# Vývoj mobilní aplikace pro sledování polohy vozů veřejné dopravy ve Zlínském kraji

Petr Jelínek

Bakalářská práce
2022



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Petr Jelínek** |
| Osobní číslo: | **A19045** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Softwarové inženýrství** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Vývoj mobilní aplikace pro sledování polohy vozů veřejné dopravy ve Zlínském kraji** |
| Téma práce anglicky: | **Mobile Application Development for Monitoring of Public Transport-Vehicles Location in the Zlín Region** |

## Zásady pro vypracování

1. Vypracujte literární rešerši na téma vývoje mobilních aplikací, včetně technologií, které hodláte při vývoji použít.
2. Navrhněte mobilní aplikaci pro sledování polohy vozů veřejné dopravy ve Zlínském kraji.
3. Naprogramujte mobilní aplikaci dle návrhu.
4. Ověřte funkčnost aplikace.
5. Mobilní aplikaci a postup jejího vývoje dostatečně popište.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:


1. JEMEROV, Dmitry a Svetlana ISAKOVA. Kotlin in action. Shelter Island, NY: Manning Publications Co., [2017]. ISBN 9781617293290.
2. WALLS, Craig. Spring Boot in action. Shelter Island, NY: Manning Publications, [2016]. ISBN 9781617292545.
3. MARTIN, Robert C. Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall, c2009. ISBN 9780132350884.
4. MARTIN, Robert C. Clean architecture: a craftsman's guide to software structure and design. London, England: Prentice Hall, [2018]. ISBN 0134494164.
5. FOWLER, Martin. Refaktoring: zlepšení existujícího kódu. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0299-1.


Vedoucí bakalářské práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

**prof. Mgr. Roman Jašek, Ph.D., DBA** v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

**Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 4.4.2022

Petr Jelínek v. r.
podpis studenta

## ABSTRAKT

Hlavním cílem této bakalářské práce je přispět k digitalizaci veřejné dopravy ve Zlínském kraji (regionální linky, MHD Uherské Hradiště a MHD Zlín) a vytvořit mobilní aplikaci pro dva hlavní mobilní operační systémy Android a iOS. Tato aplikace bude uživatelům jasně a přehledně prezentovat data o vozidlech. Konkrétně se jedná např. o polohu na mapě zjištěnou na základě dat z GPS modulů z palubního počítače vozidel, přesné zpoždění vozů, jízdní řády a vizualizaci polohy a také virtuální odjezdové a příjezdové tabule pro vybranou zastávku.



Klíčová slova: Android, iOS, Flutter, widget, veřejná doprava, mobilní aplikace, mobilní zařízení



## ABSTRACT

The main objective of this bachelor thesis is to contribute to the digitalization of the public transport in the Zlin region (regional lines, public transport in Uherské Hradiště and Zlin) and to create a mobile application for the two main mobile operating systems Android and iOS. The mobile application will provide a clear and user-friendly way of presenting data and information about public transport vehicles. For example, the location of a vehicle on the map based on the onboard GPS module, accurate delays, timetables and visualization of the current location, and virtual departure and arrival boards for the selected stop.



Keywords: Android, iOS, Flutter, widget, public transport, mobile application, mobile device

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce panu Ing. Tomášovi Vogeltanzovi, Ph.D. za jeho čas, ochotu, cenné rady, vedení a celkovou pomoc při psaní této práce a také odborníkovi na dopravu a mému kamarádovi Ing. Janovi Kolaříkovi za jeho odbornou pomoc s doménou veřejné dopravy.

# OBSAH

# ÚVOD

Hlavním cílem této bakalářské práce je myšlenka nějakým způsobem pomoci digitalizovat a modernizovat veřejnou dopravu ve Zlínském kraji a doručit uživatelům hezkou a přehlednou aplikaci, která by jasně a zřetelně informace a data o veřejné dopravě prezentovala.

Zlínský kraj má totiž poměrně dost zajímavých, a ne plně využitých dat o veřejné dopravě (regionálních linkách a MHD Uherské Hradiště) jako například jejich aktuální polohu z GPS modulů ve vozidlech, přesné zpoždění vozů, informace o poslední navštívené zastávce, jízdní řády a mnoho dalšího. Motivem této bakalářské práce je tedy vytvořit mobilní aplikaci, která bude zpracovaná a naformátovaná data prezentovat uživatelům. Aplikace bude obsahovat mimo regionální linky také data a informace o vozidlech MHD Zlín, aby aplikace uživatelům poskytla ještě více užitečných informací na jednom místě a nemuseli tak používat více aplikací pro jednu věc.

Aplikace nabídne uživatelům možnost sledování polohy vozů veřejné dopravy ve Zlínském kraji. Budou si moci vybrat, zdali chtějí sledovat pouze vozy linek MHD Zlín nebo vozy regionálních linek, včetně MHD Uherské Hradiště. U jednotlivých vozů budou zobrazeny informace o samotném vozidle, jeho přesné zpoždění, cílová a poslední navštívená stanice a jízdní řád vybrané linky, který bude navíc vizualizovat aktuální polohu mezi zastávkami. Aplikace nabídne také filtrování a hledání vozů přímo na mapě na základě linky a typu vozidla.

Další funkcí v aplikaci budou virtuální odjezdové tabule, ve kterých si mohou uživatelé zvolit, zda chtějí zobrazit odjezdy či příjezdy pro zvolenou zastávku/stanici. Ve výsledcích se budou promítat informace o vozidle, jeho přesné zpoždění, jízdní řád a pokud bude vozidlo dostupné na mapě, tak také možnost zobrazit jeho aktuální polohu přímo na mapě. Dále zde bude možnost hledat a filtrovat vozidla podle jejich destinace/původní stanice.

Samotná aplikace nabídne také možnost vybrat si ze dvou barevných režimů – tmavý a světlý, které se následně ukládají do uživatelských preferencí. Celá aplikace je tvořena v Material Design stylu, který nabídne uživatelům známé a povědomé prostředí z jiných aplikací, převážně od firmy Google, což ji bude dělat více intuitivní a jednoduchou pro používání.

# I. TEORETICKÁ ČÁST

# 1 FRONTEND – POUŽITÉ TECHNOLOGIE

## 1.1 Flutter

Flutter je relativně nový open source framework pro tvorbu multiplatformních aplikací vyvíjený primárně Googlem. Ve Flutteru je možné vytvořit za pomocí stejného zdrojového kódu aplikaci, kterou lze distribuovat na všechny hlavní platformy – iOS, Android a web. Tento přístup má velkou výhodu hlavně v tom, že není nutné udržovat repositář s kódem pro každou platformu zvlášť, ale stačí pouze jedna tzv. code base. Nejedná se však o hybridní řešení, jako je například Ionic, který zabalí aplikaci napsanou v JavaScriptu do jakéhosi "webového obalu", což je v podstatě forma webového prohlížeče a samotná aplikace se chová jako webová stránka. Kód, který je napsaný ve Flutteru, se kompiluje přímo do nativního kódu pro danou platformu.

Samotný framework, což je sada několika nástrojů, využívá programovací jazyk, taktéž vyvíjený Googlem, který se jmenuje Dart. Syntaxe tohoto jazyka je velmi podobná například JavaScriptu, Javě anebo C#, takže samotný přechod a učení tohoto jazyka je velmi rychlé a jednoduché, pokud má vývojář zkušenosti s vývojem ve dříve zmíněných jazycích nebo vývojem obecně [1].

### 1.1.1 Ahead-of-time kompilace

Ahead-of-time (dále AOT) kompilace je proces, kdy překladač překládá kód, který je napsaný ve vyšších programovacích jazycích (Java, C#, Dart), případně zpracovaný mezi-jazyk z virtuálního stroje (bytecode), do nativního strojového kódu pro danou platformu tak, aby mohl být spuštěn nativně, tzn. využíval instrukce a optimalizace specifické pro danou platformu. Flutter využívá AOT kompilátor dart2native. Kód z tohoto kompilátoru má velkou výhodu oproti kódu, který byl zkompilován např. v Just-in-time kompilátoru (dále JIT), má totiž mnohem svižnější a rychlejší start samotné aplikace a stabilnější výkon. Oproti tomu, kód z JIT kompilátoru má o něco pomalejší start, nicméně jeho výhodou je právě optimalizace za běhu programu, kdy se snaží předvídat, které instrukce budou v programu používané nejčastěji [1][2].

### 1.1.2 Vykreslování komponent

Velkou výhodou jazyka Dart a potažmo i Flutteru je jeho schopnost ovládat každý pixel na obrazovce zařízení. Na rozdíl od ostatních frameworků pro vývoj mobilních aplikací, kde se vývojáři musí spoléhat na operační systém, který vykreslí komponenty a samotné uživatelské rozhraní na obrazovku, Flutter pracuje tak, že požádá operační systém o nativní canvas (plátno), na který následně sám vykresluje komponenty. To umožní Flutteru vykreslit například nativní komponenty pro iOS na zařízení s operačním systémem Android a naopak, což by jinak nebylo možné. K vykreslování používá open source grafickou knihovnu Skia, která poskytuje API, jenž má podporu na různých hardwarových a softwarových platformách. Díky Flutteru je možné vytvořit aplikaci, která bude vypadat na všech zařízeních a operačních systémech identicky, nicméně je samozřejmě také možné vzhled pro danou platformu více specifikovat. Vzhledem k tomu, že je Flutter stále ještě v poměrně rané fázi, často se objeví chyby spojené s vykreslováním, například některé widgety nejsou úplně 1:1 podobné k těm nativním nebo vykreslování animací není vždy plynulé a tak dále [3].

### 1.1.3 Widgety

Hlavní myšlenka Flutter frameworku je ta, že všechno je widget – ať už se jedná o jednoduchý text anebo složité karty či listy. Základní stavební kámen každé Flutter aplikace je widget, který může mít vzhled jednoho ze dvou předdefinovaných stylů – Material a Cupertino style [1].

Material Design je jakýsi designový jazyk vytvořený Googlem, který ho používá ve svých aplikacích napříč všemi službami a zařízeními. Styl se rozšířil i mezi další vývojáře a dnes se hojně používá na webech a v aplikacích, a to nejen na těch, které jsou spojeny s Googlem. Tento styl je typický svými responzivními animacemi, přechody, využití světel, stínů a mnoho dalšího [4].

Cupertino styl (iOS-style) je na druhou stranu styl nativních widgetů, které byly vytvořeny společností Apple a jsou hojně používány na jejich systémech, aplikacích a službách díky poskytovanému API pro vývojáře [5].

### 1.1.4 Stateless a Stateful widgety

Ve Flutteru je možné tvořit aplikaci ze dvou typů widgetů – `Stateless` a `Stateful`, z nichž má každý odlišné využití. Oba typy widgetů obsahují hlavní `build` metodu, která se volá vždy, když widget vstoupí do view (widget je viditelný na obrazovce). Do této metody je nutné psát veškerou prezentační logiku spojenou s uživatelským rozhraním a případně zde volat menší privátní metody, pokud je do nich kód rozdělen [1].

#### *1.1.4.1 Stateless widget*

`Stateless`, neboli bezstavový widget, se používá tehdy, pokud v budoucnu nebude nutné aktualizovat jeho vlastnosti a tím pádem ani překreslovat view (pohled) uživateli. Je tedy vhodný pro malé, statické widgety, které nebudou nikdy měnit svůj obsah na základě nějaké události. Metoda `build` tohoto widgetu je většinou volána jen jednou, a to hned po vytvoření widgetu a vložení do widget tree.

Pro vytvoření tohoto widgetu je nutné vytvořit vlastní třídu, která bude rozšiřovat abstraktní třídu `StatelessWidget` definovanou v knihovnách Flutteru a zároveň také přepsat `build` metodu, definovanou v této abstraktní třídě (Obrázek 1 - Vytvoření Stateless widgetu). Velkou výhodou moderních vývojových prostředí je to, že vývojář nemusí vždy tento celý kód psát manuálně. K vytvoření stačí použít zkratky pro generování těchto úryvků kódu, ve kterých se potom pouze mění název vlastní třídy a vše je připraveno k implementaci [6].

```
class Example extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

Obrázek 1 - Vytvoření Stateless widgetu

#### *1.1.4.2 Stateful widget*

`Stateful` widget, neboli widget se stavem, se na druhou stranu používá tehdy, pokud bude v budoucnu potřeba dynamicky překreslovat view (pohled) a dát tak uživateli vědět, že nastala změna (např. se vrátila odpověď ze serveru, na kterou uživatel čekal). `Stateful` widget obsahuje `State` objekt, na kterém se volají metody pro překreslení view, pokud se nějakým způsobem změní jeho vnitřní stav.

`Stateful` widget se vytváří velmi podobným způsobem jako `Stateless` widget s tím rozdílem, že je zde nutné vytvořit také `State` objekt, který se vytváří tak, že v samotném widgetu se zavolá metoda `createState`, která vytvoří a vrátí instanci třídy rozšiřující `State`, definovanou pod samotným widgetem (Obrázek 2 - Vytvoření Stateful widgetu). Generování tohoto kódu je díky zkratkám ve vývojovém prostředí opět hodně zjednodušené a není třeba ho psát stále znovu [7].

```
class Example extends StatefulWidget {

  @override
  State<Example> createState() => _ExampleState();
}

class _ExampleState extends State<Example> {

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

Obrázek 2 - Vytvoření Stateful widgetu

#### *1.1.4.3 Metoda pro nastavování stavu*

Metoda pro nastavování stavu, tedy `setState` se používá tehdy, pokud chceme dát frameworku vědět, že je nutné překreslit view (pohled) a uživateli tak zobrazit změnu. Voláním této metody tedy informujeme framework o tom, že se změnil vnitřní stav `State` objektu tak, že by mohl mít vliv na uživatelské rozhraní, což způsobí to, že se v blízké době, až to bude možné, celý widget tree (stromová struktura, která reprezentuje všechny widgety a závislosti mezi nimi v aktuálním kontextu) znovu sestaví a překreslí na obrazovce [1].

Metoda `setState` se používá v kombinaci se `Stateful` widgetem, který obsahuje i `State` objekt, tudíž si drží svůj vnitřní stav. Po zavolání této metody se zavolá samotná `build` metoda, která celý widget vykreslí znova a tím reflektuje nové změny uživateli.

### 1.1.5 Funkce Hot Reload

Při vývoji aplikace ve Flutteru je možné využívat funkci Hot Reload, která poměrně hodně zjednoduší a zrychlí celý vývoj. Tato funkce umožní promítnout změny v kódu za běhu celé aplikace bez nutnosti restartu. Pokud se například provede drobná změna (úprava textu či jiný odstín barvy) a soubor se uloží, na simulátoru, případně reálném zařízení, jsou provedené změny viditelné ihned, bez nutnosti celou aplikaci restartovat a znovu sestavovat [8]. Tato funkce funguje na principu injektování změněného zdrojového kódu do běžícího Dart Virtuálního Stroje (VM). Poté, co VM aktualizuje třídy s novými verzemi a změnami v kódu, framework znovu automaticky vytvoří celý strom widgetů, což umožní vývojáři vidět změny na simulátoru či na reálném zařízení.

### 1.1.6 Simulátory a zařízení

Díky tomu, že je Flutter multiplatformní framework, je taktéž dobré aplikaci na těchto platformách ladit. První možnost je samozřejmě využití reálného zařízení, na kterém po připojení do počítače lze aplikaci spustit a ladit ji v reálném čase. Se zařízeními s operačním systémem Android není problém jak na Windows, tak na macOS, problém však nastává s telefony od Apple, u kterých je nutné, aby měl vývojář vývojářskou licenci od Applu, díky které je potom umožněno vyvíjenou aplikaci ladit na reálných zařízeních a využívat další výhody, které tato licence přináší. Nutno také zmínit, že ladit aplikace mířené na iOS platformu lze pouze na zařízeních s operačním systémem macOS.

Druhá varianta je možnost využít simulátorů (Obrázek 3 - Android a iOS simulátory). Pokud bude vyvíjená aplikace dostupná na obě hlavní platformy, bude pravděpodobně třeba, aby vývojář vlastnil počítač s operačním systémem macOS od Apple, protože simulátory iPhonů a iPadů jsou dostupné v rámci vývojového prostředí XCode, které je dostupné pouze na macOS. Výhodou je to, že simulátory Android zařízení jsou dostupné jak na Windows, tak na macOS, tudíž je možné ladit Android aplikaci i na macOS. Díky vlastní zkušenosti mohu potvrdit zjištění, že simulátory iPhonů jsou mnohem více odladěné, plynulé a pracují mnohem svižněji než simulátory telefonů s operačním systémem Android [1].

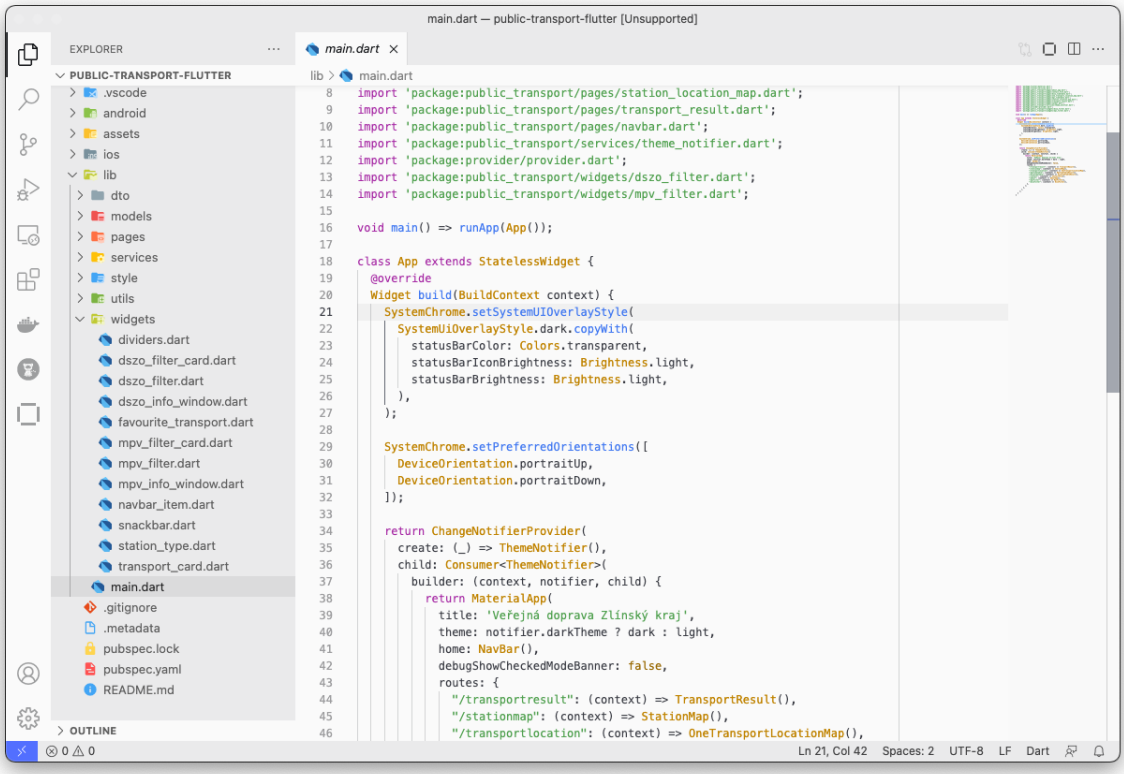

Obrázek 3 - Android a iOS simulátory

### 1.1.7 Vývojové prostředí

Vyvíjet aplikaci za použití Flutter frameworku je dneska možné ve velkém množství vývojových prostředí, případně editorů, a je tedy pouze na vývojáři a na jeho preferencích, které si pro vývoj zvolí. Většina z vývojových prostředí a editorů už přímo podporuje i Flutter, takže se sestavená aplikace pouze nainstaluje do simulátoru a přes prostředí lze celý proces řídit (využívání funkce Hot Reload, automatická instalace nových verzí aplikace atd.). Mezi nejpopulárnější a nepoužívanější prostředí pro vývoj se řadí následující.

