí Creately.

Snažil jsem se na tom modelu alespoň popsat, jak pracuje MapsActivity: odkud bere, jak a kde zpracovává data a kam je posílá dál. Velkou roli v mé práci hraje můj algoritmus na rozpočet cesty, rád bych ho tedy probral. Jak jsem říkal, abych mohl rychle přebírat data z API, používám takzvaný Retrofit, popisoval jsem ho výše, pokud budete chtít, můžete se k tomu vrátit. Abych mohl poslat asynchronní požadavek na Google Maps API použiji „enqueue“, který oznámí „callback“ jestli to proběhlo v pořádku nebo ne.

Abychom mohli zjistit, jak to proběhlo, použijeme metody od *„callback“:*

1. **„onResponse“** – musíme nejprve zkontrolovat odpověď, jestli je úspěšná nebo ne. Pokud ano, můžeme pokračovat dál, v opačném případě se vyvolá dialogové okno s případnou chybou. Příklady chyb: *404* (objekt nenalezen – objekt s požadovaným URL neexistuje, většinou se jedná o překlep v URL) a *500* (vnitřní chyba serveru – při zpracování dotazu došlo v programu serveru k blíže neurčené chybě).

2. **„onFailure“** – pokud došlo k neočekávané výjimce nebo k výjimce sítě, při komunikaci se serverem, *onFailure* bude vyvolán. Zde se můžeme zobrazovat nějaké dialogové okno s chybovou hláškou.

První jeho část je onResponse, představme si, že se nám vrátila odpověď bez žádné chyby, můžeme tedy začít. První, co uděláme, je, že zkontrolujeme pole trasy (routes) jestli není náhodou prázdné, pokud ne, pokračujeme v algoritmu. V opačném případě se nám vyvolá dialogové okno s chybovou hláškou. Pak za pomocí trasy (routes) přistoupíme k nohám (legs) a až poté ke krokům (steps). Teď bychom měli projít všemi kroky, níže uvádím Obrázek 24 s úryvkem kódu a pod obrázkem ho budu popisovat.

```
for (int j = 0; j < steppes.size(); j++) {
    countryBoardStart = new LatLng(steppes.get(j).getLatLonCoordinatesEnd().getLat(), steppes.get(j).getLatLonCoordinatesEnd().getLng());
    //if it's first/last step, we didn't used html_instructions.
    if (steppes.size() - 1 == j || j == 0) {
        if (j == 0) {
            String countryName = getCountryNameFromAddress(legs.getStartAddress());
            obj = createCountry(idCountry, countryName, countryBoardStart, distance: "0", comment: "0", toll_cost: "0", fuel_cost: "0");
            countryOnWays.add(obj);
            getCountriesCostsApi(countryName, obj);
            idCountry++;
        } else {
            countryOnWays.add(createCountry(idCountry, destination, countryBoardStart, distance: "0", comment: "n/a", toll_cost: "0", fuel_cost: "0"));
            idCountry++;
        }
    }
    //if it's not first/last step we used html_instructions, for get countryName.
    String countryName = splitHtmlInstructionsCountry(steppes.get(j).getHtml_instructions());
    if (!countryName.isEmpty()) {
        obj = createCountry(idCountry, countryName, countryBoardStart, distance: "0", comment: "0", toll_cost: "0", fuel_cost: "");
        countryOnWays.add(obj);
        getCountriesCostsApi(countryName, obj);
        idCountry++;
    }
}
```

Obrázek 24 – Kolika různými státy pojedeme (úryvek kódu).

Pro první a poslední krok není potřeba kontrolovat html_instructions, je potřeba z nich vytvořit objekty, přidat do listu a jenom pro první je potřeba dle názvu nahrát ceny z CountriesCostsApi, zatímco pro poslední to není potřeba, protože to přidáme jinde. Pak začínáme procházet všemi ostatní kroky. Pro ně je potřeba zkontrolovat html_instructions. Proč je potřeba kontrolovat html_instructions pro každý krok?

Tohle potřebujeme proto, abychom mohli pochopit kolika různými státy projíždíme a pak od každého vstupu do jiného státu, uložím souřadnice vstupní hranice.

Mimochodem bychom se mohli podívat na funkce, která nám vrátí z html_instructions jméno státu v podobě řetězce.

Tuto funkce bych raději popsal přímo na reálném příkladu html_instructions: „*Continue onto E50<div style=\"font-size:0.9em\">Partial toll road <div style=\"font-size:0.9em\">Entering Germany *“. Z tohohle velkého řetězce potřebujeme jenom jméno státu, což je tady Germany (Německo). Dolů uvádím Obrázek 25 s úryvkem kódu pro rozdělování html_instructions, s podrobným podpisem pod tím.

```
private String splitHtmlInstructionsCountry(String strMain) {
    String result;
    StringBuilder stringBuilder = new StringBuilder();
    strMain = strMain.replaceAll( regex: "<", replacement: ">");
    String[] strArray = strMain.split( regex: ">");
    List<String> strList = new ArrayList<>(Arrays.asList(strArray));
    for (int i = 0; i < strArray.length; i++) {
        if (!strArray[i].contains("Entering")) {
            strList.remove(strArray[i]);
        } else {
            strMain = strArray[i];
            strArray = strMain.split( regex: " ");
            strList = new ArrayList<>(Arrays.asList(strArray));
            for (String aStrArray : strArray) {
                if (aStrArray.contains("Entering")) {
                    strList.remove(aStrArray);
                } else {
                    stringBuilder.append(aStrArray).append(" ");
                }
            }
        }
    }
    result = stringBuilder.toString();
    return result;
}
```

Obrázek 25 – Vytahovani z html_instructions jméno státu (úryvek kódu).

Prvním krokem v tomto řetězci vyměníme všechny levé „<“ uhlové závorky za pravé „>“ uhlové závorky, tímhle krokem rychleji rozdělíme řetězec na jednotlivé prvky a ty pak uložíme do pole, ve kterém na 18. pozici dohledáme „Entering Germany“. Poté nám zbývá rozdělit dle mezery tento řetězec a nechat jenom „Germany“ jako návratovou hodnotu.

Pro lepší pochopení představme, že plánujeme cestu z „Prahy, Česko“ do „Calpe, Španělsko“. Takže po proběhnutí tohoto úryvku kódu (Obrázek 24), by náš list měl vypadat přibližně takto: Czechia (Česko), Germany (Německo), France (Francie) a Spain (Španělsko). Pak nám zbývá jen spočítat ujetou vzdálenost v každém státu, podle čeho

budeme počítat jak mýtné, tak i palivové výdaje. Jako vždy níže uvádím Obrázek 26 s úryvkem kódu pro počítání vzdálenosti mezi státy.

```
//This part of code is for calculate distance between countries if them are more than one.
for (int i = 0; i < countryOnWays.size(); i++) {
    setStartFinishPoints(i, countryOnWays);
    final int finalI = i;
    GoogleApiService apiService = GoogleRetroClient.getApiService();
    apiService.getJson( origin: start.latitude + "," + start.longitude,  destination: finish.latitude + "," + finish.longitude).enqueue(new Callback<DirectionResults>() {
        @Override
        public void onResponse(@NonNull Call<DirectionResults> call, @NonNull Response<DirectionResults> response) {
            if (response.body().getRoutes().size() != 0) {
                Route route = response.body().getRoutes().get(0);
                Legs legs = route.getLegsList().get(0);
                setCountryDistance(finalI, countryOnWays, legs);
            }
            showDialogResponse = true;
        }

        @Override
        public void onFailure(Call<DirectionResults> call, Throwable t) {
            showDialogResponse = false;
            ShowDialog.showProgressDialog( show: false, getString(R.string.make_trip_msg), mActivity);
            ShowDialog.showErrorDialog(getString(R.string.error_calculate_distance), getString(R.string.error_calculate_distance_msg), mActivity);

            Log.e( tag: "onFailure",  msg: "createPathsRoute: " + t);
        }
    });

    if (countryOnWays.size() - 1 != i)
        countriesResult.add(i, countryOnWays.get(i));
}
```

Obrázek 26 – Počítání vzdáleností mezi dvěma body (úryvek kódu).

Abychom mohli spočítat vzdálenost mezi dvěma body, které nám jsou představeny jako souřadnice zeměpisné délky a šířky, musíme se znovu obrátit na Google Maps API s novým požadavkem na výpočet trasy pro tyto dva body. Jeden z těch bodů je počáteční a druhý koncový. V minulých krocích algoritmu jsme se dostali k finálnímu listu se všemi body, teď jenom musíme správně zavolat každý z těch bodů a vypočítat mezi nimi vzdálenost. Pro správné volaní bodů, použijeme metodu, kterou uvádím níže jako Obrázek 27 s úryvkem kódu, který popíšu až pod tím.

```
private void setStartFinishPoints(int i, List<CountryOnWay> countries) {
    if (countries.size() - 1 != i) {
        start = getStartFinishLatLng(i, countries);
        finish = getStartFinishLatLng( i: i + 1, countries);
    } else {
        start = getStartFinishLatLng( i: i - 1, countries);
        finish = getStartFinishLatLng(i, countries);
    }
}
```

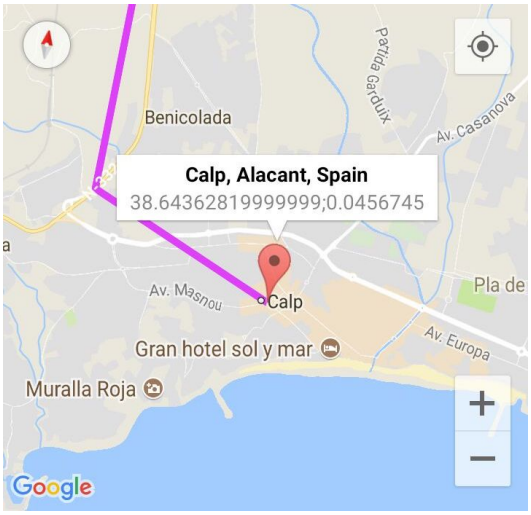
Obrázek 27 – Nastavování počátečních a koncových souřadnic (úryvek kódu).

Tento kód, který vidíte nahoře neděla nic těžkého. Jediné, co dělá, je, že nastavuje správné počáteční a koncové body, pro výpočet vzdálenosti mezi nimi. Což po proběhnutí tohoto úryvku kódu (Obrázek 26) by nám mělo nastavit u každého státu, kolik kilometrů jím budeme projíždět.

V našem případě by to mohlo vydat takhle:

1. Czechia (Česko): 171 Km
2. Germany (Německo): 519 Km
3. Francie (Francie): 870 Km
4. Spain (Španělsko): 632 Km.

Posledním krokem v algoritmu je nastavit takzvaný polylin, dříve jsem říkal, že z Google Maps API obdržíme objekt (overview_polyline) který obsahuje textový řetězec s kódovanou reprezentací křivky kroku. Nejprve musíme dekódovat tento řetězec, uděláme to za pomocí *PolyUtil.decode (řetězec)*. Pak musíme nastavit markery, osobně jsem vyřešil, že k markerům přidám ještě nějaké popisy, jako například: souřadnice a adresa. Níže je uveden Obrázek 28, který demonstruje, jak v mé aplikace vypadají markéry.