#### *1.1.7.1 Visual Studio Code*

Visual Studio Code, zkráceně VS Code, (Obrázek 4 - Visual Studio Code) je spíše textový editor, než o IDE (Integrated Development Environment – vývojové prostředí). Dneska je mezi vývojáři hojně využíván právě tento editor, a to hlavně z důvodu jeho jednoduchosti a možnosti instalace mnoha doplňků, které udělají z tohoto editoru opravdu silný a efektivní nástroj pro vývoj. Oproti ostatním IDE je VS Code velmi odlehčený a rychlý, což je jedna z jeho hlavních předností. Editor je primárně vyvíjený společností Microsoft, nicméně má otevřený zdrojový kód, takže může přispívat kdokoliv z komunity vývojářů [9]. Tento editor mimo Flutter podporuje také velké množství dalších jazyků a frameworků. V dnešní době je asi nejvíce využíván pro vývoj frontend webových aplikací například v Reactu, Angularu, Svelte anebo třeba Vue.

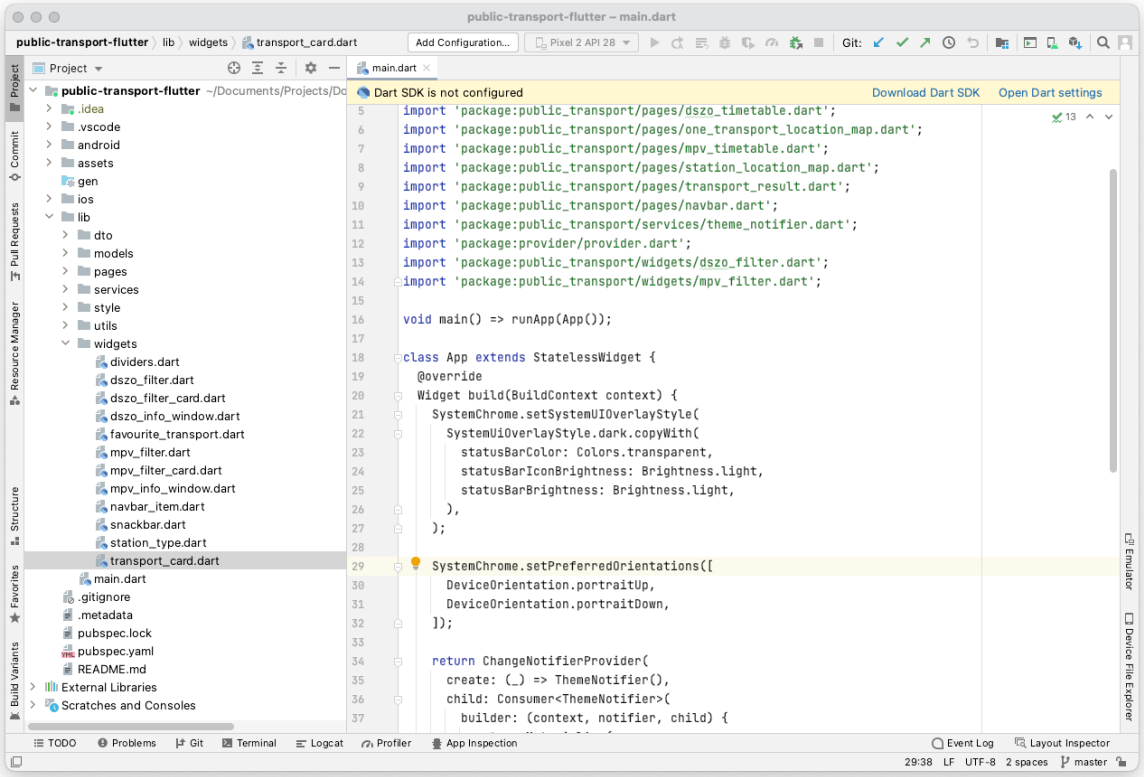



Obrázek 4 - Visual Studio Code

#### *1.1.7.2 Android Studio*

Android Studio je založeno na prostředí IntelliJ IDEA, což je IDE určené převážně pro vývoj aplikací v Javě, Kotlinu, Groovy či v dalších jazycích, které využívají Java Virtual Machine (JVM). Pokud má vývojář zkušenosti s již zmíněnou IntelliJ IDEA, jistě uvítá povědomé rozmístění všech prvků, známé klávesové zkratky a další funkce, které mu usnadní vývoj.

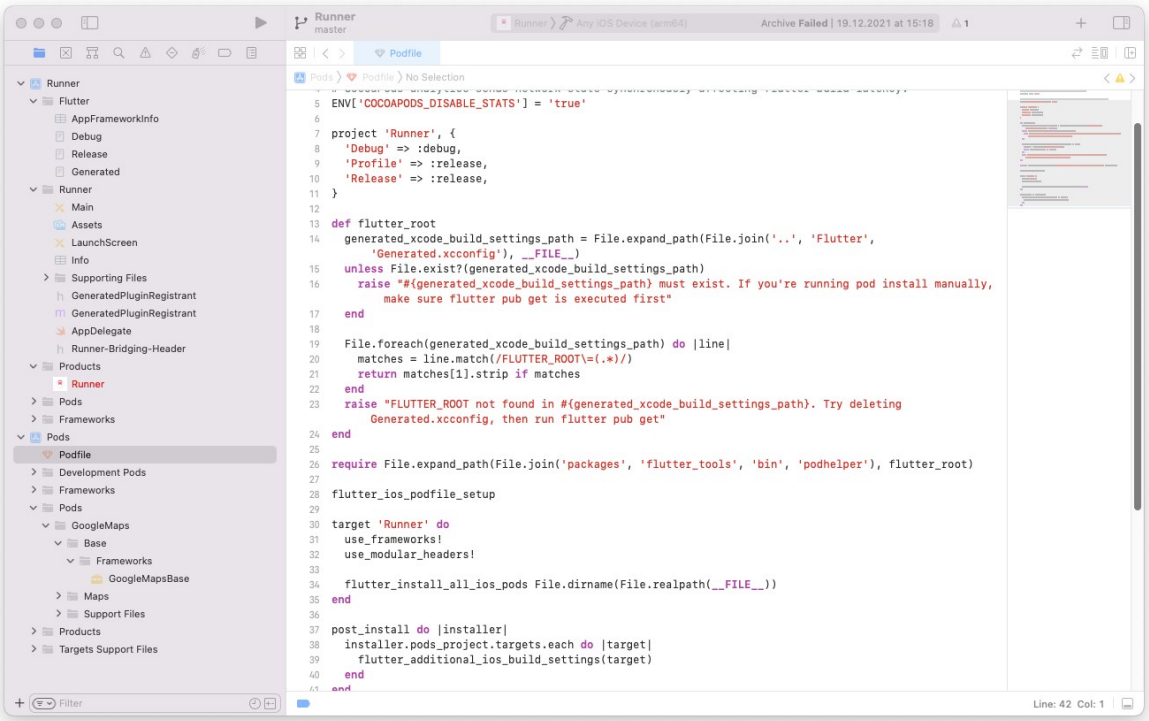
Je zde také možnost instalace různých doplňků z online market place, které ještě více zefektivní vývoj (například pro generování kousků kódu, zvýraznění syntaxe apod.). Oproti VS Code je toto IDE poměrně náročné na výkon počítače, což může být občas problém, zvláště tehdy, pokud vývojář nemá dostatečně výkonný počítač pro běh simulátoru, náročnějšího IDE a sestavování kódu zároveň. Nutno ještě podotknout, že spolu s Android Studiem přijdou v instalaci také veškeré simulátory zařízení s Androidem, nicméně není nutné samotné IDE používat i na vývoj, ale pouze na start a případně vytvoření nových profilů pro simulátory (Obrázek 5 - Android Studio) [10].




Obrázek 5 - Android Studio

#### *1.1.7.3 XCode*

Vývojové prostředí XCode je dostupné pouze na zařízeních s operačním systémem macOS v rámci App Store. Pokud je však vyvíjena aplikace, která bude dostupná pro platformu iOS, je pohodlnější si toto prostředí nainstalovat, protože poskytuje uživatelské rozhraní pro konfiguraci aplikace před publikováním do App Store. Pro publikaci a konfiguraci je samozřejmě možné využít i jiné nástroje, samotný XCode pouze pomocí uživatelského rozhraní umožňuje upravit konfigurační soubory Flutter aplikace, díky čemuž je konfigurace jednodušší a přehlednější. Samotný vývoj aplikace ve Flutteru pro iOS však nemusí probíhat přímo přes XCode, je totiž převážně určen pro Swift, SwiftUI a Objective-C na vývoj aplikací pro Apple ekosystém [11]. Pokud je ale vývojář na toto prostředí zvyklý a má v něm zkušenosti z předchozího vývoje, může ho samozřejmě použít také (Obrázek 6 - XCode).




Obrázek 6 - XCode

### 1.1.8 Knihovny, balíčky a konfigurace

Ve Flutteru lze používat knihovny a balíčky třetích stran, které jsou dostupné v online repozitáři. Je možné si je procházet a vyhledávat na portále *https://pub.dev*, odkud si je lze jednoduše do aplikace přidat [12]. Stačí pouze zkopírovat název balíčku a jeho verzi do souboru s metadaty `pubspec.yaml`, do sekce `dependencies`, spustit příkaz `flutter pub get` (při používání VS Code s doplňkem pro Flutter stačí soubor jen uložit) a balíčky se samy

z repozitáře stáhnou a přidají do závislostí aplikace. Soubor `pubspec.yaml` slouží zároveň jako konfigurace pro celý projekt a nastavují se v něm například verze aplikace, dodatečná konfigurace doplňků, závislosti pro vývoj, cesty k ikonám a další (Obrázek 7 - Soubor pro konfiguraci pubspec.yaml).

```yaml
environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.2
  google_maps_flutter: ^2.0.6
  provider: ^4.3.1
  ...

flutter_icons:
  ios: true
  android: true
  ...
```

Obrázek 7 - Soubor pro konfiguraci pubspec.yaml

# 2 BACKEND – POUŽITÉ TECHNOLOGIE

## 2.1 Java

Java je objektově orientovaný, staticky typovaný programovací jazyk a také výpočetní platforma, která byla prvně vyvinuta firmou Sun Microsystems v roce 1995 (firma byla koupena společností Oracle v roce 2010). V roce 2007 uvolnila firma zdrojové kódy Javy a tím se stala open source s možností kontribuce vývojářů a dalších firem do projektu. Mimo jiné se nově začal používat název OpenJDK (od verze Javy 7). V dnešní době se jedná o jeden z nejpopulárnějších a nejpoužívanějších jazyků, potažmo platform, mezi vývojáři vůbec. Díky své snadné přenositelnosti, jednoduchosti a rozšířenosti se dá Java použít pro téměř jakoukoliv úlohu [13].

Samotný zdrojový kód napsaný v Javě se při kompilaci převádí do tzv. Java bytecode (z `.java` vznikají při kompilaci `.class` soubory), který není závislý na žádné specifické platformě. Při spuštění aplikace je nutné mít na cílovém zařízení nainstalovanou JRE (Java Runtime Environment), která obsahuje mimo jiné knihovny a závislosti i JVM (Java Virtual Machine), která překládá Java bytecode do nativního kódu pro danou platformu. Pro vývoj aplikací v Javě si vývojáři musí stáhnout JDK (Java Development Kit), které obsahuje vše nutné k vývoji a spuštění aplikace.

Java má za sebou již několik let vývoje a verzí – na podzim roku 2021 byla vydána nejnovější verze 17, což je tzv. LTS (Long Term Support) verze. To znamená, že bude podporována delší dobu než standardní verze Javy, v tomto případě až do roku 2029. Další verze Javy jsou již naplánovány na následující roky. Nutno ještě podotknout, že nové verze Javy jsou zpětně kompatibilní s předchozími verzemi, takže například zkompilovaný kód na verzi 17 bude fungovat na prostředí, kde je nainstalované JRE verze 11. Java obsahuje také ještě několik edicí, z čehož je každá určena pro jiné prostředí. Nejrozšířenější edice je Java SE (Standard Edition), která je určena pro pracovní stanice a servery. Tato edice obsahuje veškeré základní knihovny a nástroje pro vývoj klasických aplikací. Další populární edicí je Java EE (Enterprise Edition) často také označována jako Jakarta EE nebo J2EE. Rozšiřuje Standard Edition o knihovny pro práci s webovými službami a vývoj webových aplikací, servletů, knihovny pro práci s databází, dependency injection anebo také podporu Enterprise Java Beans [14]. Tato edice Javy se stala základem pro následující frameworky jako jsou například Spring nebo potom Spring Boot, které Javu EE postupně nahrazují. Mezi další

edice patří také třeba JavaCard (používána například na kreditních kartách) nebo Java ME (Micro Edition), která je určena pro aplikace na mobilních zařízeních.

## 2.2 Kotlin

Stejně jako každá technologie, i Java má své problémy a omezení, proto v roce 2011 vznikl programovací jazyk Kotlin, který spoustu těchto problémů Javy řeší. Kotlin je často popisován jako evoluce Javy. Nyní je primárně vyvíjen firmou JetBrains, která stojí za projekty jako IntelliJ IDEA, PyCharm apod., nicméně zdrojové kódy Kotlinu jsou od roku 2012 open source, takže kdokoliv z komunity se může podílet na vývoji.

Kotlin je kompatibilní s Javou, tzn. je možné používat knihovny napsané v Javě i v projektech, které jsou založeny na Kotlinu a naopak. Toto fungování je možné díky tomu, že Kotlin mimo jiné prostředí běží také na JVM (Java Virtual Machine) a kompiluje se do Kotlin bytecode, který je prakticky stejný jako Java bytecode (často referováno jako jazyk JVM). Hlavní myšlenka Kotlinu je ta, že by vývojáři měli postupně přidávat a přepisovat stávající Java kód na Kotlin a nakonec Javu v projektu nahradit Kotlinem úplně [15].

Jedny z jeho největších výhod oproti Javě jsou tzv. null safety, nullable typy a safe calls, což znamená to, že potencionální NPE (`NullPointerException`) jsou kontrolovány v kompilačním čase. Ve výchozím stavu nelze do proměnné přiřadit `null`, pokud to tak sami neurčíme pomocí `?` u datového typu – `var num: Int? = null`. Pokud je proměnná označená jako nullable, před použitím této proměnné kompilátor vývojáře vyzve ke kontrole, zdali se proměnná nerovná `null` a tím pádem se vývojář vyhne potencionální NPE. Používání nullable proměnné v kódu lze provést mimo jiné způsoby také pomocí tzv. safe calls, které ve výsledku stejně provádí kontrolu, zda daná proměnná není `null`, nicméně lze k tomu použít funkce a konstrukty, které Kotlin implementuje – `num?.let { it.toString() }`, `num?.toString()`. Pokud bychom však NPE v programu chtěli z nějakého důvodu vyhodit a odchytit ji na nějakém jiném místě, je možné u proměnné použít tzv. not-null assertion operator, který, pokud je proměnná `null`, NPE vyhodí – `num!!.toString()` [16].

Další velkou výhodou jsou tzv. extension functions, které umožní vývojářům rozšířit již existující knihovny a třídy o další funkcionalitu [17]. Například k třídě `String` je možné přidat funkci, která bude vracet první písmeno z textu a tato funkce bude potom následně dostupná u všech instancí této třídy, takže není nutné psát třídu se statickými metodami či podobné řešení, které by tuto funkcionalitu implementovalo.

V neposlední řadě Kotlin nabízí také mnohem stručnější, přehlednější a čitelnější syntax, kde spoustu opakujícího se kódu za nás generuje sám Kotlin kompilátor [15] (např. `getters` a `setters` u tříd anebo `equals()`, `copy()`, `hashCode()` metody, které jsou dostupné u datových tříd) [18].

## 2.3 Spring Boot Framework

Jedná se o velmi populární a v dnešní době hojně používaný open source framework, který obsahuje nástroje pro tvorbu produkčních, vysoce škálovatelných aplikací, které běží na Java Virtual Machine (JVM), což umožní vývojářům psát v jazyce jako je například Java, Kotlin nebo Groovy. Spring Boot Framework je založený na Spring Frameworku, který ho rozšiřuje a zjednodušuje jeho konfiguraci, která byla u Spring aplikací nutná dělat manuálně, tím pádem se eliminuje velké množství tzv. boilerplate kódu (repetitivní kód) [19].

Spring, resp. Spring Boot obsahuje knihovny a sadu frameworků pro ORM (Object-Relational Mapping – mapování z objektu na databázové tabulky a naopak), MVC (Model, View, Controller), Security, JDBC (Java Database Connectivity) a mnoho dalšího. Do projektu lze přidat například i knihovnu, která podporuje práci s webovými REST servisy. Tato knihovna tedy obsahuje také vestavěný Tomcat server, který je už v základu nakonfigurovaný tak, aby stačilo aplikaci jen sestavit a spustit bez dalšího nastavování. Samotný framework lze také rozšířit o další závislosti a knihovny, které poskytují vývojářům novou funkcionalitu. Velké množství knihoven je vyvíjeno přímo vývojáři frameworku, tudíž jsou dobře podporované a testované. V závislosti na zvoleném nástroji pro automatizované sestavování (Gradle nebo Maven) lze přidávat do aplikace knihovny do konfiguračního souboru k tomu určenému, odkud se kódy knihoven stáhnou z online repozitáře [20].

Při vytváření nové aplikace ve Spring Bootu lze použít online nástroj Spring Initializer - *https://start.spring.io*, ve kterém je možné si zvolit knihovny, které budou v aplikaci zahrnuty už od samotného začátku vývoje [21]. Po nastavení verze frameworku, jazyka, jména a dalších detailů lze stáhnout šablonu, která je připravena na další vývoj a implementaci business logiky a není nutné ji dále konfigurovat a nastavovat, což je velká výhoda celého Spring Boot ekosystému.

## 2.4 Cloud

V dnešní době se stává čím dál více populární Cloud Computing, který poskytuje vývojářům a společnostem mnoho výhod. Jednou z nich je například to, že platíte pouze za to, co je opravdu využito – pokud má aplikace v určitou denní dobu zvýšený provoz, dá se jednoduše vyškálovat tak, aby tento provoz zvládla zpracovat. Naopak na druhou stranu, pokud např. v noci aplikace není tolik využívaná, je možné ji škálovat dolů, aby zbytečně nespotřebovává zdroje, čímž vývojářům a společnostem ušetří peníze za provoz. (Pokud aplikace běží jako kontejner a je použit orchestrační nástroj jako Kubernetes, myslí se škálováním snížení nebo zvýšení počtu instancí podů/kontejnerů). Další velkou výhodou je samozřejmě to, že není potřeba se starat o datacentra, samotný hardware a tím také odpadá nutnost platit lidi, kteří by tyto činnosti vykonávali. Samotný cloud se dělí na několik typů.

- **Privátní** – Jedná se o tzv. on premise řešení, což znamená to, že je datové centrum plně v režii společnosti, která se stará jak o hardware, bezpečnost i software.
- **Veřejný** – Cloud, který je dostupný pro všechny veřejně a kdokoliv si může pronajmout služby, které poskytuje (databáze, hardware, Kubernetes servis atd.).
- **Hybridní** – Kombinace dvou výše zmíněných typů – pokud firma např. uchovává citlivé údaje nebo z důvodu legislativního nařízení nemá oprávnění ukládat data na infrastrukturu poskytovatele třetí strany, ale zároveň chce využívat výhod veřejného cloudu, je toto řešení ideální.

Mezi největší poskytovatele veřejných cloudů patří Amazon s jeho Amazon Web Services (AWS), Microsoft s Azure a Google s Google Cloud Platform (GCP) [22]. Spousta dalších společností staví na základech těchto cloudů a dělá nad nimi svoje “nadstavby”, jako například Heroku.

### 2.4.1 Heroku

Heroku je cloudová platforma, která je postavená nad Amazon Web Services a poskytuje mnoho služeb a nástrojů pro sestavení a nasazení aplikace, monitorování, analýzu, cachování atd. V porovnání s jiným cloudem je Heroku o poznání jednodušší, a tudíž i mnohem vhodnější pro malé projekty [23]. Heroku nabízí uživatelům poměrně štědrý bezplatný tarif, který je dostačující pro prototypování a testování. Jeho nevýhodou je však to, že po půl hodině neaktivity škáluje počet instancí na 0, což způsobí to, že dotaz, který přijde na “spící” aplikaci, trvá delší dobu, protože se musí aplikace znovu spustit. Celkový čas, kdy může aplikace běžet, je 550 hodin za měsíc pro neověřené účty. Pokud si vývojář účet ověří

přidáním kreditní karty, dostane dalších 450 hodin. Tento účet zdarma dostane runner, který má však pouze 512 MB RAM paměti a nemá přístup k detailním statistikám [24]. Jeden z nejlevnější tarifů, který Heroku nabízí stojí 7$ za měsíc [25], poskytuje časově neomezený běh aplikace, detailní statistiky z používání, avšak stále pouze jen 512 MB RAM, což by pro hobby projekt mělo stačit a samozřejmě záleží, v jaké platformě je projekt vytvořen a kolik samotná aplikace spotřebuje.

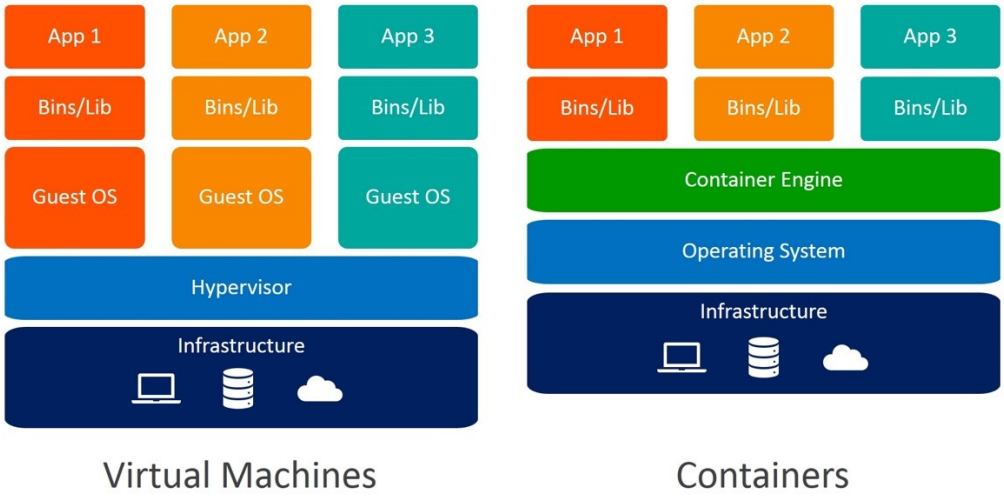
Samotné nasazení aplikace je velmi jednoduché – Heroku nabízí svůj nástroj pro příkazový řádek, který si lze nainstalovat, poté je nutné se přihlásit a přidat do projektu adresu na nový `git remote` s adresou na Heroku projekt. Při nahrávání změn a nové verze na server se automaticky spustí sestavení aplikace díky nástroji Cloud Native Buildpacks [26], který automaticky rozpozná, o jaký projekt se jedná a na základě toho vybere vhodný nástroj pro sestavení (Gradle, Maven, Node…) a vytvoří Docker image, který je poté spuštěn a zpřístupněn na výchozí adrese projektu (je zde také možnost přesměrovat záznam na vlastní adresu).

### 2.4.2 Kontejnerizace

Velmi populární varianta je spouštět aplikace zabalené do kontejnerů – tento přístup umožní vývojářům rychlejší a spolehlivější nasazení, sestavení a správu aplikací. Používání kontejnerizace je také mnohem úspornější z hlediska zdrojů a mimo jiné nabízí mnohem lepší přenositelnost v případě migrace nebo nasazení na více zdrojů. S tradiční metodou (využití virtuálních strojů – Virtual Machines – VMs) je kód vyvíjen zpravidla pro jednu specifickou platformu, což při přesunu na jinou platformu může způsobit mnoho chyb – například přesun z operačního systému Linux na Windows či z lokálního počítače do virtuálního stroje.

Při kontejnerizaci aplikace se myslí to, že se samotná aplikace zabalí spolu s nezbytnými procesy operačního systému a nutnými závislostmi pro běh do velmi jednoduchého souboru zvaného kontejner, který lze poté spustit kdekoliv, kde je nainstalovaný software, který umožní práci s kontejnery, takže není nutné mít na cílové platformě (host platform) nainstalovanou například JRE (Java Runtime Environment) pro běh aplikace napsané v Javě, protože je JRE již obsažená v kontejneru s aplikací. Ve srovnání s virtuálními stroji nemají v sobě kontejnery celý operační systém, což se projeví na jeho výsledné velikosti. Kontejnerizace poskytuje také izolovanost a oddělenost aplikací, které by například na virtuálním stroji běžely na jednom hostitelském operačním systému a mohly by se navzájem ovlivňovat.

Díky kontejnerům mohou aplikace sdílet stejné zdroje (CPU, RAM – běžet na stejném serveru), avšak navzájem jsou od sebe izolovány a nemohou se ovlivnit [27] (Obrázek 8 - Porovnání aplikace na virtuálním stroji a v kontejneru [28]). Tento přístup je velmi populární hlavně v oblasti cloudu, kdy mohou vývojáři pracovat s nástroji jako Docker, Kubernetes, OpenShift a další, které jsou na kontejnerizaci postaveny. Díky těmto nástrojům mohou jednoduše aplikace škálovat a spravovat dle požadavků nebo aktuální zátěže. Pro poskytovatele cloudu je tento přístup také velmi výhodný v tom, že nemusí pro každou aplikaci, případně zákazníka, alokovat celý server zvlášť, aby byla aplikace oddělená od ostatních zákaznických produktů (pokud si sám zákazník nevybere v cloudu službu, která mu nabízí čistý virtuální server). Dnes už každý poskytovatel cloudu nabízí plně spravované služby (zákazník se nestará o infrastrukturu, která je pod službou, nestará se tedy o aktualizace či bezpečnostní záplaty), které nabízí běh aplikací zabalené jako kontejnery.




Obrázek 8 - Porovnání aplikace na virtuálním stroji a v kontejneru [28]

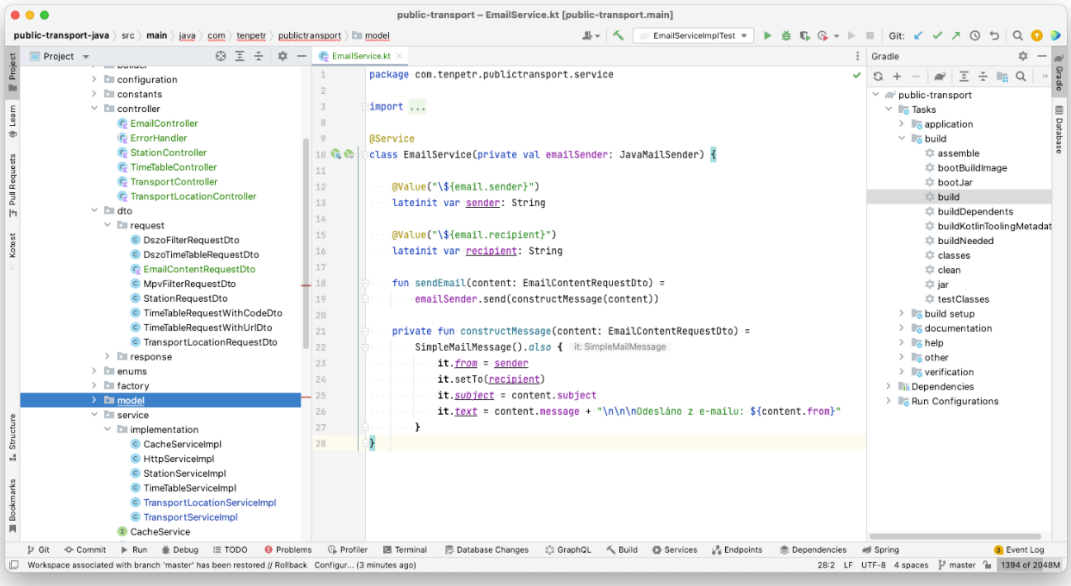
Mezi nejpopulárnější software pro práci s kontejnerizovanými aplikacemi je Docker, který nabízí sadu nástrojů pro správu, vytváření a běh kontejnerizovaných aplikací. Díky Dockeru lze tedy aplikaci zabalit do kontejneru (vytvořit z aplikace tzv. Docker Image), stačí pouze nadefinovat ve zdrojovém kódu jednotlivé kroky v `Dockerfile` a spustit řadu příkazů, které jsou dostupné v dokumentaci [30]. Pokud má aplikace kontejnerů více, je možné využít nástroj pro správu a orchestraci kontejnerů jako je například Kubernetes, díky němuž lze jednoduše škálovat a spravovat veškeré části aplikace (Pods – nejmenší jednotka je Pod, který obsahuje jeden a více kontejnerů), které v rámci tohoto nástroje běží. Definice těchto Podů a kontejnerů spolu s další konfigurací se definují do `.yaml` souborů a lze také využít nástroj pro příkazovou řádku, díky které se Kubernetes ovládají [31].

# 3 POUŽITÉ NÁSTROJE A METODIKY PŘI VÝVOJI

## 3.1 Vývojové prostředí a textový editor

Pro vývoj backend aplikace bylo použito vývojové prostředí od firmy JetBrains, tedy IntelliJ IDEA. Jedná se o vývojové prostředí, které nabízí velké množství užitečných funkcí pro zrychlení vývoje, refactoring, generování kódu apod. Toto vývojové prostředí (IDE) je dostupné v komunitní i placené verzi, která nabízí o něco více funkcí, než komunitní verze zdarma (širší podpora Spring Boot frameworku, nástroje pro práci s databází, Kubernetes, a další). Samotná IDEA podporuje spoustu jazyků, primárně však ty pro JVM (Java Virtual Machine) jako jsou Java, Kotlin Scala a Groovy. Mimo zmíněné jazyky má podporu také pro HTML, CSS, JavaScript. Velká výhoda jsou doplňky, které si lze do tohoto prostředí instalovat z marketplace a díky nim získat podporu pro další různé nástroje nebo prostředí nějakým způsobem vylepšit a personalizovat. IDEA má v sobě také vestavěný nástroj pro verzovací systém Git, tím pádem operace spojené s verzováním lze provádět přímo v IDEI (Obrázek 9 - IntelliJ IDEA).

Pro vývoj frontend části, tedy mobilní aplikace, byl zvolen textový editor od firmy Microsoft – Visual Studio Code (VS Code). Tento textový editor nabízí taktéž možnost instalace velkého množství doplňků, které pomáhají při vývoji a ulehčují práci. VS Code má mimo jiné samozřejmě i plugin pro Fluter, díky kterému lze například používat funkce jako Hot Reload přímo v editoru bez nutnosti použití příkazového řádku.




Obrázek 9 - IntelliJ IDEA

## 3.2 Verzování

### 3.2.1 Nástroj Git

Jako verzovací systém byl v projektu použit Git, což je velmi pokročilý, distribuovaný open source nástroj, který v dnešní době používá spousta firem a vývojářů. Původní projekt založil tvůrce Linuxu Linus Torvalds v roce 2005 [32].

V projektu je použit systém pull requestů, feature branch, tagů a issues, které jsou dostupné na platformě GitHub. Každá nová funkce na základě issue (issue popisuje novou funkci, která má aplikace přinést v další verzi, jedná se tedy o jakýsi ticket, který popisuje business logiku nové funkce) je vyvíjena na své tzv. "feature branch" označená podle čísla a názvu issue/funkce – například `feature/54-station-location-implementation`. Po následné implementaci je otevřen pull request, kde se kód zkontroluje a prochází, jestli odpovídá požadavkům, neobsahuje chyby a splňuje všechny body zadání. Tento proces je samozřejmě více užitečný, pokud na aplikaci pracuje více než jeden vývojář. Po schválení je větev integrována do hlavní větve `master`, která se poté nasazuje na cloud Heroku a označí se tagem s aktuální verzí. Pokud se v aplikaci objeví chyba, samotná oprava je implementována ve vlastní tzv. "fix branch" (`fix/wrong-bus-number-prefix`) a poté je integrována přímo do hlavní větve `master`.

### 3.2.2 Sémantické verzování

Pro verzování aplikace (jak pro mobilního klienta, tak pro backend) byl zvolen systém sémantického verzování. Tento systém verzování se hojně používá pro open source knihovny, interní nástroje, mobilní aplikace a mnoho dalšího. Hlavní myšlenkou je rozdělení na tři části, které se s postupnými aktualizacemi a změnami inkrementují. Sémantická verze může vypadat následovně – `2.12.5-alpha`. Veškeré specifikum a pravidla k sémantickému verzování je sepsáno v oficiální dokumentaci na webových stránkách – *https://semver.org/lang/cs*.

Samotná verze aplikace se konfiguruje v souboru `pubspec.yaml`, který je obsažen v kořenovém adresáři každého Flutter projektu. Tento soubor slouží rovněž také pro přidávání závislostí, konfigurace fontů apod., jak již bylo zmíněno v kapitolách dříve (Obrázek 10 - Soubor pubspec.yaml s názvem a verzí aplikace).

Samotná sémantická verze je rozdělena na následující části [33].

- **Major** – Inkrementace nastává tehdy, pokud v softwaru nastala změna, která není zpětně kompatibilní s ostatními funkcemi, jinými slovy, pokud aplikace prošla větší změnou a byla přidána zásadní funkce, která má vliv na zpětnou kompatibilitu.
- **Minor –** Inkrementováno tehdy, pokud se do softwaru přidá funkcionalita nebo větší oprava chyb, která však neporuší zpětnou kompatibilitu.
- **Patch a Build** – Patch verze software se inkrementuje tehdy, pokud nová verze obsahuje opravy chyb z předchozích verzí a zachovala se zpětná kompatibilita. Build, případně upřesnění verze software je volitelný a není povinný – přidává se na konec čísla verze.


Obrázek 10 - Soubor pubspec.yaml s názvem a verzí aplikace

## 3.3 Čistý kód a refaktorování

V rámci vývoje byl kód aplikace, jak na straně serveru, tak na straně mobilního klienta, několikrát přepsán a refaktorován tak, aby byl, pokud možno co nejčistší, nejčitelnější a odpovídal co nejvíce pravidlům, které se mají při psaní kódu dodržovat. Tato pravidla jsou popsána v několika knihách od autorů jako Robert Cecil Martin [34][35] nebo Martin Fowler [36].

První princip čistého kódu, který byl v rámci psaní kódu v aplikaci dodržován, je to, že by metoda měla dělat pouze jednu činnost, bez jakýchkoliv postranních efektů, což znamená to, že když metoda například formátuje textový řetězec se zpožděním vozu, neměla by dělat nic jiného, jako například ukládat data do cache paměti a podobně. V rámci vývoje první verze některé metody toto pravidlo porušovaly, avšak v dalších verzích během vývoje se díky refaktorování metody rozdělily tak, aby toto pravidlo splnily. Co se týče metod a refaktorování, při přepisování byly také upraveny vstupní argumenty, které metoda

přijímala. Podle pravidel čistého kódu by metoda neměla přijímat více než dva až tři argumenty, proto, pokud je jich v metodě potřeba více, bylo nutné vytvořit datovou třídu, která tyto argumenty zaobalovala do sebe a v metodě se následně používala instance dané datové třídy.