Obrázek 28 – Příklad markéru v mé aplikace.

Nakonec za pomocí tohoto úryvku kódu (Obrázek 29) vykreslíme polylin.

```
PolylineOptions polylineOptions = new PolylineOptions().
        geodesic(true).
        color(ContextCompat.getColor(mActivity, R.color.colorAccent)).
        width(10);

for (int i = 0; i < points.size(); i++)
    polylineOptions.add(points.get(i));

polylinePaths.add(mMap.addPolyline(polylineOptions));
```

Obrázek 29 – Vykreslování polylinu (úryvek kódu).

Algoritmus na tom končí, ale jsem si jistý, že jste nepochopili kde a jak rozpočítávám cestu. MapsActivity ovládá mapu a sbírá data, které pak pošle do DetailTripActivity. DetailTripActivity zase přeposílá tato data do RecyclerViewInfoTripAdapter, který provede jak nastavení vizuálního kontentu, tak výpočet všeho, co je potřeba dle uvedených vzorců.

Vím, že bych to mohl udělat trochu lepší, ale rozhodl jsem se, že to nechám pro bakalářskou práci v tomto stavu.

Než se pustíme do poslední části mé bakalářské práce, chtěl bych se podívat na vzorce pro rozpočet cesty.

Prvním v řadě je rozpočet paliva. Velmi jednoduchá věc. Uživatel předtím, než začne vytvářet svůj výlet přidá nebo vybere nějaké existující auto s nějakou průměrnou spotřebou na 100 kilometrů. Tuto spotřebu využijeme pří rozpočtu.
Vzorec vypadá takhle: *vzdálenost / 100 * průměrná spotřeba * cena paliva.*

Pro vypočet mýtného, raději uvedu níže Obrázek 30 jako vždy s popisem pod obrázkem.

```
private String calculateTollCosts(String tollCountry, String distance, String comment) {
    ///     getCountriesCostsApi(country.getCountry());
    double toll_cost = 0;
    if (comment.contains("per 100")) {
        toll_cost = Double.parseDouble(distance) / 100 * Double.parseDouble(tollCountry);
    } else if (comment.contains("price is average") || comment.contains("10 days") || comment.contains("weekly") || comment.contains("annual")) {
        toll_cost = Double.parseDouble(tollCountry);
    } else if (comment.contains("Tolls no pay only for one day") || comment.contains("Payment at tollgates") || comment.contains("No pay tolls")) {
        toll_cost = 0;
    }
    return String.valueOf(toll_cost);
}
```

Obrázek 30 – Vypočet mýta.

Pokud si pamatujete, každý stát v mém Countries Cost API, má řetězec s komentářem, takže tento komentář se nejenom zobrazuje uživateli v aplikaci, ale i za jeho pomoci můžu rozhodovat, který vzorec budu používat.

Máme několik různých typu komentářů:

1. Price per 100 km (Cena je za 100 km)
   toll_cost = vzdálenost */ 100 * mýtné ceny;*
2. Price is average (Cena je průměrná)
   toll_cost = mýtné ceny;
3. Minimal vignette is weekly (Minimální je 7-denní dálniční známka)
   toll_cost = mýtné ceny;
4. Minimal vignette is annual (Minimální je roční dálniční známka)
   toll_cost = mýtné ceny;
5. Minimal vignette is for 10 days (Minimální je 10-denní dálniční známka)
   toll_cost = mýtné ceny;
6. Tolls no pay only for one day (Mýtné se neplatí jenom pro první den)
   toll_cost = 0;
7. Tolls only for some bridges and tunnels (Mýtné jenom pro nějaké mosty nebo tunely)
   toll_cost = 0;
8. Payment at tollgates (Platba u mýtných bran)
   toll_cost = 0;
9. No pay tolls (Žádné mýtné)
   toll_cost = 0;

8 Implementace a popis aplikace

Poslední kapitola této bakalářské práce se bude zabývat jak popisem funkcí aplikace, tak i její implementační strukturou.

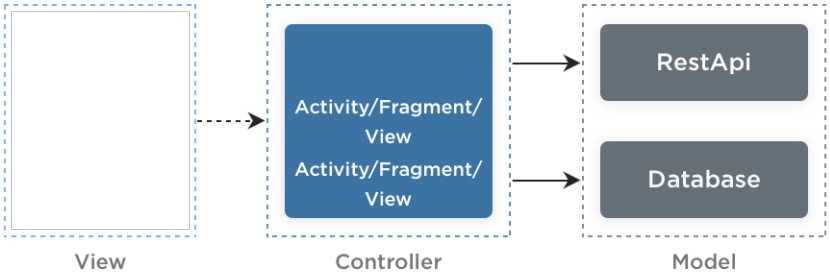
8.1 Implementace

Jak jsem říkal na začátku kapitoly „Návrh“, návrhový model by měl být takový, pomocí kterého lze rovnou začít implementovat části aplikace. Chtěl bych Vás upozornit, že před začátkem implementace aplikace bychom měli rozhodnout, zda budeme používat nějakou architekturu nebo ne. Protože je mnohem jednodušší dodržovat jediný styl práce na celém projektu.

8.1.1 Architektury pro androidové aplikace

Nejčastěji používané architektury (celá kapitola čerpá ze zdroje: [21]):

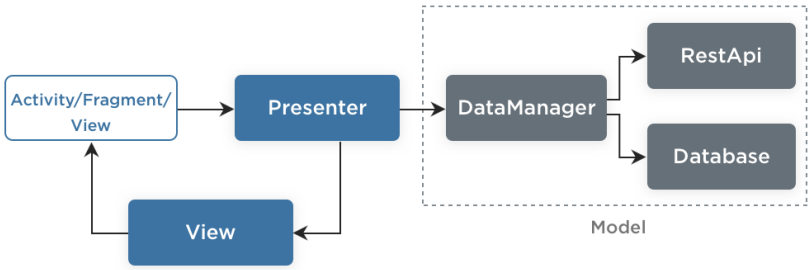
1. **MVC (Model View Controller)**



Obrázek 31 – MVC architektura. Zdroj: [21].

   a. **Model –** v architektuře MVC představuje Data + Stav + Business logiku. V podstatě je to mozek naší aplikace, který není svázaný ani s *View* ani s *Controllerem*, proto může být opakovaně použitelný v mnoha kontextech.
   b. **View –** je reprezentace *Modelu*, odpovědná za to, že se vykreslí uživatelské rozhraní (UI) a za komunikaci s *conrollerem* když uživatel interaguje s aplikací. V MVC architektuře je *View* obecně docela hloupý, nemá žádnou představu o *Modelu* a nerozumí stavu ani aktivitě po klepnutí uživatelem na tlačítko. Myšlenka spočívá v tom, že čím méně ví, tím volnější je spojení s *Modelem*.
   c. **Controller** – je lepidlo, které spojuje aplikaci dohromady. Když uživatel klepnul na tlačítko *View*, pošle zprávu controlleru, zatímco *controller* rozhodne, jakým způsobem by měl *Model* odpovídat. V případě aplikace pro Android je controller téměř vždy reprezentován aktivitou nebo fragmentem.

2. MVP (Model View Presenter)



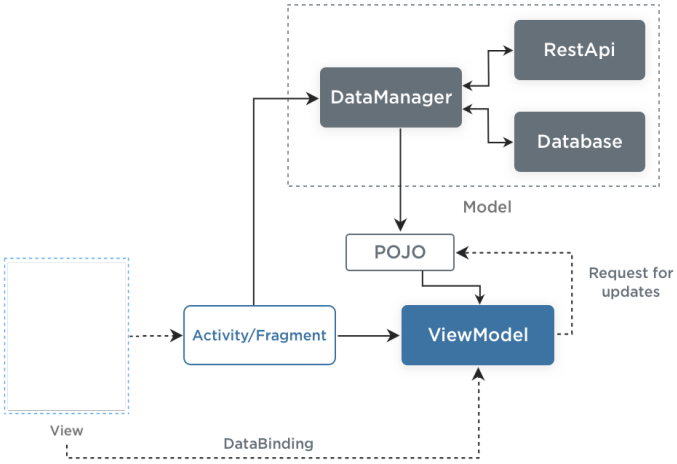
Obrázek 32 – MVP architektura. Zdroj: [21].

a. **Model –** stejné jako u architektury *MVC (Model View Controller)*.

b. **View –** Jedinou změnou je, že aktivita / fragment je nyní považován za součást View.

c. **Controller** – v podstatě se jedná o stejný controller *od MVC (Model View Controller),* s tím rozdílem, že není vázán na *View* – Jedinou změnou je, že aktivita / fragment je nyní považován za součást View., ale jenom na rozhraní.

3. MVVM (Model View ViewModel)



Obrázek 33 – MVVM architektura. Zdroj: [21].

a. **Model –** stejné jako u architektury *MVC (Model View Controller)*.
b. **View –** je kombinace aktivity / fragmentu s nějakém souborem. Údaje o požadavcích na aktivitu z „backend“, získávají data a předávají je do ViewModel.
c. **ViewModel** – je zodpovědný za zabalení modelu a přípravu pozorovatelných dat, potřebných pro zobrazení. Poskytuje také zachycení pro View, aby události byly předané modelu. ViewModel však není vazbou na View.

Existují i další architektury, které mají jako zaklad některé z těch, které jsem vypsal výše. Pro svou bakalářskou práce jsem nezvolil žádnou architekturu, ale při pracovních projektech se nikdy nevyhýbám architekturám, neboť přinášejí jenom benefity a to jak při testování tak i při návrhu jsou lepší na pochopení.

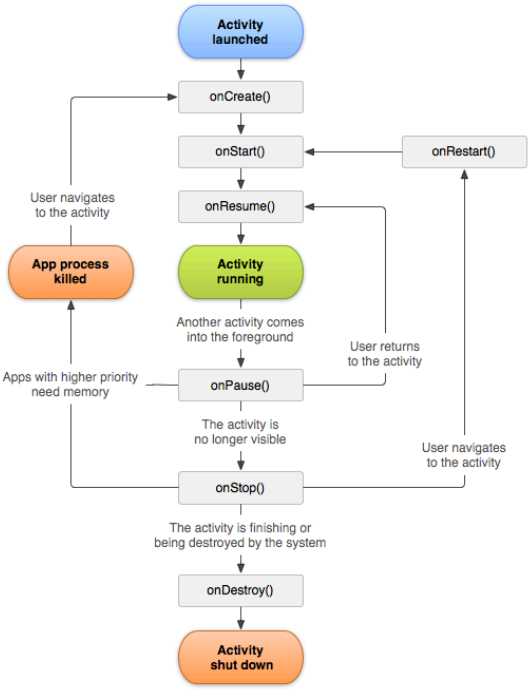
8.1.2 Activity & Fragment

Předtím než přejdeme k popisu aplikace, chtěl bych se zeptat, zda víte, jaký je rozdíl mezi Aktivitou a Fragmentem? Víte proč bychom měli kombinovat použití fragmentů s aktivitou? A jaké výhody mají fragmenty oproti Aktivitě?

Activity (Aktivita)

Aktivita je základní komponenta pro vykreslování grafického návrhu naší aplikace. Pro příklad, uživatel spustí mou aplikaci a první, čeho si všimne, je úvodní obrazovka s logem a to je naše aktivita.