Pokud však metoda přijímá více než 3 argumenty, může to znamenat, že metoda nedělá pouze jednu činnost, jak by měla. Při refaktorování metod a jejich zmenšování je dobré rozdělit činnost metody na více drobných privátních metod, které jsou potom ve veřejné metodě volány. Tímto způsobem, pokud přijde ke kódu někdo nový, jasně pochopí, o co se jedná a co daná metoda dělá. V ideálním případě by měla být metoda dlouhá v rozmezí mezi 5 a 10 řádky, nicméně hodně také záleží na tom, v jakém jazyce a jaké konvence pro programování jsou použity, u některých jazyků se například píší otevírací složené závorky na nový řádek, což ve výsledku metodě na délce přidá, nicméně může to však znamenat větší čitelnost a přehlednost metody. Stejně jako počet řádků, samotný řádek by se měl držet, pokud možno co nejkratší [34][36].

Dalším velkým faktorem čistého kódu je pojmenování proměnných, metod, tříd a všech komponent, které se v kódu nachází. Správné a výstižné pojmenování je jedna z nejtěžších věcí při vývoji, protože se ji vývojář učí po celou dobu své kariéry. Existuje však několik pravidel a konvencí, které se při vývoji používají jako například to, že metoda by měla být pojmenována jako sloveso – `printLocation()`, proměnná typu `boolean` by měla začínat `is`, případně `are` – `isSelected`, třídy by měly být pojmenovány s velkým písmenem na začátku – `LocationService` (Pascal Case), neměly by se používat názvy kolekcí v rámci proměnných, které je využívají – `locationsList`, mělo by se vyhnout používání slov jako "Manager, Processor, Data nebo Info" a mnoho dalšího [34][36].

Stejné pravidla platí také pro třídy, to znamená, že i třídy by se měly starat pouze o jednu doménovou věc. Například servisní třída pro polohu vozů by v sobě neměla obsahovat metody, které budou nějakým způsobem pracovat s odjezdovými tabulemi apod.

V rámci celého kódu je také mnoho metod, a dokonce i tříd znovu použito tak, aby se vyhnulo repetetivnímu kódu, takže například pro parsování textového řetězce, které je potřeba volat z mnoha míst a mnoha servisů, je vytvořena pouze jedna metoda, která se o toto parsování stará a je znovu použita na více místech [34].

Důležitým principem je oddělování prezentační a business logiky celé aplikace. Pokud má aplikace například databázi a entity, které se na databázové tabulky mapují (v tomto případě

k tomuto účelu slouží cache anebo servery poskytovatelů dat), je ideální, aby tyto entitní třídy zůstaly izolované v rámci servisy a nebyly používány v kontrolerech či jiných třídách, určené pro prezentaci dat uživateli. Proto jsou v aplikaci vytvořeny speciální datové třídy, označované jako DTO (Data Transfer Object), které slouží k přenášení dat mezi uživatelem (controller) a samotnou business logikou (servisní třídou). K mapování entitních tříd na DTO a naopak slouží mappery (existují i frameworky, které tyto mappery generují na základě specifikované method signature v rozhraní). Velká výhoda tohoto přístupu nastává tehdy, když se změní nějaká entita anebo uživatel požaduje jiný formát objektu, tímto způsobem se tedy danou změnou nerozbije celá aplikace [35].

V kódu také nejsou komentáře, respektive se zde objevují ve velmi malé míře. Samotné komentování kódu není nutné, pokud je metoda či proměnná samo-vysvětlující. Problém s komentáři je ten, že pokud se daná implementace změní, často se zapomíná také na změnu a upravení komentáře, který danou funkcionalitu popisoval, po změnění implementace tedy tento komentář není relevantní a spíše mate, takže pokud se po delší době vývojář ke kódu vrátí, nic z toho nezíská [36].

V neposlední řadě při vývoji platilo pravidlo “Keep It Simple, Stupid!” (KISS), což znamená to, že je lepší, pokud je daná věc implementována spíše jednodušeji než komplexněji. Komplexnější způsob implementace nemusí být všemi v týmu pochopen a po nějaké době, když se ke kódu sám vývojář, který funkci implementoval, vrátí, nebude sám vědět, co daná metoda nebo implementace dělá. Proto bylo v rámci vývoje toto pravidlo bráno na zřetel a pokud možno co nejvíce využíváno [34].

# 4 SESTAVENÍ APLIKACE A PŘÍPRAVA NA DISTRIBUCI

Flutter framework umožňuje kompilovat zdrojový kód napsaný v jazyce Dart hned na několik platforem – Android, iOS a web. Samotný proces sestavování a publikace aplikace se na každé platformě mírně liší. Google však na stránkách oficiální dokumentace Flutteru jednotlivé kroky detailně popisuje a je tedy třeba tyto kroky dodržovat, aby se celý proces úspěšně dokončil [37][38]. Výsledná sestavená aplikace může být použita pro testování na reálných zařízeních, anebo rovnou pro distribuci mezi veřejnost, případně mezi testery aplikace.

## 4.1 Ikony

Základem každé aplikace je její ikona, proto je nutné v prvním kroku tuto ikonu vygenerovat. U Androidu se generují dílčí velikosti pro jednotlivé verze systému a velikosti obrazovky zařízení. Od Androidu verze 8.0 je možné používat tzv. adaptive icons, které se samy přizpůsobí tvaru určeného systémovou nástavbou, případně verzí Androidu a velikostí displeje. Tyto ikony mají také při jejich manipulaci a přesouvání na ploše jednoduché vizuální efekty [39]. Flutter adaptivní ikony taktéž podporuje, je pouze potřeba vytvořit pozadí a samotnou ikonu jako dva oddělené obrázky a následně pomocí pluginu `flutter_launcher_icons` jednotlivé ikony a konfiguraci nechat vygenerovat. Plugin vyžaduje přidání cest k obrázkům a další konfiguraci do souboru `pubspec.yaml` (Obrázek 11 - Soubor pubspec.yaml a konfigurace ikon). Po spuštění příkazu se vygenerují ikony a upraví se také konfigurace v `./android` složce [37].

Ikony pro iOS a proces generování probíhá stejným způsobem jako na Androidu, je pouze potřeba přidat specifickou konfiguraci v `pubspec.yaml`, protože iOS adaptivní ikony nepodporuje. Po vygenerování ikon různých velikostí se automaticky upraví i konfigurace tak, aby se přepsala výchozí ikona pro iOS aplikaci, takže není nutné nic měnit manuálně [38].

```yaml
flutter_icons:
  ios: true
  android: true
  image_path_ios: "assets/icons/icon.png"
  image_path_android: "assets/icons/icon.png"
  adaptive_icon_background: "assets/icons/background.png"
  adaptive_icon_foreground: "assets/icons/foreground.png"
```

Obrázek 11 - Soubor pubspec.yaml a konfigurace ikon

## 4.2 Kontrola konfigurace

Před samotným sestavením by vývojář měl zkontrolovat a projít důležité konfigurační soubory. V rámci Android aplikace je první z těchto souborů umístěn v adresáři `./android/app/src/main` a jedná se o `AndroidManifest.xml`. V tomto souboru se kontroluje oprávnění, které může aplikace potřebovat ke správnému chodu (přístup k internetu, poloze, fotoaparátu…), název aplikace, který bude zobrazen u ikony, popis aplikace atd.

V dalším konfiguračním souboru `build.gradle`, který je umístěn v `./android/app`, je nutné zkontrolovat unikátní identifikátor aplikace, verzi a jméno aplikace, minimální a cílovou verzi Androidu a podobně [37].

Stejný princip platí také pro iOS projekt, kdy je vhodné si znovu projít a překontrolovat veškeré náležitosti, jako je jméno aplikace, Bundle ID, minimální podporovanou verzi iOS, cílenou verzi iOS atd. Veškerá konfigurace je uložena v `Runner` souboru, který lze najít v `./ios` adresáři. Pro otevření tohoto souboru a nastavení konfigurace je nutné mít nainstalovaný Xcode, kde se vše nastavuje v uživatelském rozhraní [38].

## 4.3 Sestavení aplikace

Google při sestavování Android aplikace umožňuje samotný kód zmenšit a "ořezat" o zbytečné části, aby výsledná sestavená aplikace zabírala, pokud možno, co nejméně místa (nástroj provádí například přejmenování proměnných, odstranění mezer apod.). Tato možnost pomocí nástroje R8 je ve výchozím nastavení zapnuta, pokud by uživatel chtěl zmenšování vypnout, musí při sestavování aplikace předat do příkazu parametr `--no-shrink` [37].

Při sestavování Android aplikace je možné si vybrat ze dvou výsledných formátů, do kterých lze aplikaci sestavit – `App Bundle` nebo `APK` (Android Application Package). Pokud se vývojář chystá aplikaci rovnou publikovat na Google Play, je lepší vybrat preferovaný formát `App Bundle`, který obsahuje veškerý kompilovaný kód. Google Play používá tento balíček pro generování a poskytování optimalizovaných `APK` souborů pro jednotlivá zařízení, která se mohou lišit například v architektuře procesoru. Oproti tomu `APK` formát může být použit na testování produkčně sestavené aplikace na reálných zařízeních. Publikace aplikace v tomto formátu není doporučená, hlavně kvůli optimalizaci pro jednotlivá zařízení architektury procesorů [40].

Sestavení aplikace pro iOS probíhá velmi podobným způsobem jako u Android projektu. Flutter framework nabízí řadu příkazů, díky kterým lze sestavit aplikaci a následně

z ní vytvořit archiv, který obsahuje sestavenou aplikaci pro vybrané architektury. Celý proces sestavování a generování archivu lze provést také v uživatelském rozhraní v XCode, které mimo jiné funkce nabízí přímé napojení do App Store Connect (správa projektů a aplikací, které mají být distribuovány na App Store), pokud by vývojář chtěl aplikaci a výsledný archiv publikovat pro veřejnost nebo do programu TestFlight pro testery [38].

# II. PRAKTICKÁ ČÁST

# 5 PŘÍPADY UŽITÍ A POŽADAVKY NA APLIKACI

## 5.1 Požadavky na aplikaci

Hlavním cílem při tvorbě této aplikace bylo to, aby uživatelům poskytla co nejvíce užitečných a použitelných funkcí, které jim usnadní cestování veřejnou dopravou ve Zlínském kraji a dá jim podrobnější informace o jejich spoji, na který čekají. Všechny tyto funkce byly navrženy tak, aby se uživatelům používaly co nejlépe, aby měli z aplikace dobrý pocit a byla jednoduchá pro používání i pro starší uživatele. Tyto vlastnosti se tedy řadí mezi nefunkční požadavky na aplikaci a heslovitě mluvíme o jednoduchosti, minimalistickém designu, intuitivnosti, přehlednosti prezentovaných informací, plynulém chodu celé aplikace, líbivém a příjemném vzhledu, potažmo barvách a podporu pro co nejvíce mobilních zařízení.

### 5.1.1 Funkční požadavky na aplikaci

- Uživatel může vyhledat zastávku a zobrazit si příjezdovou nebo odjezdovou tabuli pro vybranou zastávku
- Uživateli jsou našeptávány dostupné zastávky na základě zadaného textu ve vyhledávacím poli
- Uživatel si může uložit zastávku do oblíbených výsledků vyhledávání, kde si může tyto zastávky dále spravovat (např. mazání)
- Uživatel může filtrovat výsledky vyhledávání na základě jména stanice
- Uživatel se může podívat na jízdní řád vybraného spoje/linky, který vizualizuje aktuální polohu spoje
- Jízdní řád zobrazuje časy odjezdů ze zastávek
- Uživatel se může podívat na informace týkající se spojů jako je zpoždění, cílová nebo původní stanice, čas příjezdu nebo odjezdu, číslo linky atd.
- Uživatel se může podívat na aktuální polohu vozidla na mapě přímo z výsledků vyhledávání u odjezdových/příjezdových tabulí
- Uživatel si může zvolit zdroj dat – regionální linky, MHD Zlín a také vše kombinovat
- Uživatel si může vybrat mapový podklad – letecký anebo výchozí
- Uživatel může automaticky zaměřit svoji polohu na mapě
- Uživatel může sledovat aktivní spoje dostupné na mapě a zobrazit si informace o nich jako jsou například poslední navštívená zastávka, cílová nebo původní

zastávka, přesné zpoždění, číslo linky, možnost zobrazit jízdní řád a u vybraných vozů také dostupné funkce jako například obrazovka, klimatizace atd.
- Uživatel může na mapě vyhledávat spoje na základě čísla linky/spoje, funkce se mírně liší pro vybraný zdroj dat
- Systém v pravidelných intervalech aktualizuje data na mapě
- Uživatel může aplikaci používat v tmavém anebo světlém barevném režimu
- Uživatel si může zobrazit dopravní informace na mapě od Google
- Uživatel může kontaktovat autora aplikace pomocí kontaktního formuláře o chybě nebo připomínce k aplikaci
- Uživatel si může přečíst informace o aplikaci a o zdrojích dat
- Systém ukládá uživatelsky nastavené hodnoty do paměti zařízení

## 5.2 Primární případy užití

Aplikace byla vyvíjena se záměrem pomoci lidem při cestování veřejnou dopravou ve Zlínském kraji, a to hned v několika ohledech popsaných v následujících ukázkových případech (Obrázek 12 - Diagram případů užití).

### 5.2.1 Sledování zpoždění vybraného spoje

Uživatel aplikace se chystá odjet z práce domů za pomocí prostředků veřejné dopravy. Když vychází z kanceláře, uvědomí si, že si na stole zapomněl klíče od bytu. Ve chvíli, kdy se vrací, má jeho vlak plánovaný odjezd za 2 minuty a cesta na zastávku mu trvá zhruba 5 minut, proto se podívá do aplikace, aby zjistil, jestli vlak nemá zpoždění a on tak věděl, jestli má pospíchat, aby spoj stihl nebo má v klidu počkat na další vlakové spojení, případně se podívat na alternativní dopravu domů (kombinace trolejbusu a autobusu a podobně).

### 5.2.2 Sledování polohy vybraného spoje

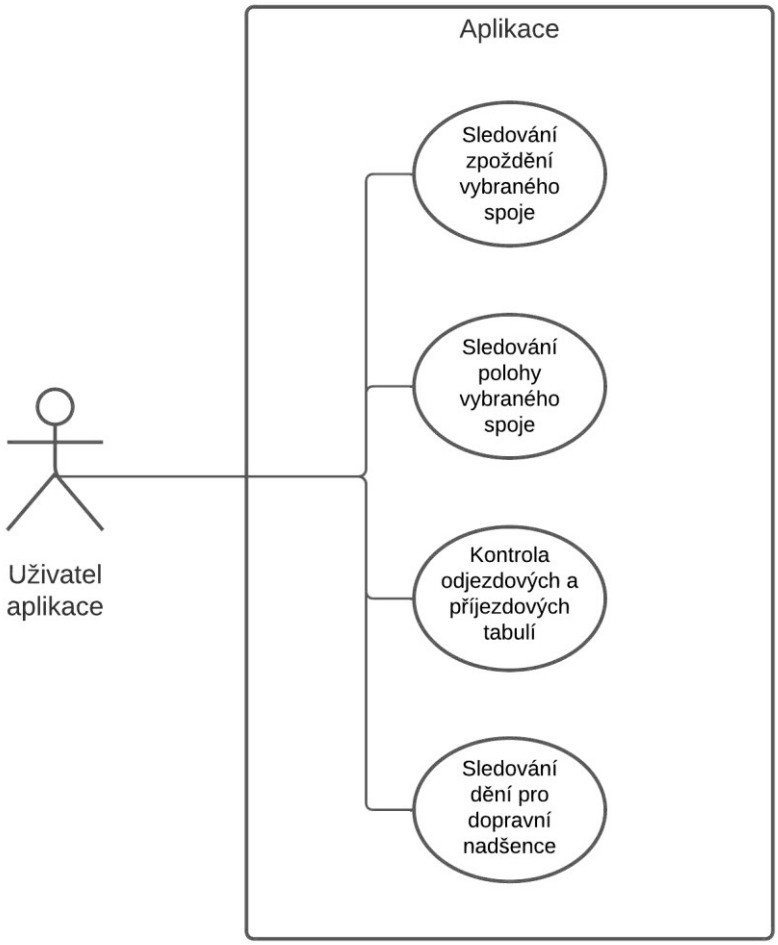
Uživatel aplikace čeká na autobusové zastávce na spoj, kterým jezdí každé ráno pravidelně do práce. Na zastávce však čeká už dlouho a autobus na ni v době plánovaného příjezdu nepřijel. Uživatel se tedy podívá do aplikace, vyhledá spojení podle čísla linky a najde na mapě, kde se spoj nachází. Díky tomu zjistí, že autobus stojí v koloně v křižovatce, kde je díky dopravní špičce velký provoz.

### 5.2.3 Kontrola odjezdových a příjezdových tabulí

Uživateli aplikace ujel pravidelný spoj, kterým se dopravuje z práce domů. Aby nemusel chodit na zastávku a mohl si zajít ještě něco vyřídit do města, podívá se do aplikace na virtuální odjezdové tabule vybrané zastávky, aby zjistil, kdy mu jede další spojení směrem domů a na základě této informace se dále zařídí.

### 5.2.4 Sledování pro dopravní nadšence tzv. šotouše

Uživatel aplikace je velkým nadšencem do veřejné dopravy (šotouš) a rád fotí vlaky, které projíždí místem, které je kompozičně velmi zajímavé. Použije proto aplikaci, aby se podíval, jaký vlak tímto místem pojede a kdy tam bude, takže se stihne připravit na fotografování.




Obrázek 12 - Diagram případů užití

# 6 GOOGLE MAPS A FUNKCE SLEDOVÁNÍ POLOHY VOZIDEL

Aplikace využívá k zobrazení aktuální polohy vozidel (a další funkce, které využívají mapy) jako mapové podklady Google Maps, které byly zvoleny převážně kvůli jejich popularitě mezi uživateli a také díky mnoha dalším funkcím, které následně aplikace využívá a díky nim poskytuje uživatelům relevantnější data a informace.

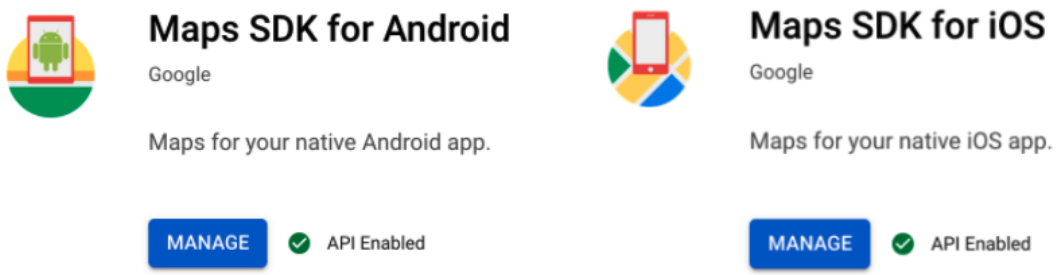
Při výběru byly brány v potaz i jiné mapové podklady, které jsou zdarma, například od OpenStreetMap, nicméně toto řešení nebylo pro aplikaci ideální (tyto mapové podklady jsou využity v mé další aplikaci, která je taktéž vytvořena za pomocí Flutteru). Jeden z důvodů bylo například nízké rozlišení mapových podkladů (map tiles), což by zhoršilo celkový uživatelský dojem z používání aplikace. Dalším důvodem byl také nedostatek motivů (tmavý, světlý), které jsou dostupné zdarma. Google na druhou stranu nabízí velké množství motivů, které si lze přizpůsobit a konfigurovat dle vlastních preferencí, stačí jen upravit hodnoty v konfiguračním souboru nebo si je upravit přímo ve webové aplikaci *https://mapstyle.withgoogle.com* a následně konfigurační soubor stáhnout. Vzhledem k tomu, že aplikace nabízí jak světlý, tak tmavý režim, byla tato možnost přizpůsobení velmi důležitá [41].

## 6.1 Vytvoření API klíče pro Google Maps na Google Cloud Platform

Aby mohla aplikace využívat mapové podklady a další funkce, které Google Maps nabízí, je nutné, aby vývojář měl tzv. API klíč (Application Programming Interface), který musí získat na Google Cloud Platform (GCP) [41].

Po vytvoření účtu na GCP je potřeba vytvořit také projekt (pokud už vytvořený není), u kterého se následně aktivuje Google Maps SDK (Software Development Kit) jak pro Android, tak pro iOS, případně i pro web (Obrázek 13 - Maps SDK pro Android a iOS). Dynamické nativní mapové podklady jsou pro web a mobilní zařízení zdarma, avšak pro aktivaci je nutné mít na svém účtu aktivní a platnou kreditní kartu. Je to z důvodu cenového modelu pay-as-you-go, díky kterému uživatelé platí jen za zdroje, které opravdu využijí. Pokud by tedy uživatel (vývojář) chtěl v aplikaci využívat třeba i Google Street View, musel by už platit podle využití této funkce v jeho aplikaci [42].

Po aktivaci SDK pro vybrané platformy už potom stačí vygenerovat samotný API klíč na stránce *https://console.cloud.google.com/google/maps-apis/credentials*, který je spojený s účtem na GCP. Tento klíč se bude v následujících krocích používat přímo v konfiguračních souborech v aplikaci [43].




Obrázek 13 - Maps SDK pro Android a iOS

## 6.2 Instalace a konfigurace knihovny v aplikaci

Google nabízí pro Flutter oficiální knihovnu, která podporuje práci s Google Maps a nabízí mnoho užitečných funkcí přímo na portále s baličky a knihovnami *https://pub.dev* – `google_maps_flutter`. Při instalaci stačí do souboru `pubspec.yaml` přidat závislost na tuto knihovnu s aktuální verzí a následně knihovnu nechat stáhnout pomocí příkazu `flutter pub get` z repozitáře. Jakmile se knihovna stáhne, je potřeba ji ještě dále nakonfigurovat. Tato konfigurace se mírně liší v závislosti na cílené platformě, pro kterou je aplikace vyvíjena.

### 6.2.1 Konfigurace pro Android

Pro správné fungování této knihovny na zařízeních s operačním systémem Android se musí nejprve nastavit minimální verze Androidu na verzi SDK 20 (Android 4.4W), tzn. že aplikace nebude podporovat nižší verze operačního systému Android, než je verze 4.4W [44]. Toto nastavení se provede v souboru `./android/app/build.gradle`. Po změně bude vypadat část konfigurace následovně (Obrázek 14 - Konfigurace minimální verze Androidu).

```
android {
    defaultConfig {
        minSdkVersion 20
    }
}
```

Obrázek 14 - Konfigurace minimální verze Androidu

Jako další krok je potřeba přidat API klíč, který byl získán v předchozím kroku na Google Cloud Platform. Do aplikace se přidává proto, aby při komunikaci s Google servery bylo možné rozpoznat, zda má aplikace na stahování mapových podkladů právo a vývojář má Google Maps SDK na svém účtu aktivní. API klíč se přidává v konfiguračním souboru `./android/app/src/main/AndroidManifest.xml` [45] (Obrázek 15 - Konfigurace API klíče v Android projektu).

```
<application
        android:name="io.flutter.app.FlutterApplication"
        android:icon="@mipmap/ic_launcher">
        <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="API_KEY"/>
```

Obrázek 15 - Konfigurace API klíče v Android projektu

### 6.2.2 Konfigurace pro iOS

Konfigurace pro iOS probíhá velmi podobně jako u operačního systému Android, mění se však konfigurace v jiných souborech, specifické pro operační systém iOS. Pro správné fungování na zařízeních od Apple je vyžadován iOS verze 9.0 a vyšší, kterou lze nastavit v souboru `./ios/Podfile`. API klíč ke Google Maps, který byl získán v předchozím kroku na Google Cloud Platform, se přidává do souboru `./ios/Runner/AppDelegate.m` nebo `./ios/Runner/AppDelegate.swift` [45] (Obrázek 16 - Konfigurace API klíče v iOS projektu).

```
import UIKit
import Flutter
import GoogleMaps

@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
  ) -> Bool {
    GMSServices.provideAPIKey("API_KEY")
    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

Obrázek 16 - Konfigurace API klíče v iOS projektu

## 6.3 Používání knihovny v aplikaci

Používání samotných map v aplikaci je možné díky knihovně zmíněné v předchozí kapitole. Tato knihovna poskytuje mnoho funkcí, které lze volat a díky nim lze dosáhnout tíženého výsledku jako jsou například různé animace na mapě, přechody, změna vrstvy map, informace o dopravě, poloha uživatele atd.

V aplikaci je vytvořena třída, založená na `StatefulWidget`, která obsahuje samotnou mapu jako jeden z prvků, který se zobrazuje na obrazovce. Kvůli tomu, že na obrazovce je zobrazeno více vrstev widgetů (tlačítka pro polohu uživatele, vrstvy mapy, app bar…), které se navzájem překrývají, byla využita třída `Stack`, díky které lze widgety organizovat i tímto způsobem (widgety jsou naskládány přes sebe). Jednoduché zobrazení mapy lze dosáhnout pomocí následujícího kódu (Obrázek 17 - Použití widgetu GoogleMap).

```
GoogleMap(
    onMapCreated: (GoogleMapController controller) {
      _mapController = controller;
      _controller.complete(controller);
    }
)
```

Obrázek 17 - Použití widgetu GoogleMap

Tento kód však vytvoří pouze základní mapu, která má výchozí nastavení a není pro většinu případů moc užitečná, proto se v následujícím kódu (Obrázek 18 - Použití widgetu GoogleMap s konfigurací vlastností) doplní tento widget také o další nastavení a vlastnosti. Mimo samotných parametrů pro zobrazení mapy se u tohoto widgetu konfiguruje také jedna z jeho metod `onMapCreated`, což je callback metoda, s jedním vstupním parametrem typu `GoogleMapController`, která se volá po vytvoření/vykreslení mapy na obrazovku. V tomto případě se nastaví `controller` do globální proměnné, aby byl dostupný v celé třídě (nejen jako lokální proměnná v rámci callback funkce), protože lze díky němu na mapě volat další funkce, které knihovna nabízí (pohyb kamery s animací, vykreslování prvků na mapě atd.).