Aktivita má svůj vlastní životní cyklus, který uvádím jako Obrázek níže.



Obrázek 34 – Životní cyklus Aktivity. Zdroj: [2].

Každá aktivita se nachází současně jen v jednom stavu: běžící (na popředí), pauza (aktivita je vidět, ale překryta jinou aktivitou, například s vyšší prioritou), zastavená (aktivita není vidět, ale uživatel se k ní ještě může vrátit), ukončená (úplně mrtvá aktivita) [2].

Metody životního cyklu:

1. **onCreate()**

   Pří spuštění aktivity se Android sám postará o vytvoření objektů a spuštění procesu, dále zavolá metodu *onCreate(),* která nadefinuje vše potřebné, aby se aktivita mohla rozjet.

2. **onStart()**

   Volá se, pokud byla aktivita poprvé spuštěna. Může se to stát jak po *onCreate(),* tak i pokud aktivita byla aktivovaná po svém skrytí (příchozí zprava, systémový dialog nebo dialog jiné aktivity).

3. **onResume()**

   Zavolá se těsně před tím, než aktivita bude posunuta do popředí (restart, první spuštění nebo po pauze).

4. **onPause()**

   Volá se před tím, než aktivita bude posunuta na pozadí a má pravomoc násilného ukončení aktivity.

5. **onStop()**

   Zavolá se, když se má aktivita zastavit.

6. **onDestroy()**

   Volá se před tím, než bude aktivita zrušena.

7. **onRestart()**

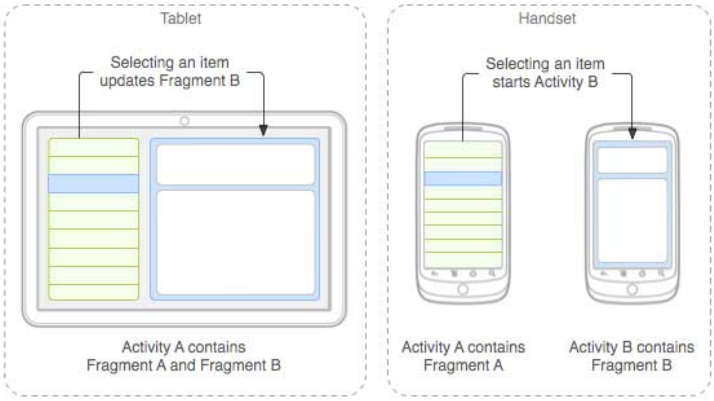
   Volá se poté, co byla zavolána metoda *onStop()* a aktivita se začala restartovat. Bude provedena těsně před *onStart()*.

Tyto metody jsou velmi užitečné, pokud je pochopíte a hlavně pokud pochopíte životní cyklus, tedy jak, kdy a co bude vyvoláno.

Aktivity jsou základním kamenem, bez kterého nejde nic udělat v androidovém vývoji. Dokonce jakmile přejdeme k fragmentům, dozvíte se, že fragmenty nemohou existovat bez aktivity.

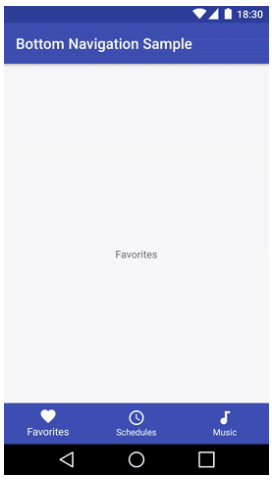
Fragment (Fragment)

Jak jsem si řekl minule, fragment nemůže existovat bez aktivity, je to její část. Fragmenty se nám poprvé objevily v Androidu 3.0 jako reakce na příchod podpory pro tablety. Už se ta aktivita nevydala hezky na deseti-palcovém displeji se stejným layoutem. Typické použití fragmentu je při tzv. master/detail flow, níže je uvedený Obrázek 35, který ukazuje, jak to funguje.



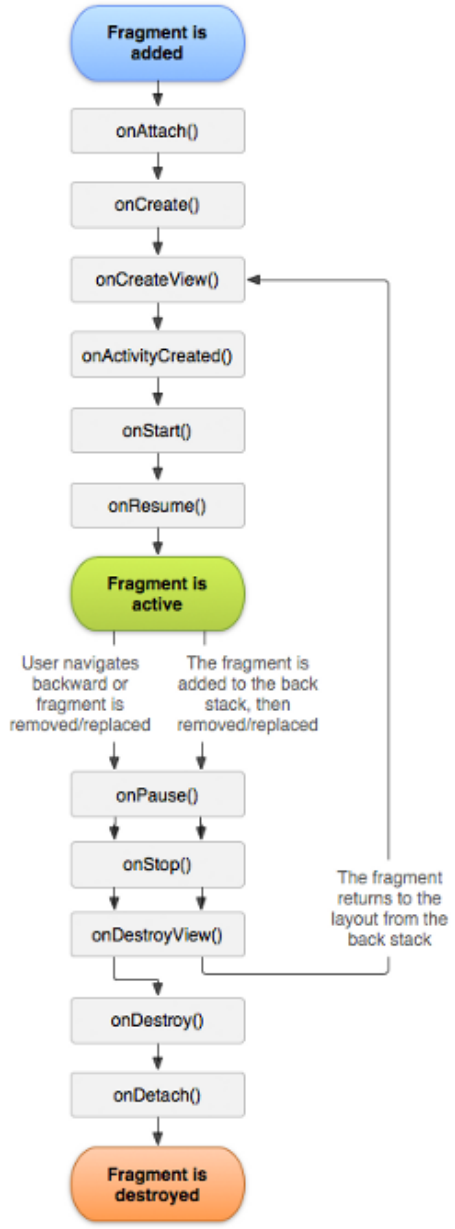
Obrázek 35 – Fragment použití pří master/detail flow. Zdroj: [2].

Ale fragmenty jde použít nejen v této situaci, těch situací jsou mraky, ale uvedl bych tu, se kterou se sám často střetávám. Potřebujeme udělat spodní navigaci v naší aplikaci (např.: profil, přidej, nastavení), udělal bych jednu aktivitu, která by řídila všechny tři fragmenty. Tímto krokem zjednodušíme naši aktivitu a překlikávání mezi položkami bude rychlejší. Níže je uvedený příklad takového menu s fragmenty.



Obrázek 36 – Spodní navigační menu s využitím fragmentů.

Jako aktivita, i fragment má svůj vlastní životní cyklus, který bych raději ukázal na obrázku níže.



Obrázek 37 – Životní cyklus Fragmentu. Zdroj: [2].

V podstatě tento životní cyklus je podobný jako i u aktivity.

Metody životního cyklu:

1. **onAttach()** – volá se jen jednou, když se spojí s aktivitou.
2. **onCreate()** – inicializace fragmentu.
3. **onCreateView()** – vytvoří a vrací hierarchii View
4. **onActivityCreated()** – je vyvolána po dokončení metody onCreate().
5. **onStart(), onResume() ,onPause() , onStop()** – stejné jak u aktivity.
6. **onDestroyView()** – umožňuje fragmentu vyčistit zdroje.
7. **onDestroy()** – stejné jak u aktivity.
8. **onDetach()** – je volaná těsně před tím, než se fragment odpojí od aktivity.

Tak co bychom měli používat v aplikaci? Docela dobrá otázka, na kterou, obávám se, nikdo nedá jednoznačnou odpověď. Ale dle mého názoru, jako všude, bychom měli používat všechno hlavou a nepoužívat v aplikaci jen fragmenty nebo jen aktivity, měli bychom to kombinovat a zároveň by to mělo dávat smysl.

8.2 Popis aplikace

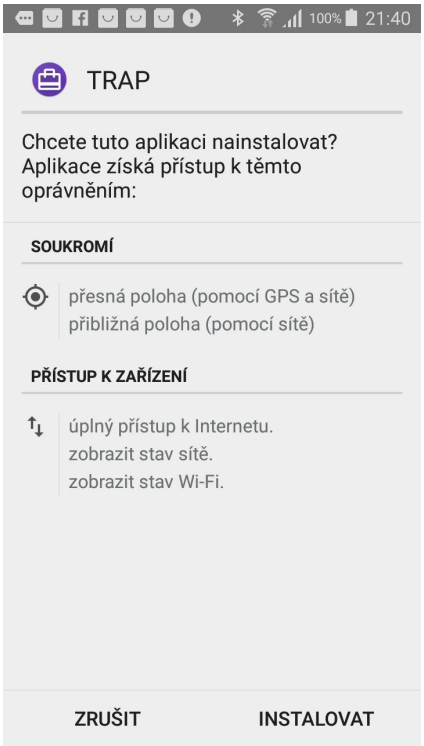
Vítám vás u poslední kapitoly mé bakalářské práce. Tato kapitola se bude zabývat funkčností mé aplikace ze strany uživatele. Zde už nenajdete žádný úryvek kódu, tato kapitola je jenom o uživatelské části aplikace.

Jak jsem psal na začátku, aplikace by měla být dobrou pomůckou pro rozpočet cesty na území Evropské Unie. Zároveň by aplikace měla být přívětivá, odolná proti chybám a rychlá pří použiti. Takže jsem se zaměřil jak na funkčnost, tak na uživatelskou část.

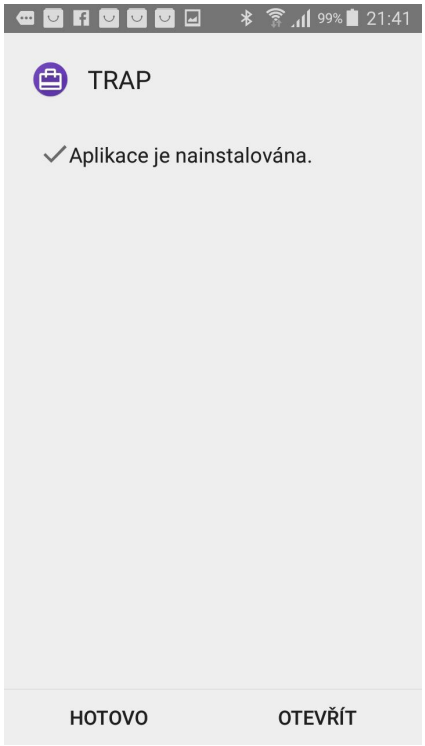
Aby to nebylo nudné, představte si, že jste uživatel, který se poprvé dostal do mé aplikace.

8.2.1 Instalace

Prvním krokem, co byste měl udělat, je nainstalovat mou aplikaci, při instalaci musíte povolit několik různých druhu oprávnění.



Obrázek 38 – Instalace oprávněni.