```
GoogleMap(
    zoomControlsEnabled: false,
    mapToolbarEnabled: false,
    myLocationEnabled: true,
    trafficEnabled: isTrafficVisible,
    minMaxZoomPreference: MinMaxZoomPreference(10, 18),
    myLocationButtonEnabled: false,
    mapType: _currentMapType,
    markers: Set<Marker>.of(markers.values),
    initialCameraPosition: CameraPosition(
        target: LatLng(49.226860, 17.668961),
        zoom: 11,
    ),
    onTap: (_) => _hideInfoWindow(),
    onMapCreated: (GoogleMapController controller) {
        _mapController = controller;
        _mapController.setMapStyle(isDark ? _mapStyleDark : _mapStyleLight);
        _controller.complete(controller);
    },
)
```

Obrázek 18 - Použití widgetu GoogleMap s konfigurací vlastností

Zde je widget rozšířený o potřebné argumenty, díky kterým se nastavuje samotné vykreslení mapy. Parametry mají následující význam a funkcionalitu.

`zoomControlsEnabled` – viditelnost tlačítek pro ovládání přiblížení/oddálení mapy

`mapToolbarEnabled` – viditelnost tlačítek pro přepnutí do aplikace Google Maps

`myLocationEnabled` – možnost zobrazení aktuální polohy uživatele

`trafficEnabled` – viditelnost dopravních informací, která se řídí na základě uživatelských preferencí z nastavení aplikace

`minMaxZoomPreferences` – minimální a maximální hodnota pro přiblížení/oddálení mapy

`myLocationButtonEnabled` – viditelnost tlačítka pro zobrazení polohy uživatele

`mapType` – typ mapových podkladů (letecká, turistická…)

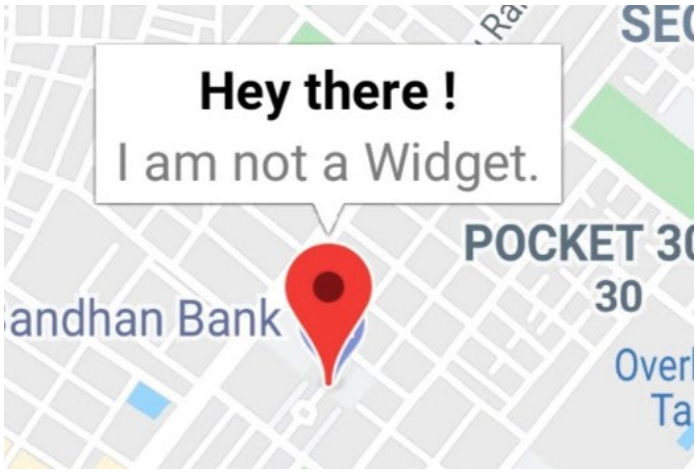
`markers` – značky, které jsou vidět na mapě (jedná se o značky samotných vozidel)

`initialCameraPosition` – výchozí lokace na mapě, která se vykreslí po prvním načtení

`onTap` – callback funkce, která se volá vždy, když uživatel klikne na mapu (např. se schová informační okno, které poskytuje detaily o vybraném vozidle)

`onMapCreated` – funkce popsána v kapitole výše je rozšířena o nastavení barevného režimu, jehož hodnota je načtena z uživatelských preferencí z nastavení (tmavý a světlý)

Dalším prvkem je již zmíněné informační okno `InfoWindow`, které se zobrazí po kliknutí na značku `Marker`. Vzhledem k tomu, že samotná knihovna neposkytuje možnost výchozí vzhled informačního okna přizpůsobit, bylo nutné vytvořit vlastní řešení, které bude odpovídat požadavkům a definovanému vzhledu aplikace. Co se týče značek, které reprezentují vozidla na mapě, ani zde není použit výchozí vzhled, nýbrž vlastní design. Původní vzhled značek je však možné jednoduše měnit díky podpoře knihovny.




Obrázek 19 - Výchozí vzhled značky a informačního okna [46]

## 6.4 Vlastní informační okno s informacemi o vybraném vozidle

Vzhledem k tomu, že knihovna neumožňuje větší nastavení vzhledu a parametrů výchozího informačního okna, které se zobrazí po kliknutí na značku, bylo implementováno řešení, které odpovídá všem požadavkům tak, aby uživatel aplikace viděl veškeré informace o zvoleném vozidle a zároveň okno nebralo velké množství místa na obrazovce a s mapou se dalo dále pohybovat.

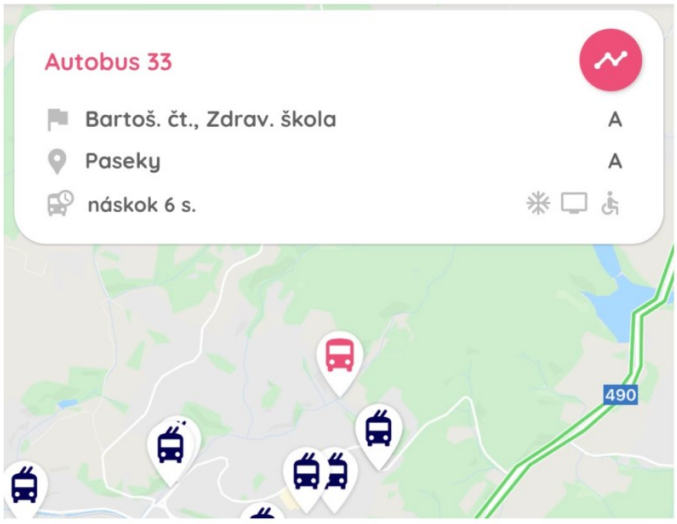
Funkce pro zobrazení informací o vozidle funguje tak, že si uživatel vybere na mapě jednu ze značek (případně vůz najde pomocí vyhledávání) a v horní části obrazovky pod hlavním panelem se zobrazí informační okno, ve kterém se nachází informace jako je zpoždění, spoj, cílová a poslední navštívená stanice, možnost zobrazení jízdního řádu s vizualizací polohy a také služby a funkce ve vozidle jako je klimatizace, bezbariérový přístup, obrazovka atd. (Obrázek 20 - Vybrané vozidlo a vlastní informační okno).

Při implementaci vlastního informačního okna byl v projektu nejprve vytvořen nový `stateful` widget, který má v konstruktoru několik parametrů, které předávají do této třídy objekt s informacemi o vozidle, číselnou proměnnou, podle které se řídí, jestli je informační okno na obrazovce vidět a callback funkci pro vycentrování mapy, která se volá tehdy,

když uživatel klikne na informační okno (pokud např. uživatel posune mapu tak, že není zvolené vozidlo na mapě vidět, kliknutím na informační okno se mapa opět vycentruje a vozidlo bude vidět). Uvnitř této třídy je mimo prezentační logiku informací o vozidle také `AnimatedPositioned` widget, ve kterém lze nastavit jeho pozici na obrazovce v pixelech, která je řízená podle výše zmíněného parametru předávaného v konstruktoru (hodnota 0 znamená, že widget není na obrazovce vidět a např. hodnota 200 znamená, že má informační okno pozici 200 pixelů od horní části obrazovky).

Instance tohoto widgetu se vytváří po kliknutí na jednu ze značek a vzhledem k tomu, že se jedná o informace, které se mění během krátkého časového horizontu, bylo zde nutné vyřešit také dynamické aktualizování informací (poslední navštívená zastávka, zpoždění…). V třídě, která zobrazuje mapu, je implementován algoritmus, který sleduje, zda má uživatel informační okno aktivní, tím pádem má zvolený i vůz a pokud tomu tak je, při příchodu nových dat se vytvoří nová instance informačního okna, které obsahuje aktuální informace o vozidle stejně tak jako i aktualizované argumenty pro callback funkci, která se po kliknutí přesune na novou polohu vozidla. Staré informační okno je zničeno. Tato operace se děje v řádu jednotek milisekund, takže uživatel nemá šanci poznat, že se ve své podstatě vyměnilo celé informační okno za úplně nové.

V současné době aplikace podporuje jak regionální linky Zlínského kraje, tak linky MHD Zlín, proto jsou zde dvě varianty informačního okna, kdy každé je mírně přizpůsobeno pro daného poskytovatele dat (např. u regionálních linek nejsou v datech dostupné informace o zónách zastávek nebo některé informace o funkcích vozidla). Aplikační logika si sama řídí, které okno se má zobrazit.




Obrázek 20 - Vybrané vozidlo a vlastní informační okno

## 6.5 Vlastní značky

Knihovna pro podporu práce s Google Maps nabízí základní vzhled značky (marker), který však neodpovídá vzhledu a funkcionalitě, která byla požadována. Bylo tedy nutné implementovat a vytvořit vlastní vzhled značek a integrovat je do aplikace. Google Maps knihovna naštěstí podporuje možnost vlastních značek, což práci výrazně zjednodušilo.

Jedna značka v aplikaci reprezentuje vozidlo, resp. jeho aktuální polohu na mapě. Třída `Marker` má vlastnosti jako `position`, `markerId`, `onTap`, `icon` a další. Hodnoty těchto vlastností jsou v aplikaci nastaveny a předány přes konstruktor. Hodnota vlastnosti `position` reprezentuje reálné souřadnice (zeměpisná šířka a délka) vozidla na mapě, které se v průběhu času aktualizují, a tím se posouvá i značka na mapě. Vlastnost `markerId` je unikátní identifikátor značky (i vozidla), který je na mapě – nelze mít dvě značky (vozidla) se stejným identifikátorem – značky se navíc ukládají do datové struktury `hashSet`, který duplicity neumožňuje. Další součást třídy je callback metoda `onTap`, která se volá vždy, když uživatel klikne na zvolenou značku. V tomto případě tato metoda zobrazí informační okno pro dané vozidlo. Co se týče ikony, jak již bylo popsáno v kapitole výše, výchozí vzhled neuspokojil potřeby aplikace, proto byl vytvořen set ikon, které reprezentují vozidla a jiné body na mapě. Každá ikona má několik variant – pro tmavý, světlý režim a také barvu pro aktivní a neaktivní ikonu (Obrázek 21 - Vlastní ikony značek na mapě).

Obrázek 21 - Vlastní ikony značek na mapě

Vozidlo, potažmo značka, má jinou barvu, pokud je uživatelem zvolená, aby se odlišila od ostatních a uživatel jasně věděl, kde se vozidlo na mapě nachází. Knihovna bohužel v aktuální verzi nepodporuje dva typy/styly pro jednu značku na základě podmínky nebo

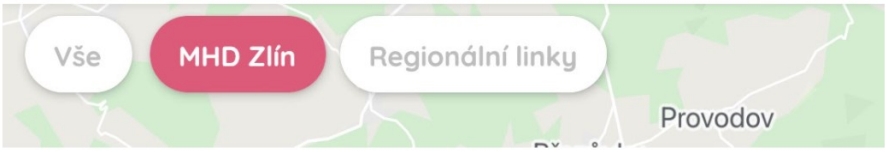
nějaké proměnné, tudíž bylo nutné tuto funkcionalitu naprogramovat manuálně. Když uživatel vybere vozidlo, na pozadí se z kolekce značek na mapě vybraná značka odebere a nahradí se novou, která má stejné vlastnosti jako ta původní s tím rozdílem, že má jinou ikonu. Toto nahrazení probíhá velmi rychle a uživatel nepozná, že se ve skutečnosti jedná o úplně jinou značku, než byla ta původní. Když potom uživatel vybere nový vůz, proces se opakuje. Když přijdou nová data ze serveru, v aplikaci se kontroluje, zdali není nějaký vůz vybraný. Pokud vybraný je, jeho poloha se nahradí značkou s ikonou pro vybraný vůz (tato značka je vyhledána na základě unikátního identifikátoru) a ostatní značky se na mapu vykreslí s ikonou pro nevybrané vozidlo. Pokud má uživatel vybraný vůz a sleduje jeho polohu na mapě a vůz např. dojede do destinace a z mapy zmizí, zobrazí se uživateli hláška, že tento vůz již není dostupný (vozidlo s identifikátorem a značkou, které bylo zvoleno, už není v aktualizovaných datech ze serveru dostupné).

Kvůli úspoře místa a výkonu bylo nutné tyto ikony zmenšit a komprimovat na pokud možno co nejmenší velikost, aby se načítaly rychle a před jejich načtením se nezobrazoval výchozí vzhled ikon, což na uživatele aplikace nepůsobí dobrým dojmem. Ke kompresi byl využit online nástroj *https://compresspng.com*, kde stačí původní ikony nahrát a nástroj se automaticky postará o kompresi. Vzhledem k tomu, že tyto ikony jsou v aplikaci zmenšené z původních 512 pixelů na základě poměru stran (aspect ratio) obrazovky zařízení na 124 nebo 95 pixelů, pro uživatele nebude komprese viditelná, a to ani na velkých displejích. Jako formát pro ikony byl zvolen PNG, který umožňuje mít průhledné okraje kolem samotné ikony. Před kompresí měly ikony velikosti od 70 kB do 150 kB a po kompresi se velikost zmenšila do rozmezí od 10 kB do 40 kB.

Renderování/vykreslování ikon je řešeno knihovnou pro podporu práce s Google Maps, která automaticky vykresluje pouze ty ikony, které jsou vidět na obrazovce. Tento jev je pozorovatelný zvlášť na telefonech s operačním systémem Android, a to tak, že když uživatel přejíždí s mapou, ikony blížící se k okraji obrazovky postupně mizí a na druhé straně zase přibývají. Tato funkce je zde hlavně kvůli optimalizaci výkonu aplikace, protože pokud by na mapě bylo vykresleno velké množství objektů, telefon by takovou zátěž nemusel zvládat. V aplikaci bývá v dopravní špičce (ráno, odpoledne) při zvolení více zdrojů dat až 400 značek, což by už uživatelé na zařízení se slabším hardware mohli poznat. Další krok, který pomáhá s výkonem a plynulostí aplikace je omezení maximálního oddálení mapy tak, aby nikdy nebyly vidět všechny značky na mapě, ale jen část a pokud uživatel potřebuje zobrazit jinou část mapy, stačí, aby se jen přesunul a značky se překreslí na základě vybrané oblasti.

## 6.6 Zdroje dat

Aplikace aktuálně podporuje zobrazení dat o poloze vozů regionálních linek a linek vozů MHD Zlín. Uživatelé mají v aplikaci možnost vybrat si, ze kterého zdroje chtějí data na mapě vidět – vše, regionální linky či MHD Zlín (Obrázek 22 - Výběr zdroje dat na mapě).




Obrázek 22 - Výběr zdroje dat na mapě

Po zvolení zdroje dat se v aplikaci začne vykonávat periodická funkce, která implementuje logiku načítání dat pro zvolený zdroj – každý zdroj dat má jiný čas obnovování, který je stanovený poskytovatelem. U regionálních linek se data v aplikaci obnovují každých 15 vteřin a u linek MHD Zlín to je 25 vteřin. V aplikaci je ošetřen také potencionální únik paměti, který by mohl nastat při neukončení periodické funkce – proto, když uživatel opouští stránku s mapou, je volána funkce `timer.cancel()`, která periodickou funkci zastaví. Rušení se volá v jedné z widget lifecycle metodě `void dispose()`, která se provádí vždy, když se widget maže z widget tree (widget není vidět na obrazovce). Při zvolení možnosti "Vše" jsou aktivní oba časovače a data se obnovují zvlášť.

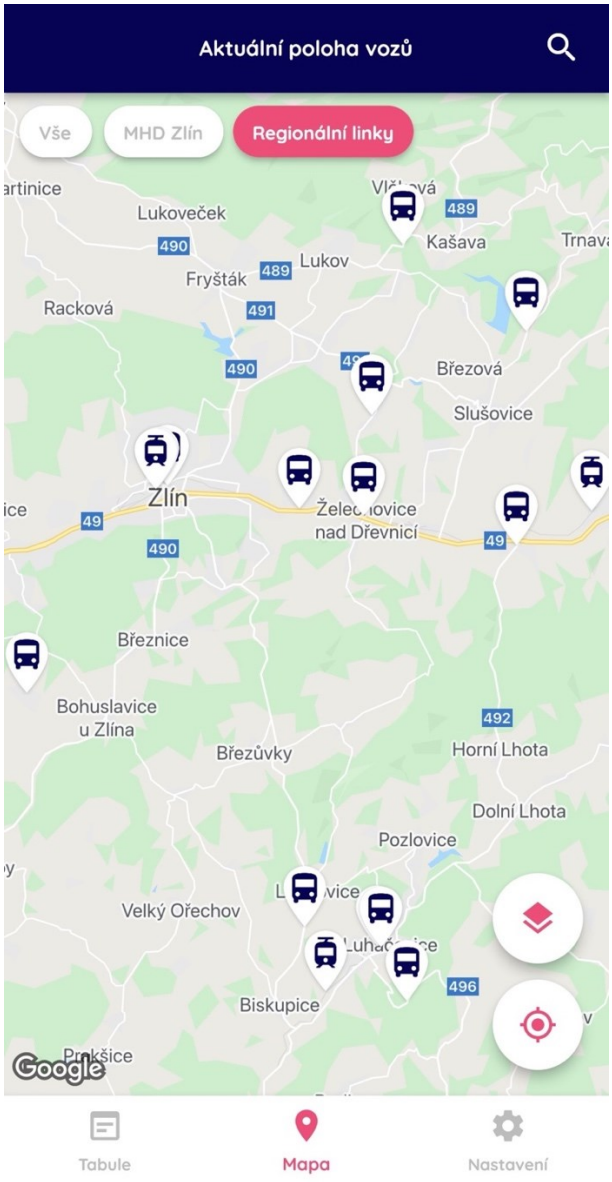
Aby aplikace nezatěžovala servery poskytovatelů dat, dotazuje se na vlastní backend, kde jsou data uložená v cache paměti a také se zde parsují a zpracovávají do požadovaného formátu, který aplikace využívá – např. poskytovatelé dat vrací jako zpoždění textový řetězec, který obsahuje různé znaky (`$z1~min.~56~s`), které nejsou pro uživatele relevantní. Z toho důvodu tento textový řetězec backend upravuje tak, aby mu uživatel plně rozuměl a přinesl mu relevantní informace o reálném zpoždění vozidla – `zpoždění 1 min. 56 s`.

Backend využívá tzv. in-memory cache, která uchovává data vrácená od jednotlivých poskytovatelů (data jsou uložena v paměti v rámci aplikace – tato varianta cache paměti je jedna z nejrychlejších cache, pokud by aplikace využívala databázi, čas odpovědi by se v mobilní aplikaci zvýšil). V každé cache paměti je nastaven čas, po který jsou data platná a validní. Vzhledem k tomu, že jsou tato data platná po poměrně krátkou dobu (15 a 25 vteřin), po každé revokaci dat se první uživatel dotazuje na server poskytovatelů dat. Následně se tato data zpracují a uloží do cache paměti v backend aplikaci a dalším uživatelům, kteří se budou dotazovat na polohu vozů, se už data vrátí z backendu a ne ze serverů poskytovatelů dat. Výhodou tohoto přístupu je jednak zmenšení zátěže na servery poskytovatelů

dat a také zkrácení času odpovědí pro uživatele v mobilní aplikaci (první odpověď, která musí přijít až ze serveru poskytovatelů dat a následně projít přes backend, má latenci zhruba 400 ms, odpověď, která se vrací z cache má okolo 40 ms na rychlém 3G připojení).

## 6.7 Vyhledávání vozidel na mapě

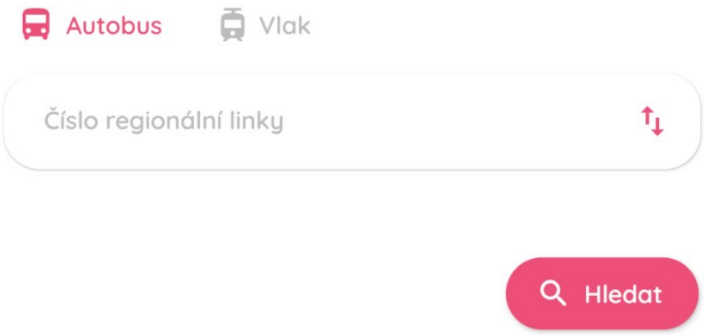
Uživatelé aplikace mohou vyhledávat vozidla také podle jejich čísla linky případně čísla spoje. Toto vyhledávání je dostupné v horní liště na záložce s mapou v aplikaci a po kliknutí na ikonu lupy (Obrázek 23 - Mapa s vozidly). Vyhledávání však není dostupné, pokud má uživatel jako zdroj dat vybranou možnost „Vše“.




Obrázek 23 - Mapa s vozidly

Pokud uživatelé například jezdí pravidelně autobusovým spojem s číslem 350, nemusí vozidlo na mapě hledat přes příjezdovou/odjezdovou stanici, respektive vozidlo hledat přímo na mapě, stačí když číslo linky napíší do vyhledávače (Obrázek 24 - Vyhledávač vozidel regionálních linek) a následně se zobrazí výsledky, kde si u daného vozidla mohou vybrat buď zobrazení jízdního řádu nebo aktuální polohu na mapě. Vyhledávač vozidel funguje pouze na právě aktivní vozidla jak regionálních linek, tak linek MHD Zlín, které jsou viditelné na mapě. Tlačítko vyhledávání je dostupné, pokud má uživatel vybraný jako zdroj dat buď MHD Zlín nebo Regionální linky. V samotném vyhledávání si u regionálních linek může ještě vybrat, o jaký typ vozidla se jedná – zdali jde o vlakový či autobusový spoj.

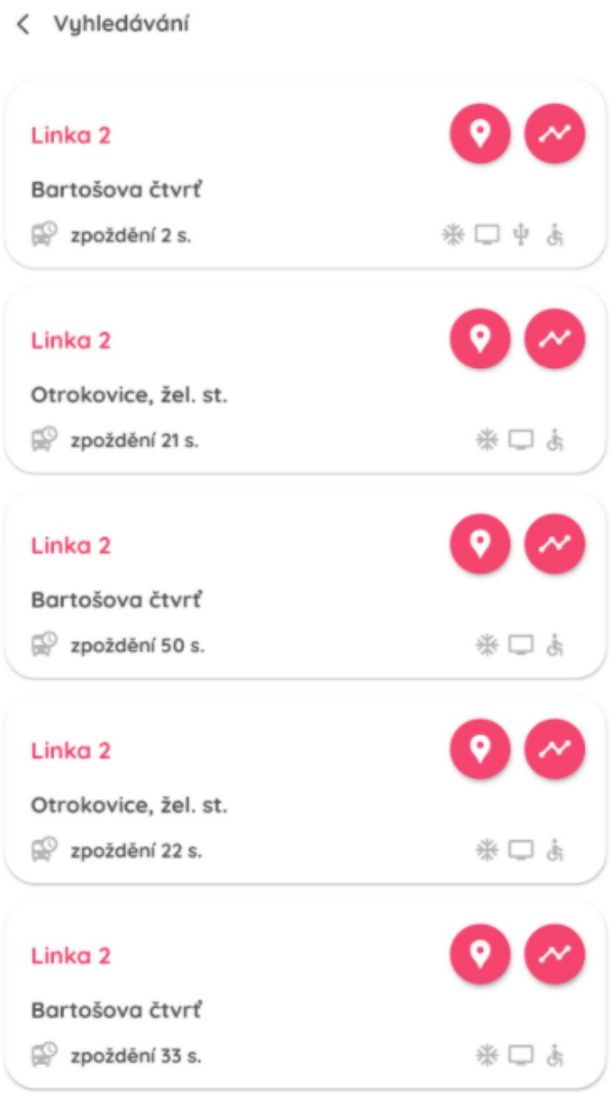
Při samotném vyhledávání se odešle dotaz na server, kde se následně načtou data buď z cache paměti nebo ze serverů poskytovatele, pokud je cache prázdná nebo neplatná. Následně se pomocí filtrování vyberou pouze ty spoje, které obsahují nebo mají konkrétní číslo linky, které uživatel hledal a vrátí se zpět do mobilní aplikace.




Obrázek 24 - Vyhledávač vozidel regionálních linek

Po vyhledání se zobrazí výsledky (Obrázek 25 - Výsledky vyhledávání), kde jsou všechny nalezené spoje s vyhledaným číslem linky. Zde je výsledný list vzhledově podobný jako informační okno, které je zobrazeno, když má uživatel zvolené vozidlo na mapě. Pokud by si uživatel přál zadat nové vyhledávání linky, je zde také tlačítko pro návrat zpět.

Na kartách s výsledky je zobrazena linka spoje, cílová stanice, aktuální zpoždění, vybavení vozu (obrazovka, klimatizace…), možnost zobrazení jízdního řádu a také přepnutí se na aktuální polohu spoje na mapě.

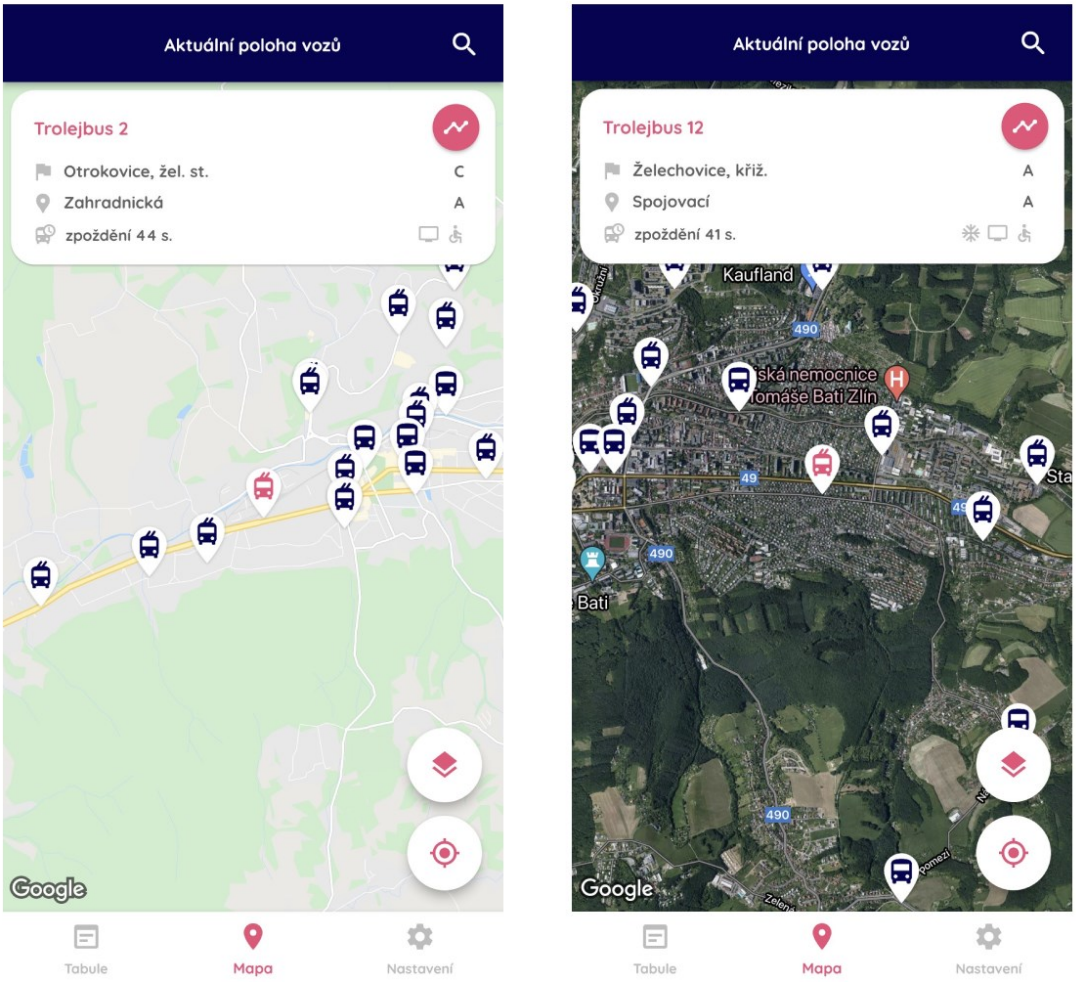



Obrázek 25 - Výsledky vyhledávání

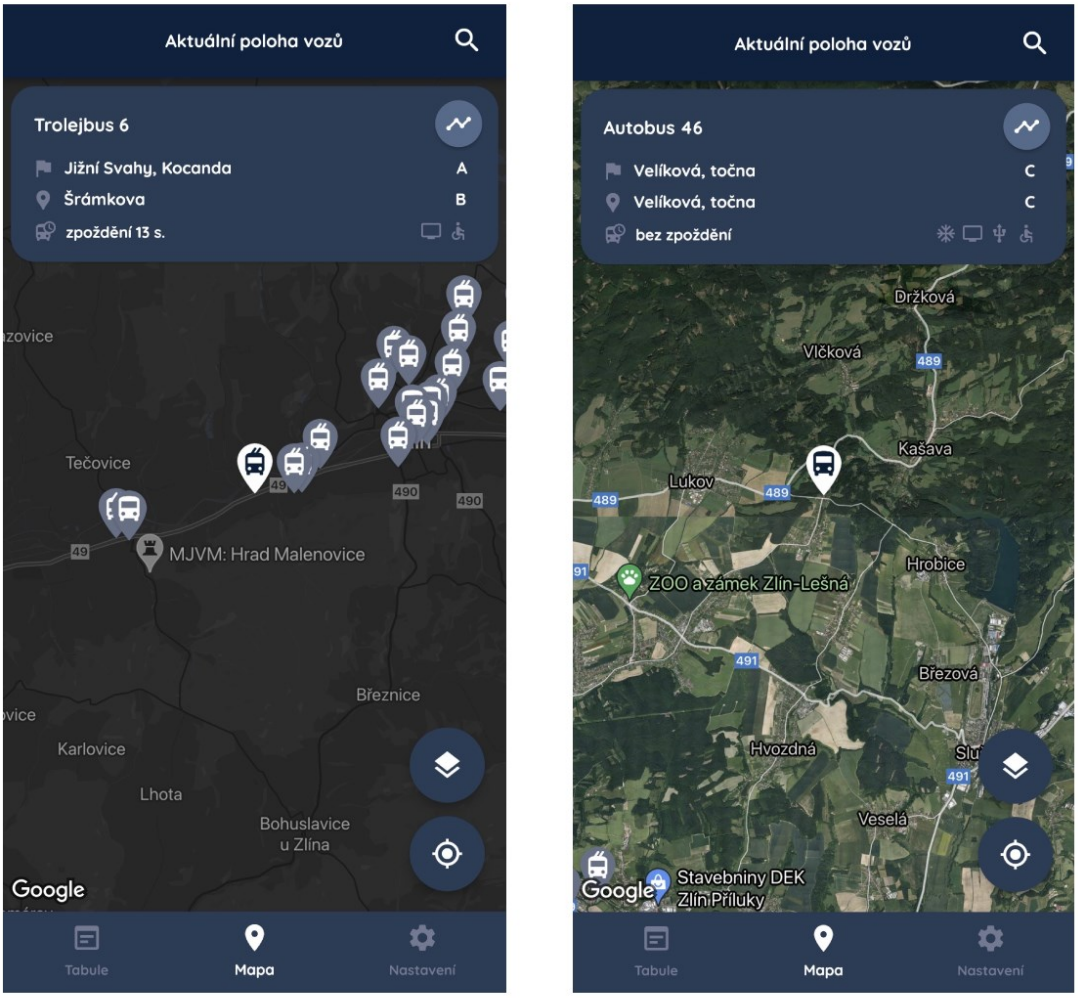
Implementace tohoto vyhledávače a filtru přinesla během vývoje několik výzev a problémů, které bylo nutné vyřešit. Jeden z největších problémů byl výběr widgetu, na kterém bude filtr zobrazen. U spousty widgetů, které byly v prvních verzích použity, byl problém v nedostatku místa – např. u widgetu `AlertDialog` nebylo možné na obrazovku přidat veškeré důležité informace, potřebné pro vyhledávání, a to ani po upravení výchozího vzhledu, kdy se navíc widget začal chovat jinak, než se očekávalo (na obrazovce bylo vidět blikání apod.). Podobný problém nastal také při použití widgetu `BottomSheet`, který by se do daného kontextu hodil nejvíce. Podrobné informace o tom, proč docházelo k těmto chybám, bohužel nejsou známy, nicméně po testování bylo odhaleno, že jsou problémy způsobeny pravděpodobně tím, že na pozadí jsou dynamicky se měnící widgety (mapa).

## 6.8 Změna podkladů a barev mapy

V rámci map je možné si vybrat, jaké mapové podklady se budou uživatelům vykreslovat. Pokud například uživatel potřebuje vidět, kde se přesně vozidlo nachází, může si zvolit letecké mapové podklady, kde to lze zjistit s větší přesností. Naopak, pokud uživatel preferuje spíše přehlednější a jednodušší vzhled map, může si zvolit výchozí typ mapy. Tato funkce je přímo podporována knihovnou pro práci s Google Maps, takže stačí pouze nastavit typ z výčtového typu `MapType`. Na základě uživatelského nastavení se mění také barva výchozích map mezi tmavým a světlým režimem (Obrázek 26 - Výchozí (vlevo) a letecký (vpravo) podklad map ve světlém režimu, Obrázek 27 - Výchozí (vlevo) a letecký (vpravo) podklad map v tmavém režimu). S tímto nastavením se mění i barvy značek reprezentující vozidla – mění se jak vybrané, tak nevybrané vozy. U leteckého režimu je toto nastavení a vzhled map stejný, zde se mění pouze značky vozidel a zbytek rozhraní.




Obrázek 26 - Výchozí (vlevo) a letecký (vpravo) podklad map ve světlém režimu

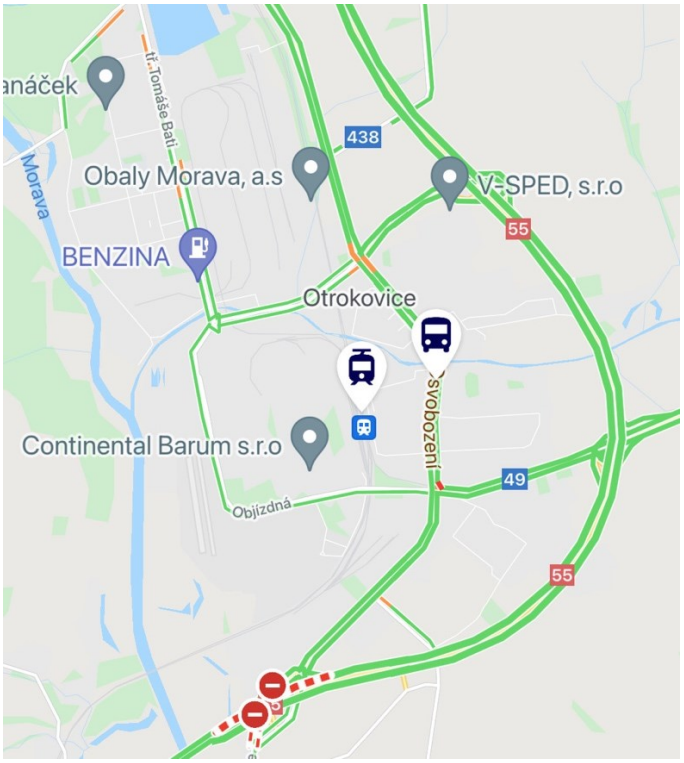



Obrázek 27 - Výchozí (vlevo) a letecký (vpravo) podklad map v tmavém režimu

## 6.9 Dopravní informace na mapě

Funkce zobrazení aktuálního dopravního vytížení na mapě se může uživatelům hodit, pokud například sledují svůj spoj, který má přijet na vybranou zastávku. Díky této funkci mohou přesněji zjistit, že se autobus nachází např. v koloně aut, a tím pádem bude mít zpoždění. Zobrazení dopravních informací je možné zapnout nebo vypnout na základě uživatelských preferencí v nastavení aplikace. Tato uživatelská preference je uložena v paměti telefonu, tudíž se nastavení nevymaže, pokud uživatel aplikaci vypne a znovu ji zapne.

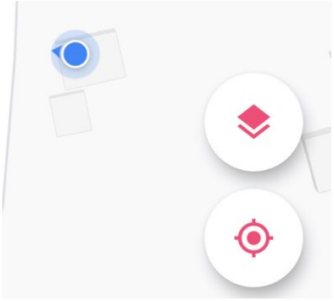
Aktuální dopravní informace poskytuje přímo Google a následně jsou v aplikaci promítnuty díky knihovně pro práci s Google Maps – hustá a velká doprava se značí červenou barvou, středně hustá doprava oranžově a zeleně je potom znázorněna mírná doprava. Na mapách se zobrazují také informace o uzavírkách nebo práci na silnici (Obrázek 28 - Dopravní informace na mapě).




Obrázek 28 - Dopravní informace na mapě

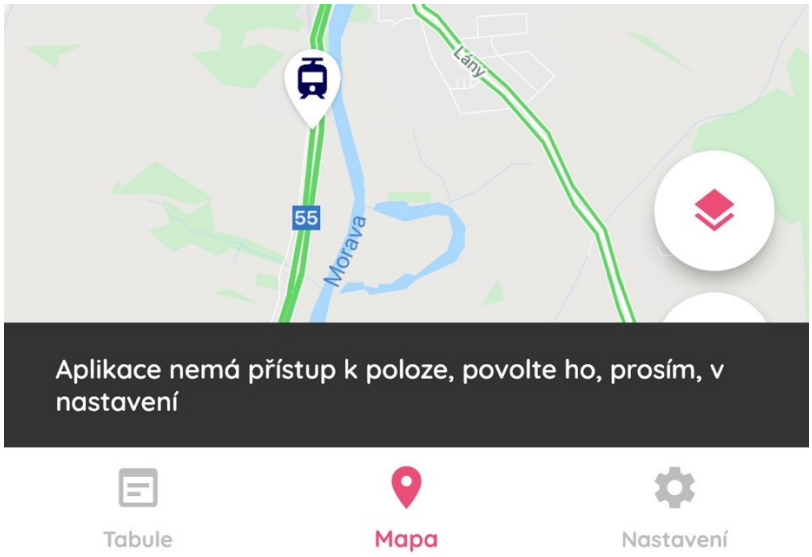
## 6.10 Aktuální poloha uživatele

Uživatel může zaměřit svoji polohu na mapě pomocí tlačítka, které je umístěno v pravém dolním rohu mapy. Tato funkce se může uživatelům hodit například tehdy, pokud jsou ve městě či vesnici a nejsou si přesně jisti, kde se nachází. Dále to uživatelům může pomoci sledovat na mapě spoj, ve kterém právě jedou, což se může hodit, když jedou do neznámé destinace a potřebují sledovat, kolik cesty jim ještě zbývá. Díky této funkci se uživatelům na mapě zobrazí modrý kruh, který reprezentuje jejich polohu. Kolem tohoto kruhu je malá šipka, která znázorňuje směr, kterým se uživatel dívá a má zařízení natočené (Obrázek 29 - Tlačítko pro zaměření polohy uživatele a jeho poloha na mapě).



Obrázek 29 - Tlačítko pro zaměření polohy uživatele a jeho poloha na mapě

Při prvním spuštění aplikace a kliknutí na tlačítko polohy bude uživatel dotázán, zda by udělil přístupová práva aplikaci k jeho datům z GPS modulu v telefonu, bez čehož by tato funkce v aplikaci nefungovala. Pokud uživatel udělení práv aplikaci zamítne, zobrazí se dole widget, který informuje o důvodu nefunkčnosti této funkce a o možném řešení – povolení polohy v nastavení zařízení (Obrázek 30 - Hláška o nepovoleném přístupu k poloze zařízení).

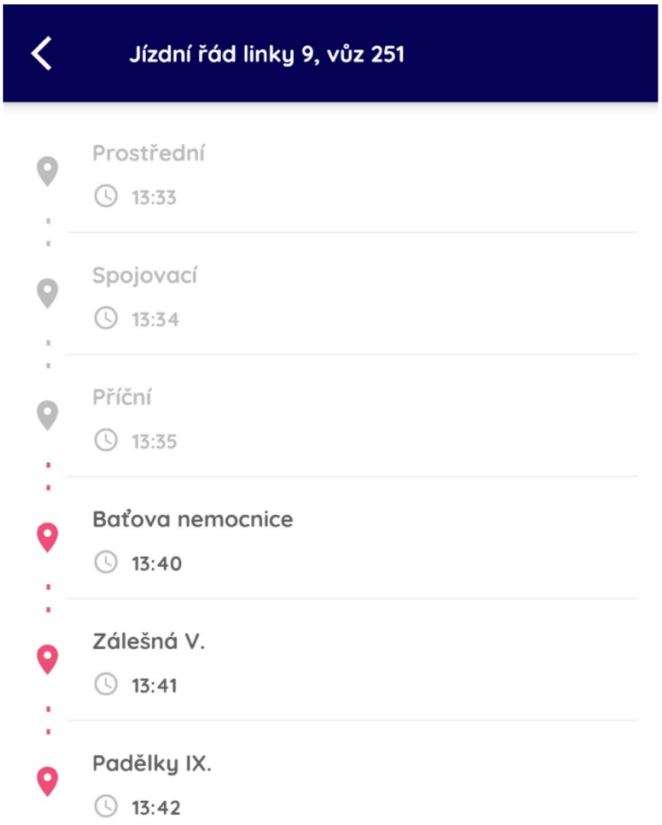



Obrázek 30 - Hláška o nepovoleném přístupu k poloze zařízení

Samotná funkcionalita je implementována pomocí další knihovny, kterou bylo nutné přidat do projektu – `geolocator`. Tato knihovna umožní aplikaci načítat periodicky data z GPS modulu v zařízení a aktualizovat polohu v reálném čase, takže pokud například uživatel jde k zastávce a sleduje mapu, bude se modrý kruh reprezentující jeho polohu na mapě pohybovat s ním. Tato knihovna je multiplatformní, což znamená, že funguje bez nutnosti velké konfigurace jak na operačním systému iOS, tak Android.

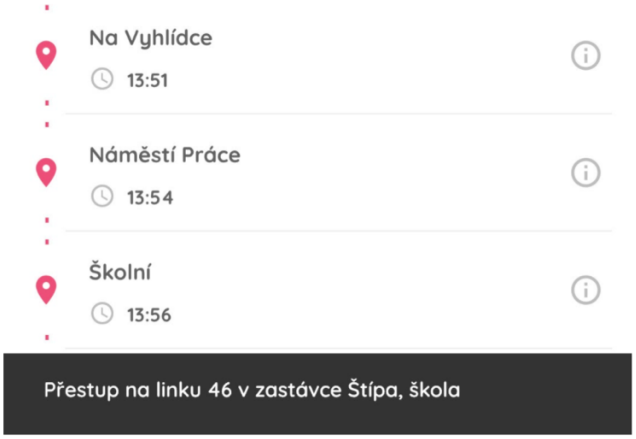
## 6.11 Jízdní řády a vizualizace polohy

U vozidel (potažmo linek a spojů) je dostupný také jejich aktuální jízdní řád, který nabízí uživatelům přehled, v jakém čase bude vozidlo na vybrané zastávce, takže si mohou lépe načasovat, kdy například na zastávku vyrazit. Na obrazovce s jízdním řádem je také vidět, kterými zastávkami už vozidlo projelo, což dá uživateli lepší přehled o jeho poloze a době, která mu zbývá, než do destinace přijede. Zastávky, které vozidlo už navštívilo, jsou barevným kontrastem odděleny od těch, které teprve vozidlo navštíví. U jednotlivých stanic je vidět čas odjezdu vozidla a u poslední, cílové zastávky, která je na jízdním řádu, je vidět čas příjezdu (Obrázek 31 - Jízdní řád a vizualizace polohy).




Obrázek 31 - Jízdní řád a vizualizace polohy

Jízdní řády a vizualizace jsou dostupné jak u regionálních linek, tak u MHD Zlín. V rámci zlínské MHD jsou také mezi daty dostupné informace, jako například výluky, přestupy na jinou linku a další informace. Pokud linka obsahuje tato data a doplňující informace, jsou u jednotlivých zastávek zobrazeny ikony, které po kliknutí zobrazí v dolní části aplikace `snackBar` widget, který uživateli tyto informace zobrazí (Obrázek 32 - Doplňující informace u jízdního řádu).

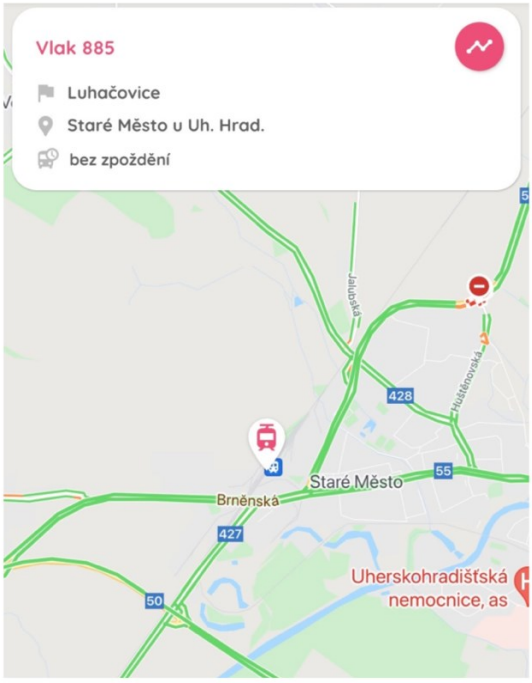