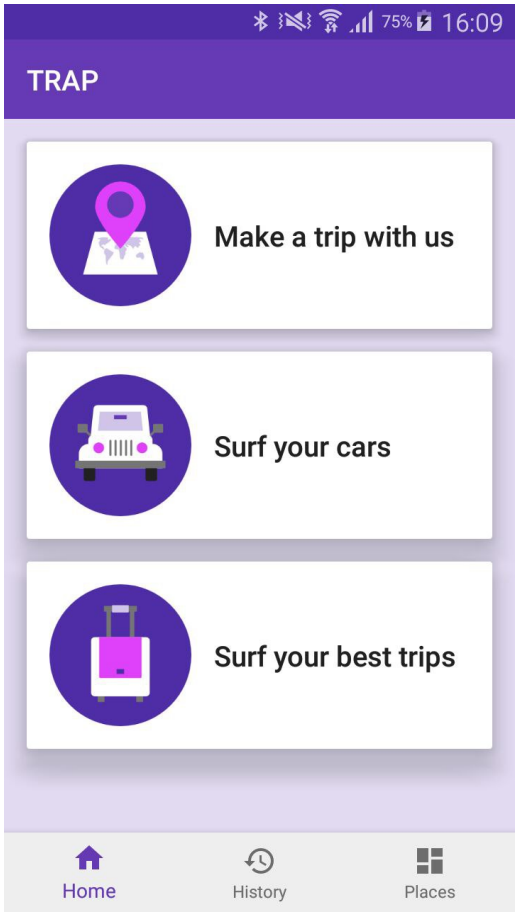


Obrázek 39 – Instalace úspěšná.

Pak si představte, kam byste chtěl jet s rodinou nebo sám, a vůbec se netrapte tím rozpočtem Vašeho výletu, o to se postará má aplikace.

8.2.2 Popis funkčnosti aplikace

Dalším krokem je první start aplikace, po spuštění se Vám zobrazí menu s nabídkou co byste měl a mohl udělat. Níže uvádím Obrázek 40, který ilustruje, jak vypadá hlavní menu v aplikaci. Pod tímto obrázkem si začneme popisovat každou jeho část.



Obrázek 40 – Hlavní menu aplikace.

První, co by se mělo líbit uživateli je, že tu nejsou mraky různých tlačítek a celkově nic složitého v tomto menu není. Což splňuje podmínku jak přívětivosti, tak i lehkosti při využiti. Projdeme postupně každé políčko v hlavním menu.

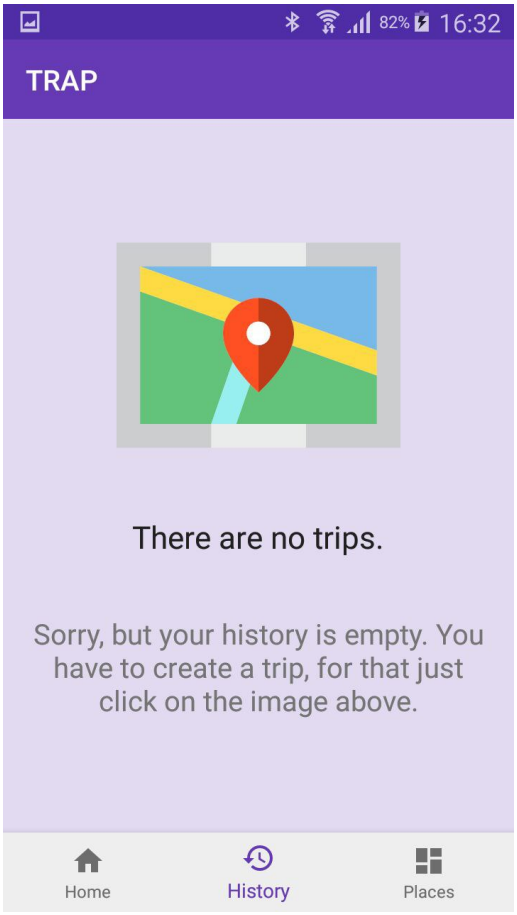
Začal bych tím spodním menu, které obsahuje takové pozice jako:

1. **Home (Domov)**

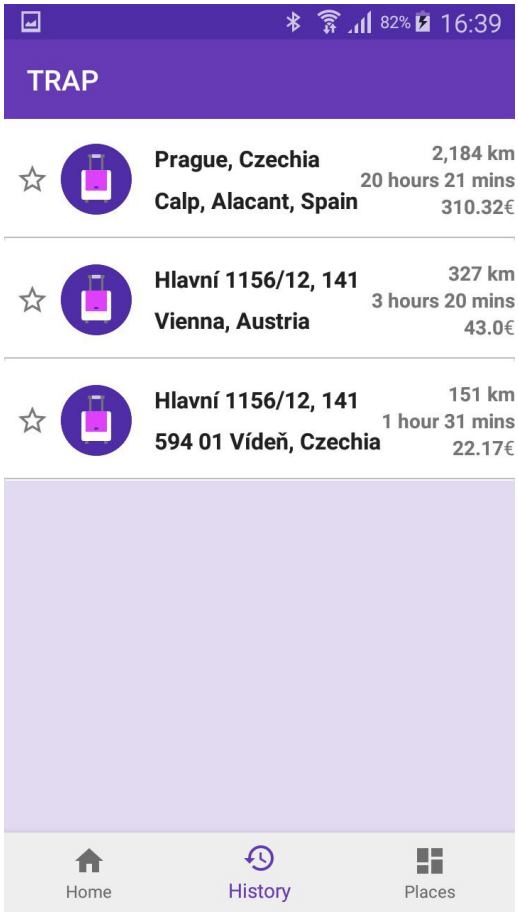
   Zjevně je pochopitelné, že tato pozice je hlavní menu. (Obrázek 40).

2. **History (Historie)**

   Zde se schovala historie Vašich výletů, pokud nějaké máte. Podívejte se, jak to vypadá na obrázcích, které jsou uvedeny níže.



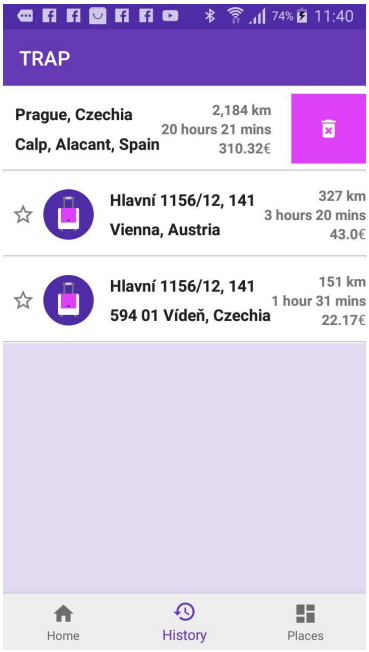
Obrázek 41 – Historie s prázdným seznamem.



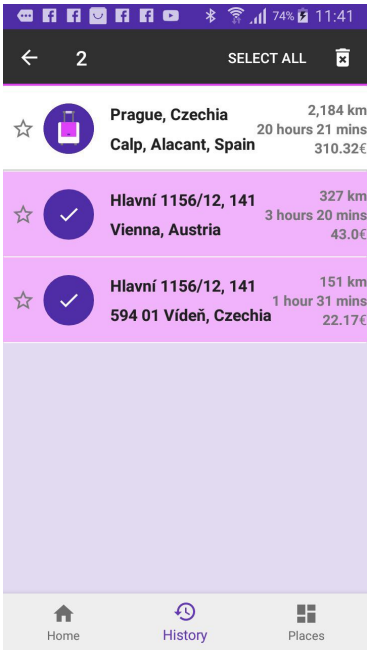
Obrázek 42 – Historie s n-tým počtem výletů.

Nahoře jsou uvedeny obrázky, které ukazují, jak se chová historie výletů ve dvou různých situacích. První situace je, pokud nemáte vytvořený ani jeden výlet a je to dobře vidět na Obrázek 41. Zde se objeví chybová hláška, že „Nejsou žádné výlety“ s podrobnějším popisem níže. Po přečtení chybové hlášky byste měl pochopit, že musíte vytvořit alespoň jeden výlet. Aby se Vám to povedlo, můžete buď klepnout na ikonu chybové hlášky, nebo přejít do *Home (Domov)* a vybrat políčko pro vytvoření výletu *Make trip with us (Vytvořte výlet spolu s námi)*.

Obrázek 42 nám představuje vyplněný seznam n-tým počtem Vašich výletů. Pokud budete chtít odebrat výlet, stačí jenom protáhnout prstem doleva a objeví se Vám ikona na odstraní vybraného výletu. Můžete si také zvolit více výletů k odstranění, aby se Vám to povedlo, udělejte buď dlouhé kliknutí na celý řádek nebo klepnutí na ikonu výletů. Níže uvádím dva obrázky, které zjevně ukážou, jak to bude vypadat v aplikaci. Obrázek 43 demonstruje případ s protahováním, zatím co Obrázek 44 ukazuje ten režim výběru.



Obrázek 43 – Odstranění výletů protahováním prstů.

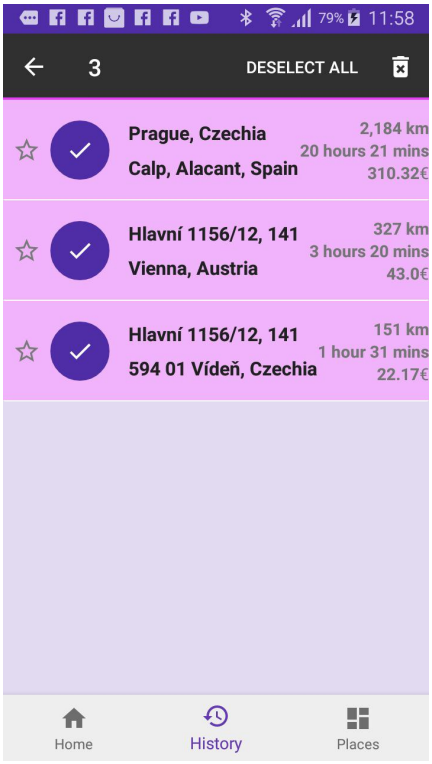


Obrázek 44 – Režim výběrů, pro více možností odstranění.

Určitě je pochopitelné, že byste mohl omylem odstranit něco, co jste nechtěl odstraňovat, proto v případě odstranění protahováním prstem, po odebraní výletu, se dole objeví takzvaný *„SnackBar“*, který obsahuje v pravém rohu tlačítko *„Undo“* neboli *„Vrátit“*.

Obrázek 45 – Vracení zpět odstraněného výletu.

V pravém rohu obrazovky (víz Obrázek 44) byste mohl všimnout tlačítko *„Select All“*, po kliknutí na něj budou automaticky označeny všechny Vaše výlety a budou připraveny k odstranění. Jakmile budou vybrány všechny výlety, text u tlačítka se změní na *„Deselect All“,* pokud byste zrušil výběr alespoň u jednoho výletu, text by se znovu nastavil na *„Select All“*. Níže je uvedený Obrázek 46, který reprezentuje situaci s úplným výběrem všech výletů.

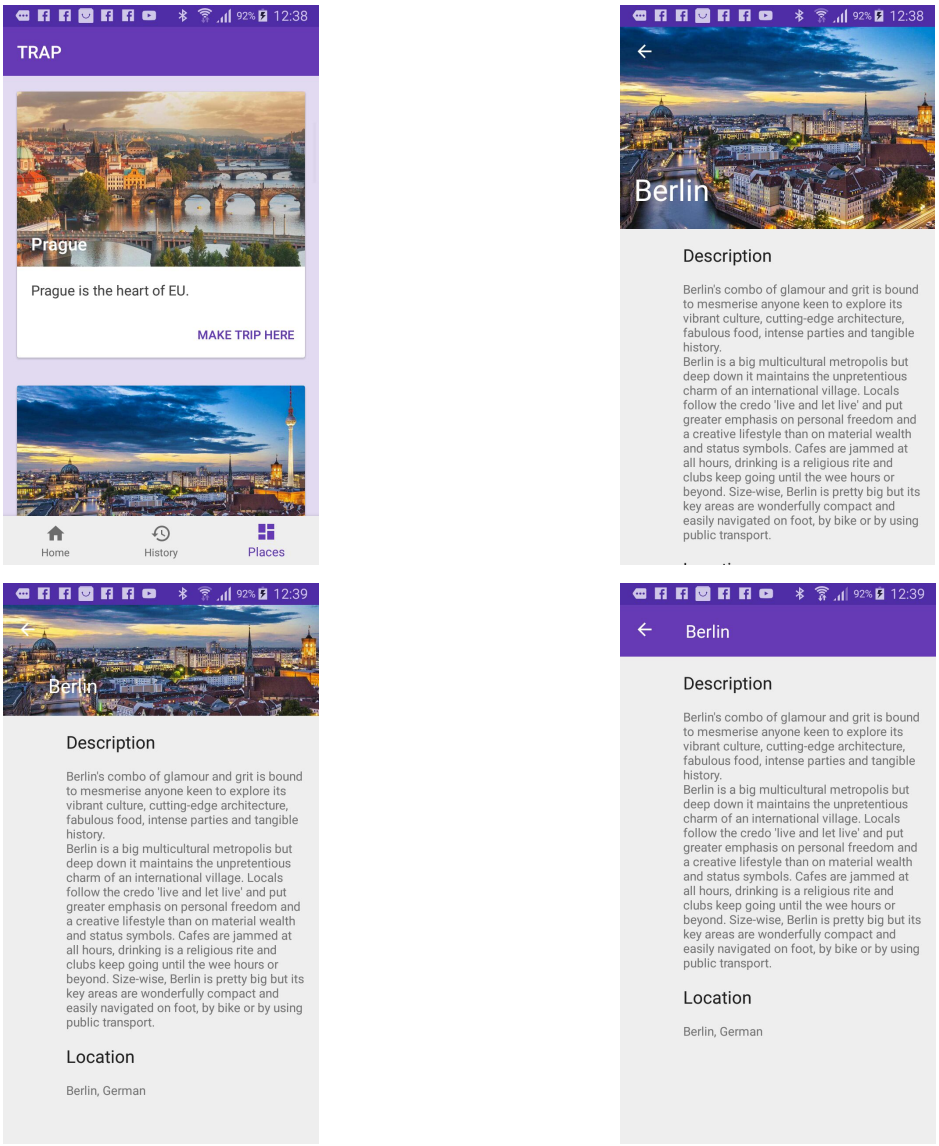


Obrázek 46 - Všichni výlety jsou zvolené.

Pokud chcete odstranit všechny vybrané výlety, stačí jenom kliknout na ikonu *„Koš“* v pravém horním rohu. Pak tyto výlety budou navždy vymazány.

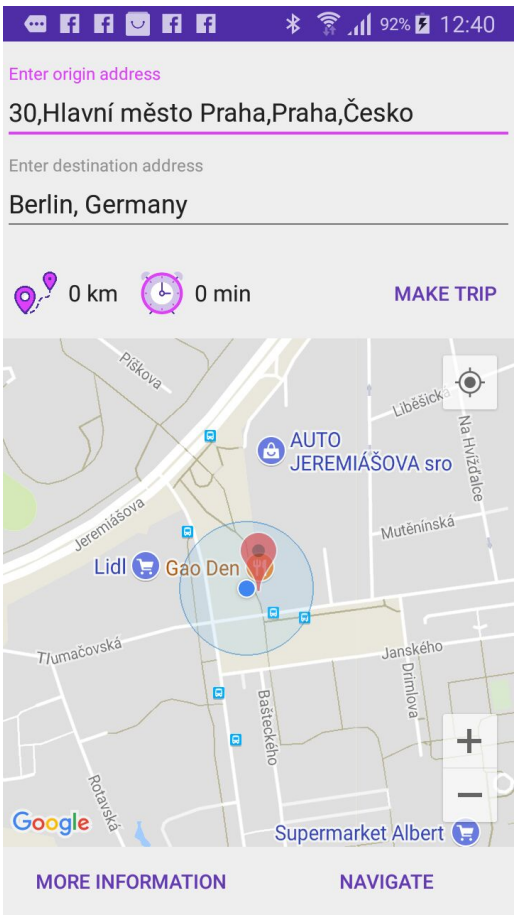
3. **Places (Místa)**

Tady najdete doporučená města pro výlety. Pro bakalářskou práci jsem si vytvořil vlastní API s omezeným počtem měst, ale pro distribuovanou verzi budu využívat buď Google APIs nebo i TripAdvisor API. Města jsou reprezentována jako karty, pokud se chcete dozvědět více o kartách, musíte se vrátit zpět do kapitoly Material Design. Karta obsahuje název města, malý komentář, obrázek na pozadí a tlačítko *„Make trip here (Vytvořte sem výlet)“*. Po stisknutí tlačítka budete přesměrován do rozhraní pro vytváření výletů, přičemž už budete mít předvyplněné údaje jak o konečném, tak i o počátečním bodě za využití umístění uživatele, pokud nebudete chtít, aby aplikace předvyplnila za Vás počáteční bod, můžete si ten bod zvolit sám. Níže uvádím skupinu obrázků a jako vždy popis najdete pod nimi.



Obrázek 47 - Skupina obrázku reprezentujících Places (Místa) v aplikace.

Výše jsem uvedl skupinu obrázků reprezentujících chování Places (Místa). Pro přiklad stisknete tlačítko *„Make trip here (Vytvořte sem výlet)“* u karty s Berlínem, aplikace Vás nasměruje do rozhraní pro vytváření výletů, níže uvádím Obrázek 48, který ilustruje, co se stane po kliknutí na toto tlačítko. Podrobnější popis tohoto rozhraní si můžete přečíst v kapitole *Make a trip with us (Vytvořte výlet spolu s námi)*.



Obrázek 48 – Po stisknuti tlačítka „Vytvoř cestu sem“.

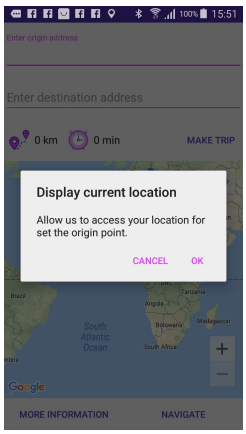
Vám pak jenom zbývá rozkliknout tlačítko *„Make trip (Vytvoř výlet)“*, vybrat si auto a počkat na výsledek rozpočtu. Pokračujte podle instrukcí, které Vám nabízí má aplikace a všechno bude v pořádku.

Vraťte se znovu do domovské časti a vrhneme se na naše hlavní menu. Probrali jsem *History (Historie)* a *Places (Místa)*. Teď se můžeme vrhnout na hlavní části mé aplikace.

Zkuste si vytvořit svůj první výlet. Klepnutím po kartě (*Make a trip with us (Vytvořte výlet spolu s námi)*), aplikace vás nasměruje do rozhraní pro vytváření výletu.

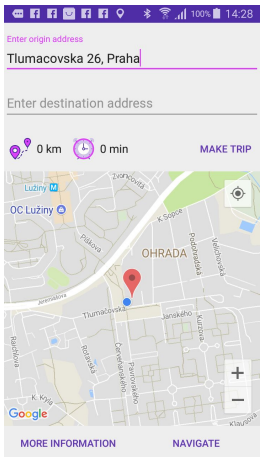
1. **Make a trip with us (Vytvořte výlet spolu s námi)**

   Po klepnuti na tuto kartu budete nasměrován do rozhraní pro vytváření výletu. Jakmile se tam dostanete, zobrazí se Vám dialogové okno s požadavkem na vejmutí Vašeho umístění pro nastavení počátečního bodu. Níže je Obrázek, který ilustruje tuto situaci.

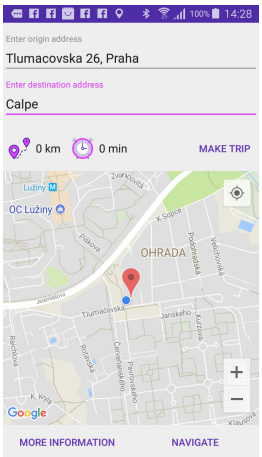


Obrázek 49 – Požadavek na využívání umístění pro nastavení počátečního bodu.

Zde máte jenom dvě cesty, buď to stornujete a necháváte počáteční adresu prázdnou, nebo povolíte aplikaci vyplnit ji za Vás dle Vašeho umístění. Řekněme, že jste rozhodl, že to povolíte, a dostáváte se do následujícího kroku. Už máte vyplněnou počáteční adresu a zbývá Vám vybrat místo, kam pojedete. Pro přiklad jste se rozhodl jet do Calpe, je to výborné město u mořského pobřeží ve Španělsku. Takže vyplňte do políčka s konečnou adresou *„Calpe“* a stiskněte tlačítko *„Make trip“*.

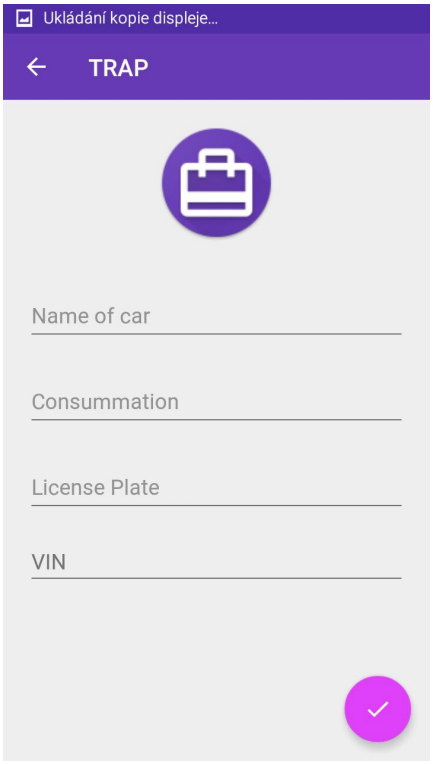


Obrázek 50 – Aplikace se zjistila umístění uživateli.

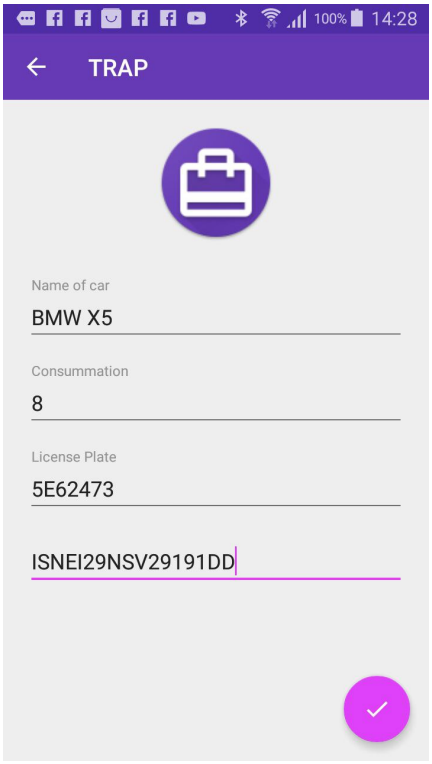


Obrázek 51 – Uživatel se nastavil konečnou adresu výletu.

Jelikož jste tady poprvé a předtím jste nikdy nevytvářel nějaký výlet, aplikace po Vás bude chtít, abyste přidal své auto, kterým pojedete. Takže aplikace Vás přesměruje do rozhraní pro vytváření auta.



Obrázek 52 – Před vyplněním uživatelského auta.



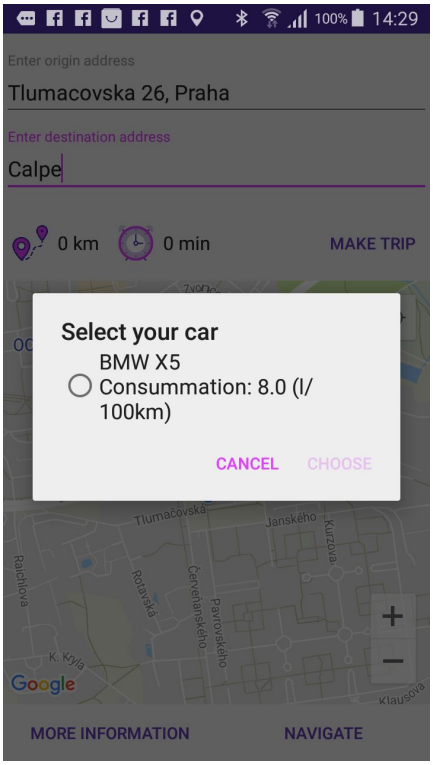
Obrázek 53 – Po vyplněním uživatelského auta.

Pak po přidání Vašeho auta se můžete vrátit zpět a znovu stisknout tlačítko *„Make trip“*. Už máte aspoň jedno auto, proto Vás aplikace pustí do dalšího kroku. Objeví se Vám dialogové okno se seznamem všech Vašich aut, pokud máte jedno auto, nic se neděje, dialog se objeví a musíte tedy zvolit auto, kterým budete cestovat během tohoto výletu. Je to zavedeno z několika různých důvodů:

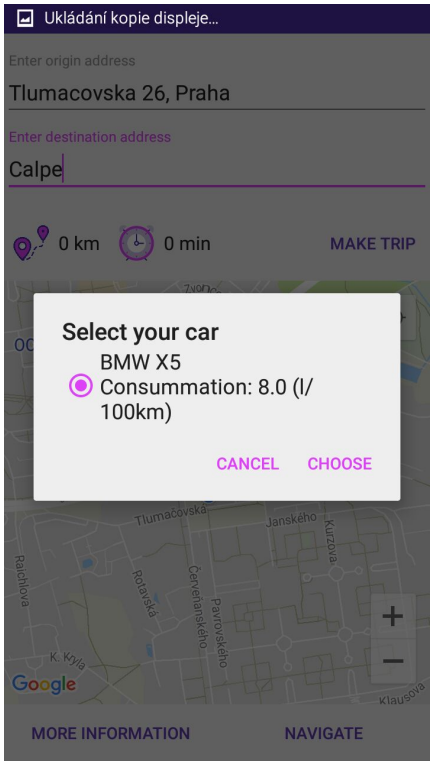
1. Někteří lidé vlastní několika různými auty a mohou cestovat libovolným z nich.
2. Uživatel může vyzkoušet auta na stejném výletu a prozkoumat, které z nich bude výhodnější.

Níže uvádím obrázky, které ukazují tento krok v aplikaci.

Z důvodů nedostatku místa Obrázek 54 a Obrázek 55 uvádím až na další stránce.

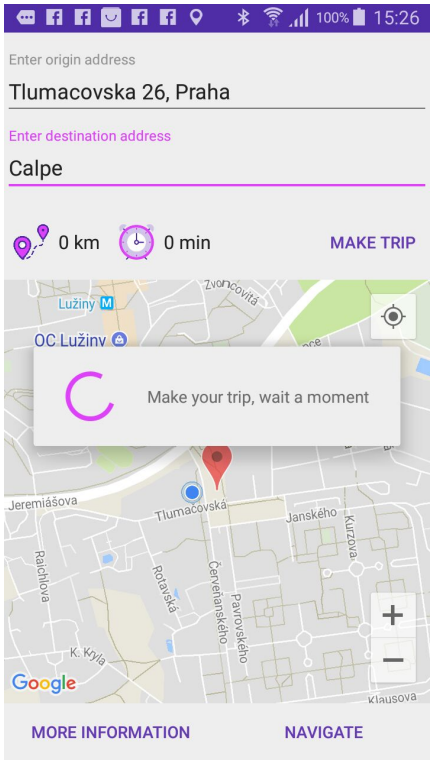


Obrázek 54 – Před zvolením auta pro nějaký výlet.



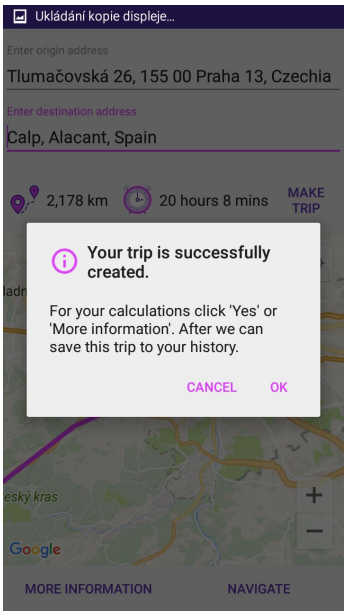
Obrázek 55 – Po zvolením auta pro nějaký výlet.

Jakmile potvrdíte Vámi zvolené auto, aplikace začne sestavovat Vaši cestu. Zobrazí se Vám tedy takové dialogové okno, které uvádím níže jako Obrázek 56.



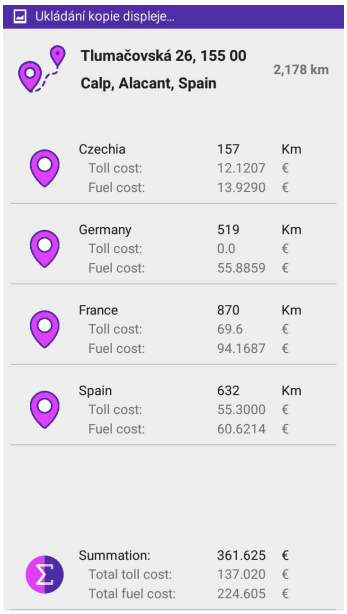
Obrázek 56 – Načítací dialogové okno.

Jakmile bude cesta připravena, zobrazí se dialogové okno s otázkou, jestli náhodou teď nechcete spočítat cestu a uložit ji do historii, pokud nechcete, nic se neděje, stačí stisknout tlačítko *„More information“* a aplikace Vám to spočítá a uloží do historie. Níže je uveden Obrázek 57, který reprezentuji ten dialog.

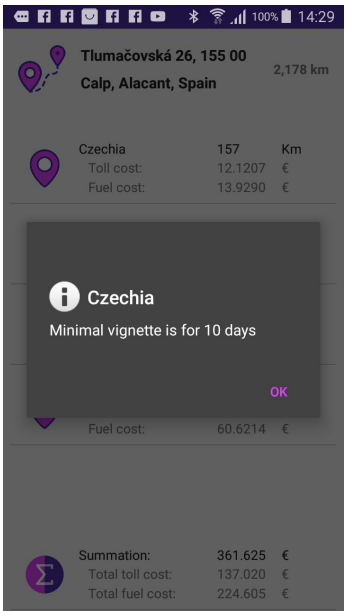


Obrázek 57 – Výlet byl přepraven k rozpočtu.

Představme si, že jste se rozhodl spočítat to rovnou a odpověděl jste *„Yes“* / *„Ok“*. Aplikace načte další rozhraní, kde se provede rozpočet a zobrazí se Vám data.



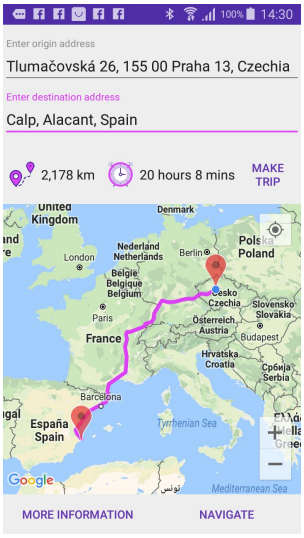
Obrázek 58 – Rozpočet cesty.



Obrázek 59 – Více informace o země.

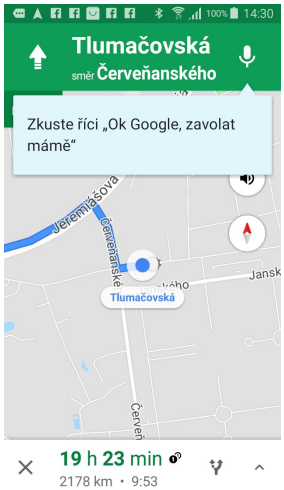
Obrázek 58 demonstruje výsledky z rozpočtu cesty. Dále pokud byste se chtěl dozvědět více informací o jednotlivém státu, pomocí klepnutí na vybraný řádek se Vám objeví dialogové okno s pokročilejšími informacemi, jako je uvedeno na Obrázek 59, že v Česku je minimálně deseti-denní dálniční známka.

Nakonec jsme se dostali do finále. Už byste měl mít finální rozpočet cesty jako já, ale na tom ještě funkčnost mé aplikace nekončí, proto se vrátíme o jeden krok zpět. Před Vámi by měla být stejná obrazovka co je uvedená níže, na *Obrázku 60*.



Obrázek 60 – Pohled z vysoka na vytvořenou cestu.

Má aplikace není navigátor, proto mám níže udělené speciální tlačítko na přesměrování uživatele do Google Maps. Aplikace předá Google Mapě všechna potřebná data pro sestavení cesty (počáteční a koneční bod). Zkuste na to kliknout, níže uvádím obrázek, který ilustruje, co se stane.



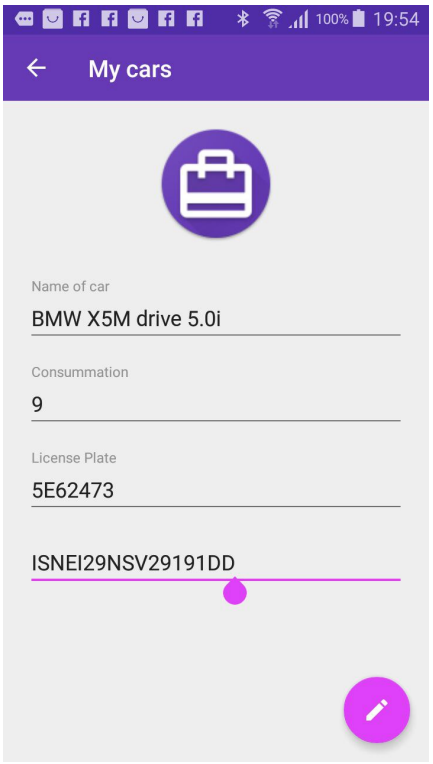
Obrázek 61 – Přesměrování do Google Mapy.

Takže si myslím, že u této části bychom mohli skončit a přejit k jiným. Vezmeme to postupně.

2. **Surf your cars (Surfujte své auta)**

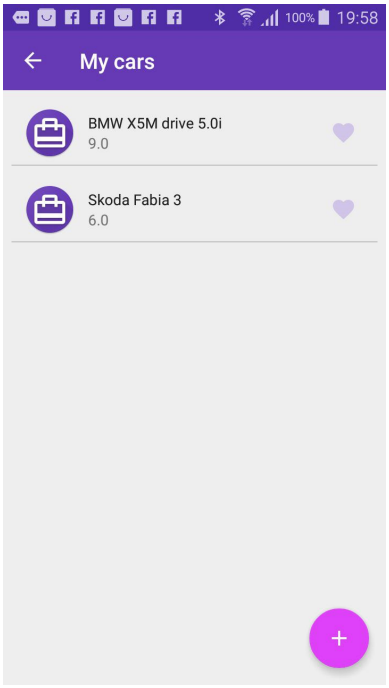
   Po klepnutí na kartu *Surf your cars (Surfujte své auta)*, se dostáváte do části, kde můžete přidávat, odebírat a editovat svá auta. Pokud bude seznam aut prázdný, zobrazí se Vám automaticky formulář pro přidání auta. Jakmile máte alespoň jedno auto, objeví se Vám seznam. Odstraňovat můžete protahováním prstem jak vpravo, tak i vlevo, dlouhým kliknutím se dá změnit pořadí mezi auty, to se však neuloží. Takže vpravo je srdce, které bohužel v bakalářské verzi nefunguje. V distribuované verzi fungují tyto věci úplně stejně jak v historii. Navíc to srdce funguje takovým způsobem, že po jeho klepnutí se auto přidá do nejoblíbenějších. Toto auto bude mít prioritu před ostatními auty a bude vždy v topu.

   Zkuste přidat nějaké nové auto a editovat již existující auto. Klepněte na řádek s vybraném autem a aplikace Vás nasměruje do nového rozhraní pro editaci auta. Níže je Obrázek 62, který reprezentuje rozhraní pro editovaní auta.



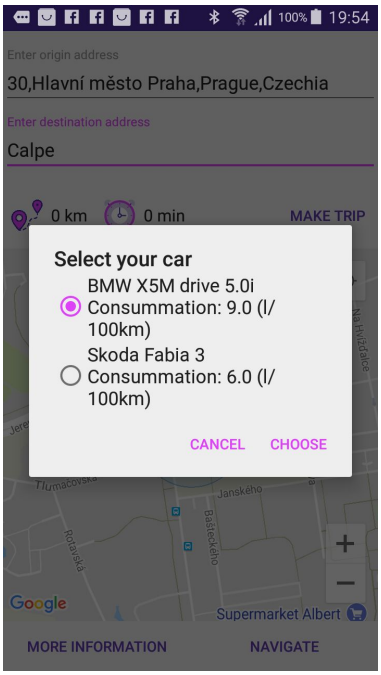
Obrázek 62 – Editace auta.

Udělejte nějaké změny a uložte je za pomoci plovoucího tlačítka v pravém dolním rohu. Po uložení budete přesměrován zpět do seznamu aut. Níže je Obrázek 63, který reprezentuje seznam aut v aplikaci.



Obrázek 63 – Seznam existujících aut uživatele.

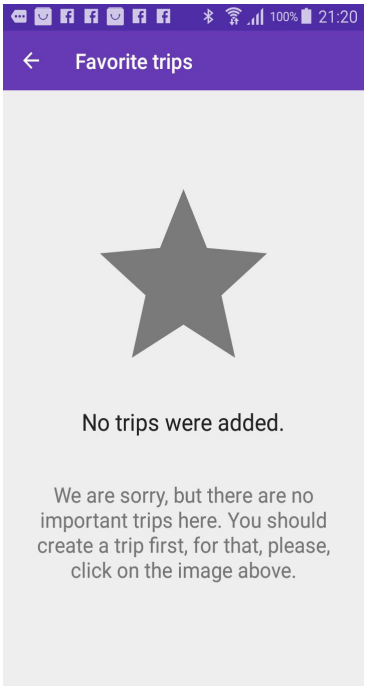
Už máte přidáno víc než jenom jedno auto, proto si při vytváření dalšího výletu budete moci vybrat. Níže uvádím Obrázek 64, který ilustruje, jak bude vypadat dialogové okno pro výběr auta, pokud máte více aut.



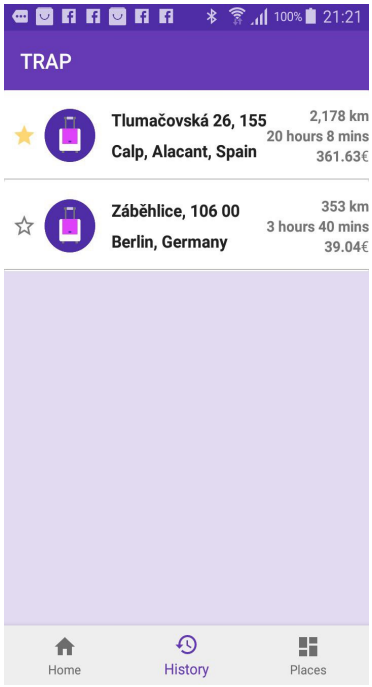
Obrázek 64 – Dialogové okno pro výběr aut.

3. **Surf your best trips (Surfujte své nejlepší výlety)**

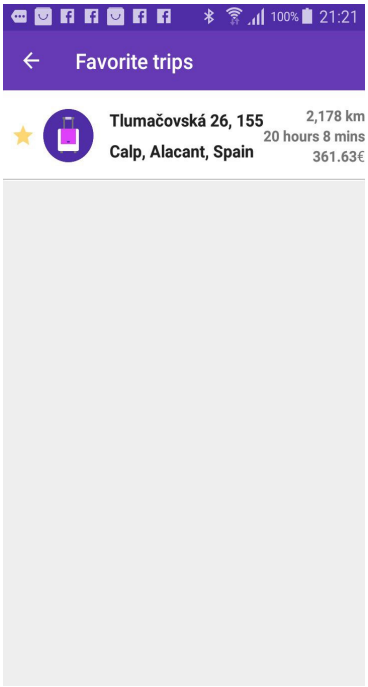
Zde můžete uchovávat Vaše nejoblíbenější výlety. Jestli seznam nejoblíbenějších výletů je prázdný, zobrazí se Vám chybová hláška s podrobným popisem. Pokud budete sledovat hlášku, nebudete mít žádný problém naučit se přidávat výlety do nejoblíbenějších, jedná se totiž jen o jediné klepnutí. Níže uvedu sadů obrázků, které popíšu až pod nimi.



Obrázek 65 – Prázdný seznam nejoblíbenějších výletů.



Obrázek 66 – Označení výletu, jako nejoblíbenější.



Obrázek 67 – Není prázdný seznam nejoblíbenějších výletů.

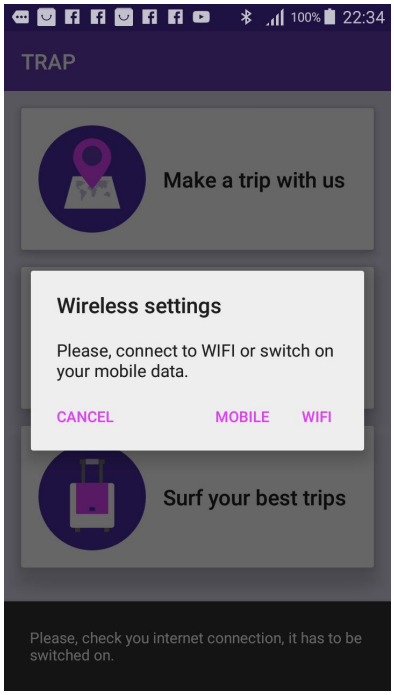
Jestli do seznamu nejoblíbenějších výletů nebyl přidán žádný výlet, je seznam prázdný a po klepnutí na kartu *Surf your best trips (Surfujte své nejlepší výlety)* se Vám objeví chybová hláška, kterou reprezentuje Obrázek 65. Po přečtení chybové hlášky byste měl přijít na to, že musíte označit alespoň jeden z výletů jako oblíbený za pomocí hvězdy, pro ukázku jsem uvedl Obrázek 66. Obrázek 67 reprezentuje seznam nejoblíbenějších výletů, které jste označil v historie.

Důvodem zavedení této funkce je to, aby uživatel neprocházel mraky výletů v historii pro hledaní svého nejoblíbenějšího výletu. Stačí ho jen přidat do tohoto seznamu a pak je snadné ho tu najít.

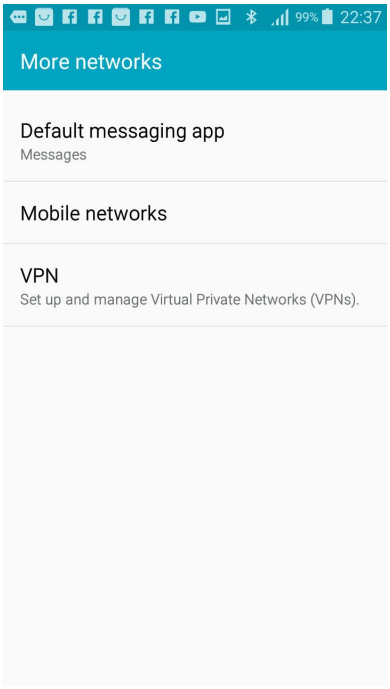
8.2.3 Ošetření chyb v aplikaci

Pro některé časti mé aplikace je potřeba mít stabilní internet a zapnuté umístění, proto musím vždy sledovat zda má uživatel zapnuté internetové spojení a umístění. Jelikož tohle je jen bakalářská práce, nezavedl jsem tyto kontroly ve všech částech aplikace. Pro tuto práce jsem se rozhodl, že to budu sledovat na horních úrovních, což znamená, že nebudu sledovat každou aktivitu v té aplikaci. Představte si, že se Vám vypl WIFI nebo došly mobilní data, když jste používal mou aplikace. Teď pokuste otevřít *Places (Místa)*., *Home (Domovská)* nebo *Make a trip with us (Vytvořte výlet spolu s námi)*.

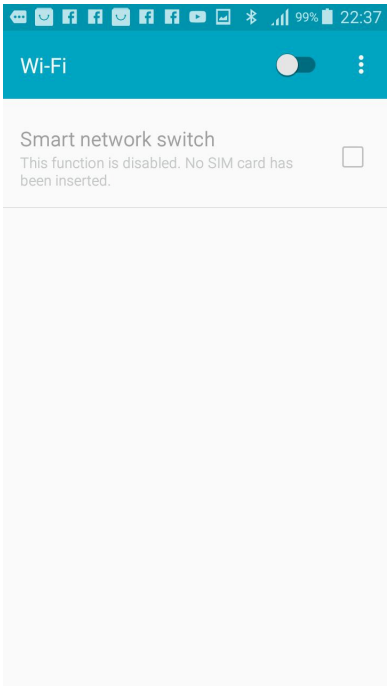
Níže uvádím sadů obrázku, které ukážou, jak se aplikace chová, pokud nemá internetové připojení.



Obrázek 68 – Chybová hláška internetového připojení.

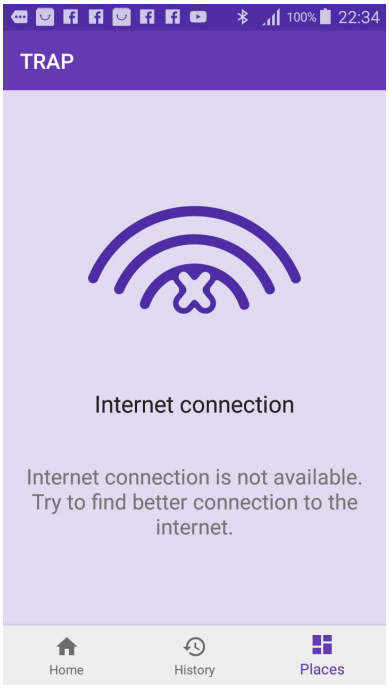


Obrázek 69 – Přesměrování po stisknutí tlačítka „Mobile“.



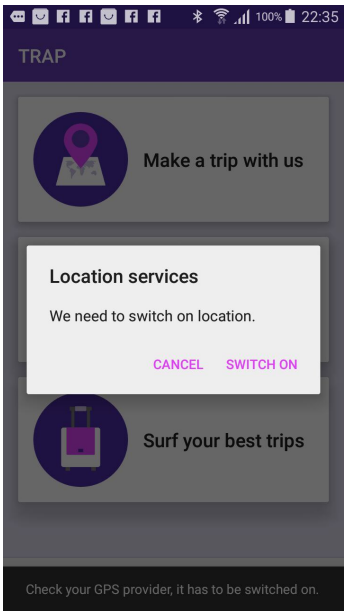
Obrázek 70 – Přesměrování po stisknutí tlačítka „WIFI“.

Nahoře jsou uvedené obrázky, které ukazují chování aplikace, pokud nejste připojen k internetovým službám. Aplikace Vás donutí připojit se buď za pomoci mobilních dat (Obrázek 69 – Přesměrování po stisknutí tlačítka „Mobile“.) nebo WIFI (Obrázek 70). Po připojení k internetu, budete moci pracovat dál. Pokud zkusíte vstoupit bez internetového připojení do *Places (Místa)*., objeví se Vám jiný typ chybové hlášky. Tento typ nepoužívá dialogová okna, ale pozadí obrazovky, aby se tam zobrazila chybová hláška. Příklad této chyby uvádím jako Obrázek 71 až na další straně, z důvodů nedostatku místa na této stránce.

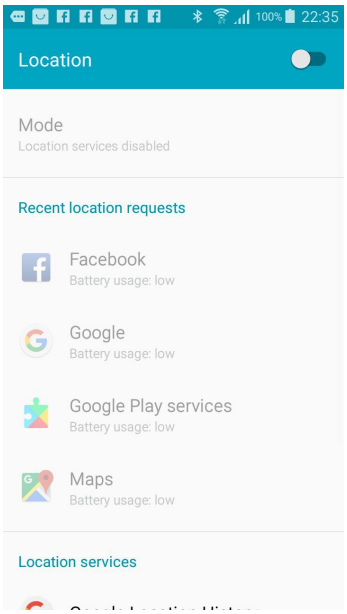


Obrázek 71 – Chybová hláska na pozadí obrazovky.

A teď se pokuste vypnout umístění na svém androidu. Po vypnutí umístění se Vám nepodaří dostat jenom do části, kde se vytváří výlet, všude jinde můžete surfovat. Níže uvádím sadu obrázků, které ukáží, jak se aplikace chová, pokud není zapnuté umístění.



Obrázek 72 – Chybová hláška není zapnutého umístění.



Obrázek 73 - Přesměrování po stisknutí tlačítka „Switch on“.

Aplikace Vás donutí zapnout umístění, v opačném případě se Vám vůbec nepodaří vytvořit ani jeden výlet. Po zapnutí umístění budete moci pokračovat dále ve vytváření Vašeho výletu.

9 Závěr

Hlavním cílem této bakalářské práce bylo vytvořit funkční, uživatelsky přívětivou, lehkou při použití androidovou aplikaci. Tato aplikace by do budoucna měla ušetřit hodně času lidem při plánovaní výletů autem. Teď ji předělávám a budu mít kompletně jiný kód a architekturu aplikace. Distribuovaná verze by neměla být omezena jenom na území EU, takže bude mnohem silněji ošetřena proti chybám, ale zůstane lehká při použití.

Tato aplikace byla napsaná v jazyce Java a využívá několik druhů nejznámějších knihoven ve světě androidu. Snažil jsem se dodržovat standardy Material Design, což není úplně lehké. Takže si myslím, že se mi podařilo vymyslet docela dobrý design aplikace, určitě pro distribuovanou verzi bych to raději probral s profesionálním designerem.

Výsledná aplikace byla otestovaná jak na různých verzích, tak i velikostech androidů. Je tam občas malý nedostatek ohledně počítání, ale předělám to jenom v distribuované verzi. Takže určitě není úplně ošetřena proti chybám a není kompletně dokončena z pohledu funkcionality, ale splňuje zadaní mé bakalářské práce.

Nejsložitějšími body této práce jsou: vymyšlení a rozvíjení nápadu a procházení všemi kroky jako Analýza a návrh aplikace. Při návrhu a analýze aplikace bylo zjevně pochopitelné, že tato aplikace nebude tak jednoduchá na vytváření. Ale opravdu jsem chtěl vytvořit tuto aplikaci a alespoň nějak pomoci jak sobě, tak i lidem v okolí.

Tuto aplikaci jsem vytvářel, když jsem ještě nepracoval jako Medior Android Developer v jedné z největších a nejúspěšnějších firem v Praze, a to nejen dle mého názoru. V průběhu několika měsíců v práci jsem dostal dost nových informací, které jsem převedl do znalosti, proto si teď můžu dovolit s novými znalostmi se vrhnout na novou verzi této aplikace. Doufám, že do několika měsíců bude tato aplikace pomáhat lidem pří sestavování a rozpočtu cesty.

POUŽITÁ LITERATURA

[1] Smartphone OS Market Share. *Smartphone OS Market Share* [online]. IDC Research, Inc: IDC Research, 2016 [cit. 2018-02-27]. Dostupné z: https://www.idc.com/promo/smartphone-market-share

[2] Android Developers. *Android Developers* [online]. Google: Google, 2018 [cit. 2018-05-07]. Dostupné z: https://developer.android.com

[3] Android Application Development. *Android Application Development* [online]. Tutorials Point Pvt. Ltd: Tutorials Point Pvt., 2014 [cit. 2018-02-27]. Dostupné z: https://www.tutorialspoint.com/android/index.htm

[4] Save Data using SQLite. *Save Data using SQLite* [online]. Google: Google, 2018 [cit. 2018-02-27]. Dostupné z: https://developer.android.com/training/data-storage/sqlite.html

[5] SCHILDT, Herbert. *Mistrovství - Java*. Brno: Computer Press, 2014. [cit. 2018-02-27]. Mistrovství. ISBN 9788025141458.

[6] SIERRA, Kathy. a Bert. BATES. *Head first Java*. 2nd ed. Sebastopol, CA: O'Reilly, 2005. ISBN 0596009208.

[7] CLIFTON, Ian G. *Android user interface design: implementing material design for developers*. Second edition. New York: Addison-Wesley, 2016. [cit. 2018-02-27]. ISBN 978-0134191409.

[8] TIOBE Programming Community. *TIOBE Programming Community* [online]. TIOBE Software BV: TIOBE Software BV, 2013 [cit. 2018-02-27]. Dostupné z: https://www.tiobe.com/tiobe-index/

[9] XAMARIN Cross-platform Language. *XAMARIN Cross-platform Language* [online]. Xamarin Inc: Xamarin, 2018 [cit. 2018-02-27]. Dostupné z: https://www.xamarin.com

[10] DAZEINFO Apple Inc. VS Google Inc. *DAZEINFO Apple Inc. VS Google Inc* [online]. Dazeinfo Media & Research Private Limited: Dazeinfo Media & Research Private Limited, 2018 [cit. 2018-02-27]. Dostupné z: https://dazeinfo.com/2014/08/20/apple-inc-aapl-ios-8-google-inc-googl-android-l-growth-difference-ui-ux/

[11] MD Material Design. *MD Material Design* [online]. Google Inc: Google, 2018 [cit. 2018-02-28]. Dostupné z: https://material.io

[12] SQLite.org. *SQLite.org* [online]. SQLite Inc: SQLite, 2018 [cit. 2018-02-28]. Dostupné z: https://www.sqlite.org/aff_short.html

[13] Oracle – základní pojmy. *Oracle – základní pojmy* [online]. MICHELL: MICHELL, 2018 [cit. 2018-03-03]. Dostupné z: http://itblok.bastler.cz/subdom/itblok/oracle-zakladni-pojmy/

[14] Úvod do SQL. IT Book. *Úvod do SQL. IT Book* [online]. FSVSB: FSVSB, 2018 [cit. 2018-03-03]. Dostupné z: http://books.fs.vsb.cz/SQLReference/Sadovski/SQL-PRVN.HTM

[15] Úvod do PL/SQL. *Úvod do PL/SQL* [online]. IT4KT: IT4KT, 2018 [cit. 2018-03-03]. Dostupné z: http://it4kt.cnl.sk/c/dbs/student/10.html

[16] MySQL. *MySQL* [online]. Oracle Corporation: Oracle Corporation, 2018 [cit. 2018-03-03]. Dostupné z: https://www.mysql.com

[17] GPS Traffic Speedcam Route Planner by ViaMichelin. *GPS Traffic Speedcam Route Planner by ViaMichelin* [online]. Michelin: Google Commerce, 2018 [cit. 2018-03-03]. Dostupné z: https://play.google.com/store/apps/details?id=com.viamichelin.android.viamichelinmobile

[18] ViaMichelin. *ViaMichelin* [online]. Michelin: Google Commerce, 2018 [cit. 2018-03-03]. Dostupné z: https://play.google.com/store/apps/details?id=com.viamichelin.android.viamichelinmobile

[19] ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. [cit. 2018-03-04]. ISBN 978-80-251-1503-9.

[20] Using Retrofit 2.x as REST client - Tutorial. *Vogela* [online]. Lars Vogel, Simon Scholz, David Weiser: Lars Vogel, Simon Scholz, David Weiser, 2017 [cit. 2018-03-07]. Dostupné z: http://www.vogella.com/tutorials/Retrofit/article.html

[21] The MVC, MVP, and MVVM Smackdown. *Academy Realm* [online]. Realm: Realm, 2018 [cit. 2018-03-11]. Dostupné z: https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/

[22] Simform, MVS,MVP,MVVM Android. *Academy Realm* [online]. Simform LLC: Simform, 2017 [cit. 2018-03-11]. Dostupné z: https://www.simform.com/mvc-mvp-mvvm-android-app-development/