Obrázek 32 - Doplňující informace u jízdního řádu

## 6.12 Sledování jednoho vozidla na mapě

V aplikaci je možnost sledovat polohu pouze jednoho vybraného vozidla na mapě. Tato funkce může být užitečná tehdy, pokud uživatel chce na mapě vidět pouze jedno vozidlo a zbytek vozidel ho nezajímá, tím pádem je mapa více přehledná (Obrázek 33 - Sledování jednoho vozidla na mapě). Tato funkce je dostupná v rámci odjezdových tabulí, kde se u výsledků nabídne možnost si dané vozidlo na mapě zobrazit, pokud je aktivní – tzn. vozidlo už vyjelo z výchozí stanice, je na cestě, a je dostupné na mapě. Tato stránka je samostatná a oddělená od mapy se všemi vozidly, nicméně poskytuje většinu funkcí, jako mapa, kde jsou vidět všechna vozidla (krom vyhledávání). Funkce sledování jednoho vozidla je podporována pouze pro regionální linky z důvodu nedostatku dat u MHD Zlín, kde tuto funkci nebylo možné implementovat. Když uživatel sleduje vozidlo, a to dojede do destinace (zmizí z mapy a nebude se vracet v odpovědi ze serveru), zobrazí se uživateli, stejně jako na mapě se všemi vozy, hláška o tom, že už vozidlo není dostupné.

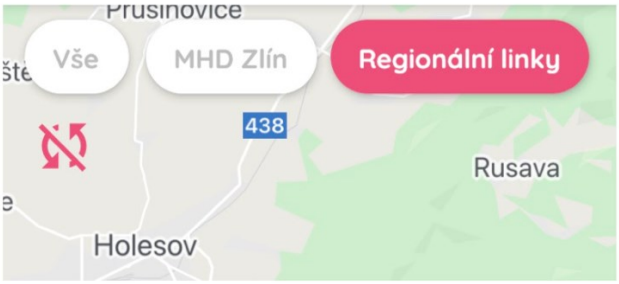
Z hlediska implementace je tato funkce opět velmi podobná, liší se pouze v tom, že se na server posílá periodicky dotaz s unikátním identifikátorem vybraného vozu. Server se následně podívá do kolekce v cache paměti nebo si aktualizuje data ze serveru a vrátí odpověď mobilnímu klientovi. Jako datová struktura je v cache použita `hashMap`, která má jako klíč unikátní identifikátor vozidla, což značně ušetří čas procházení a hledání ve všech datech.




Obrázek 33 - Sledování jednoho vozidla na mapě

## 6.13 Nejčastější chyby a jejich ošetření

Někdy se může stát, že v aplikaci, ať už to je na straně mobilního klienta nebo na straně serveru, nastane chyba. Na straně klienta se nejčastěji jedná o přístup k internetu nebo nepovolená práva k přístupu k poloze. Pokud nemá telefon přístup k internetu, zobrazí se v levém horním rohu mapy červená ikona (Obrázek 34 - Ikona signalizující chybu na mapě), která signalizuje chybu a na mapě se nevykreslí žádná data (v podobě značek vozidel). Jednoduché řešení tohoto problému je zkontrolovat, zdali ostatní zařízení v síti přístup k internetu mají (pokud je telefon připojen přes Wi-Fi) nebo v případě mobilních dat, pokud je telefon v dosahu signálu a nevyčerpal datový limit od poskytovatele.




Obrázek 34 - Ikona signalizující chybu na mapě

Další chybou, která v rámci aplikace může nastat, je chyba na serveru, která se projeví tím, že nevrátí dotazovaná data, respektive vrátí odpověď, která má HTTP status kód jiný než 2xx. Tyto chyby nejčastěji nastávají tehdy, pokud jsou například servery poskytovatelů dat nedostupné, tzn. nevrací data nebo se změnil jejich formát, se kterým aplikace neumí pracovat (data se vrací ve specifickém formátu a pokud se tento formát změní, je nutné upravit i formátování a parsování na straně serveru).

Backend část aplikace je hostována na veřejném cloudu Heroku, který byl popsán v kapitole výše v teoretické části. Aplikace se pravidelně jednou za den restartuje, hlavně z důvodu uvolnění paměti a případnému vyřešení problémů. Vzhledem k tomu, že aplikace běží pouze v rámci jedné instance, může to znamenat chvilkovou nedostupnost a výpadek, než se aplikace znovu spustí, což trvá průměrně okolo půl minuty. Tento pravidelný denní restart je plánován na dobu, kdy se aplikace používá nejméně, což jsou zpravidla noční hodiny. Pokud se jedna z těchto chyb vyskytne, v rámci mobilní aplikace se na mapách projeví zobrazením červené ikony a v jiných částech aplikace buď informačním panelem (`snackBar` widget), anebo dlouhým točením `loading_spinner` widgetu.

# 7 ODJEZDOVÉ TABULE

Aplikace mimo jiné funkce nabízí také zobrazení virtuálních odjezdových tabulí, které poskytují uživatelům přehled o tom, kdy z vybrané zastávky odjíždí nebo naopak do ní přijíždí spoj. Tato funkce je v současné chvíli podporována pouze u spojů regionálních linek a MHD Uherské Hradiště. Co se týče MHD Zlín, tato funkce není v aplikaci implementována z důvodu nedostatku informací ze strany poskytovatele dat, nicméně v budoucnu je podpora MHD Zlín v odjezdových tabulích prioritou na seznamu dalších funkcí. Odjezdové tabule jsou v aplikaci hned na první obrazovce, která se uživateli zobrazí po spuštění – zde si může vyhledat, pro jakou zastávku chce zobrazit výsledky.

## 7.1 Vyhledávání

Vyhledávání v odjezdových/příjezdových virtuálních tabulích je uživateli prezentováno hned na první obrazovce, která se zobrazí po spuštění aplikace. Na této obrazovce se nachází textové vstupní pole, které čeká na text od uživatele. Vyhledávání nabízí funkci našeptávání, aby si uživatel mohl vybrat z listu zastávek, které se shodují s dotazem, který napsal do textového vstupního pole. Tento seznam zastávek, který se uživateli zobrazí po zadání textu, se vrací z backend části aplikace, která zadaný text zpracuje, převede ho do malého písma a následně přeposílá dotaz na server poskytovatele dat. V aplikaci je implementován u vyhledávání a našeptávání tzv. threshold, což znamená to, že vždy, když uživatel zadá nové písmeno, aplikace čeká 250 milisekund a až poté odešle dotaz na server. Je to primárně kvůli tomu, aby se dotazy neposílaly tak často, což by ve výsledku nemělo žádný efekt – takto uživatel zadá například řetězec “uherske” a potom se mu vrátí prvních 5 stanic, které obsahují tento text.

Seznam se zastávkami obsahující hledaný text se zobrazuje hned pod vyhledávacím políčkem (Obrázek 36 - Našeptávání a načítání výsledků pro stanici). U každého výsledku je také znázorněno, jestli se jedná o autobusovou nebo vlakovou stanici. Když uživatel čeká na odpověď našeptávání, v pravé části vyhledávacího políčka je zobrazený malý `loading_spinner` widget, který indikuje nedokončený dotaz na server a slouží pro informování uživatele o činnosti aplikace.

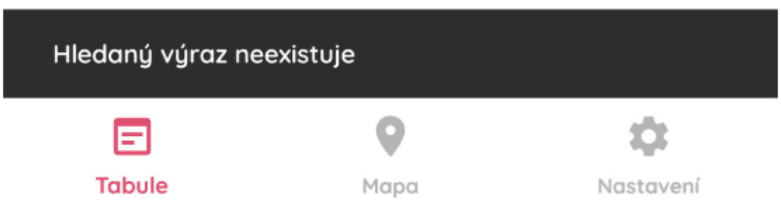
Malé omezení, které zde aplikace má, je samotná logika vyhledání, které je implementováno na straně serveru poskytovatele dat a nemohlo být nijak ovlivněno. Někdy se stává, že je potřeba zadat přesný název zastávky, aby uživatel našel opravdu tu zastávku nebo stanici, kterou hledá. Tento problém by se dal vyřešit například tím, že by byly všechny zastávky

uloženy v databázi, se kterou by napřímo komunikovala backend aplikace a nad touto databází by bylo implementováno lepší a pokročilé vyhledávání, které by vracelo přesnější a relevantnější výsledky. Problém s tímto řešením je ale to, že zastávky a informace o nich se často mění, tím pádem by v backend aplikaci musela být další logika, která by data ze serverů poskytovatelů dat pravidelně aktualizovala. Dále je také nutnost mít zaplacenou databázi nebo případně ji mít v paměti aplikace, což po testování nebylo možné hlavně kvůli nedostatku paměti v rámci Heroku cloudu a zvoleného plánu (výdaje za aplikaci by měly být co nejmenší).

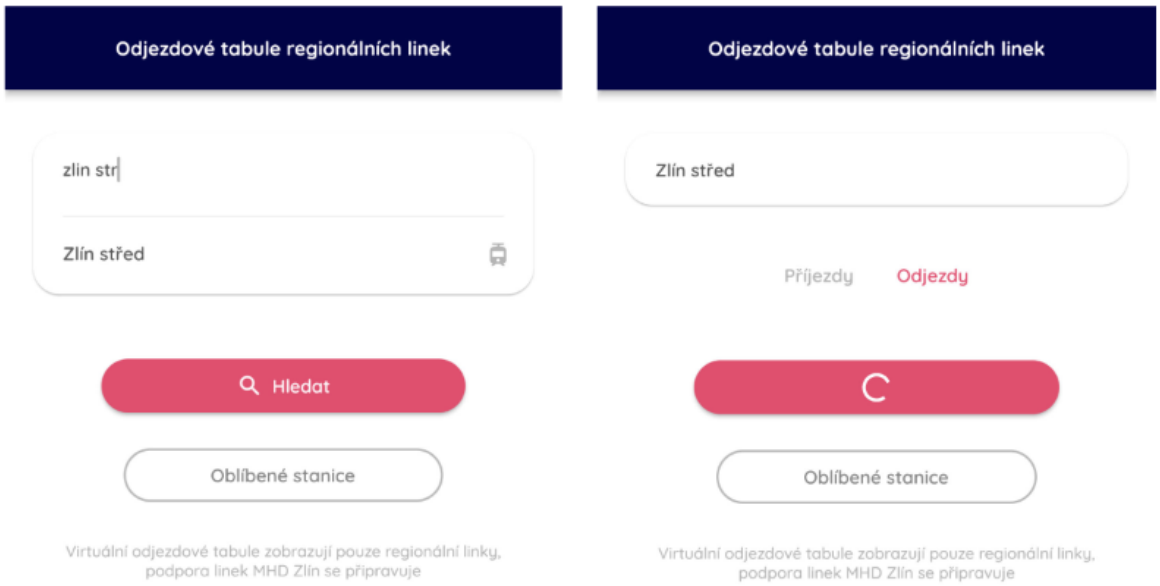
Jakmile se vrátí výsledky vyhledávání zastávek, uživatel musí vybrat, kterou zastávku hledal, čímž se na pozadí v aplikaci uloží další metadata o vybrané zastávce (např. unikátní identifikátor v systému) do proměnné, která se následně posílá v dalším dotazu na server. Pokud uživatel žádnou zastávku nezvolí a klikne na tlačítko vyhledávání, zobrazí se mu ve spodní části obrazovky validační chyba. Tato chyba se zobrazuje také tehdy, pokud uživatel zadá nějaký neexistující výraz (Obrázek 35 - Validační chyba).

Pod vyhledávacím políčkem jsou ještě na výběr dvě možnosti, a to zvolení mezi příjezdy nebo odjezdy z/do dané zastávky. Tato tlačítka jsou implementována jednoduše pomocí logiky tak, že vždy jedno z nich musí být zvoleno – takové tlačítko je zvýrazněno kontrastnější barvou, aby bylo uživateli jasné, jakou možnost má zvolenou. Následně stačí jen zmáčknout tlačítko vyhledat, které odešle dotaz na server se zvolenými parametry a uživateli je po vrácení odpovědi zobrazena stránka s výsledky. I zde platí, že když ještě není odpověď ze serveru vrácena, tak se v tlačítku zobrazí `loading_spinner` widget.

Níže pod tlačítkem vyhledávání se ještě nachází možnost zobrazení oblíbených stanic, které si uživatel může uložit, aby nemusel často hledanou stanici psát manuálně při každém hledání znovu. Na obrazovce se také nachází informační text o tom, že aktuálně jsou podporovány odjezdové tabule jen pro regionální linky (Obrázek 36 - Našeptávání a načítání výsledků pro stanici).



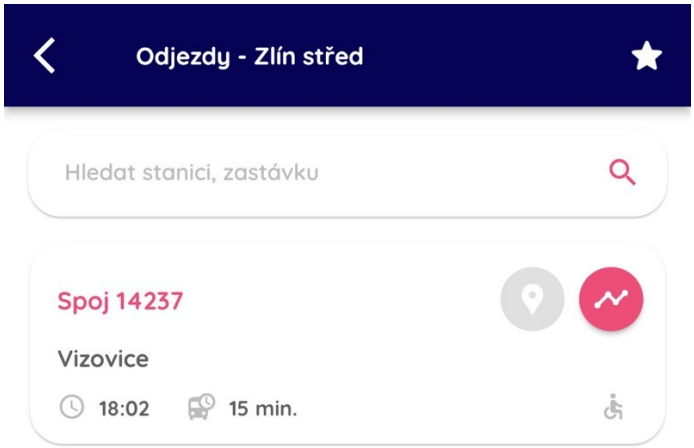

Obrázek 35 - Validační chyba




Obrázek 36 - Našeptávání a načítání výsledků pro stanici

## 7.2 Oblíbené stanice

V aplikaci je možné si přidat také oblíbené stanice a zastávky, což ve výsledku uživatelům ušetří čas a úsilí při novém vyhledávání. U výsledků vyhledávání stanic se v liště nahoře zobrazí ikony hvězdy (Obrázek 37 - Ikona pro přidání/odebrání stanice z oblíbených), která se po kliknutí změní a provede akci na základě toho, jestli zvolená stanice je nebo není mezi oblíbenými stanicemi uživatele.

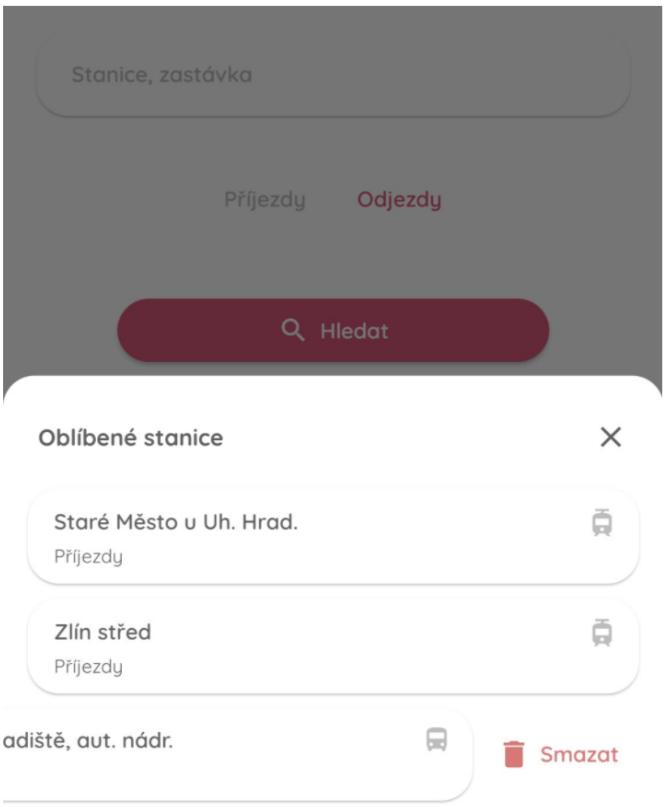



Obrázek 37 - Ikona pro přidání/odebrání stanice z oblíbených

Oblíbené stanice se stejně jako ostatní preference ukládají do lokálního úložiště zařízení. Všechny oblíbené stanice, které si uživatel uložil, je možné potom zobrazit hned na úvodní obrazovce, kde je pod hlavním tlačítkem vyhledávání další tlačítko oblíbených stanic (Obrázek 36 - Našeptávání a načítání výsledků pro stanici). Po kliknutí se uživateli zobrazí

widget `BottomSheet`, který v sobě všechny oblíbené stanice obsahuje. Pokud by uživatel chtěl nějakou stanici odebrat, může tak udělat pomocí tzv. “swipe gesta” směrem doleva přímo v tomto listu (Obrázek 38 - Oblíbené stanice s možností mazání). Další možností, jak vybranou stanici z oblíbených smazat, je právě pomocí ikony hvězdy na výsledku vyhledávání. Pokud by však uživatel chtěl vymazat všechny stanice, které má uložené, má tuto možnost v nastavení, kde je pro to přímo určena funkce. Po úspěšném odstranění či přidání stanice do oblíbených je uživatel vždy informován pomocí `snackBar` widgetu ve spodní části obrazovky.

Při ukládání do lokálního úložiště se samotný objekt vybrané zastávky serializuje do textového řetězce v JSON formátu a ten se následně ukládá do již existujícího pole s dalšími stanicemi. Pokud toto pole v paměti není, vytvoří se nové a uloží se do něj.

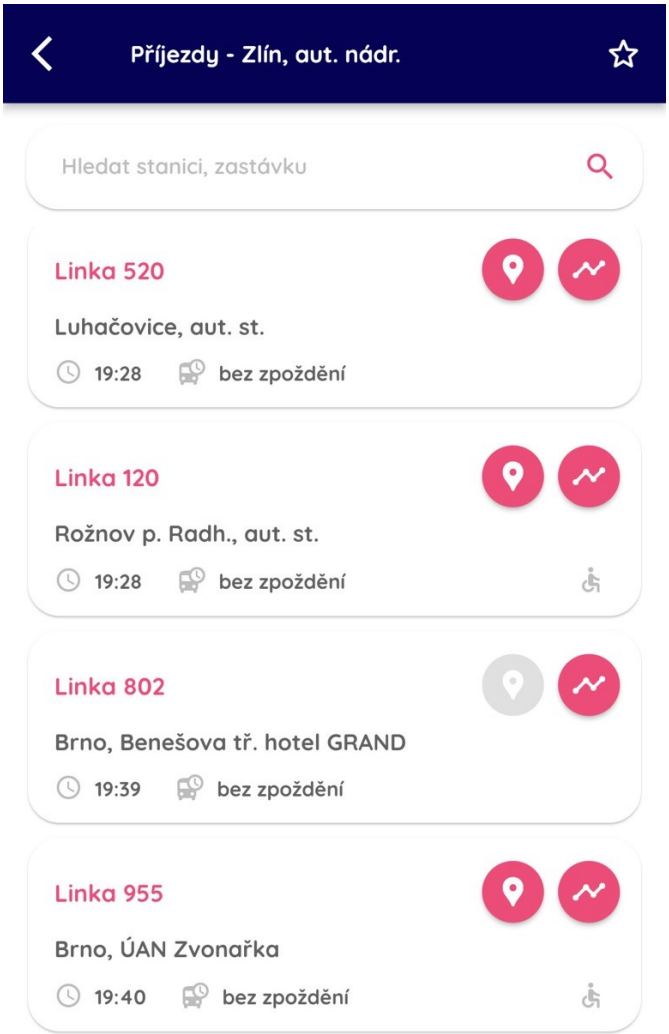



Obrázek 38 - Oblíbené stanice s možností mazání

## 7.3 Výsledky vyhledávání a filtrování

Po vyhledání a zvolení zastávky se uživateli zobrazí výsledky ať už to z příjezdové či odjezdové tabule s vypsanými spoji pro konkrétní stanici nebo zastávku. V těchto výsledcích může uživatel přehledně vidět informace nebo najít spoj, na který čeká. V případě příjezdových tabulí se u jednotlivých karet zobrazuje původní zastávka nebo stanice, ze které vozidlo

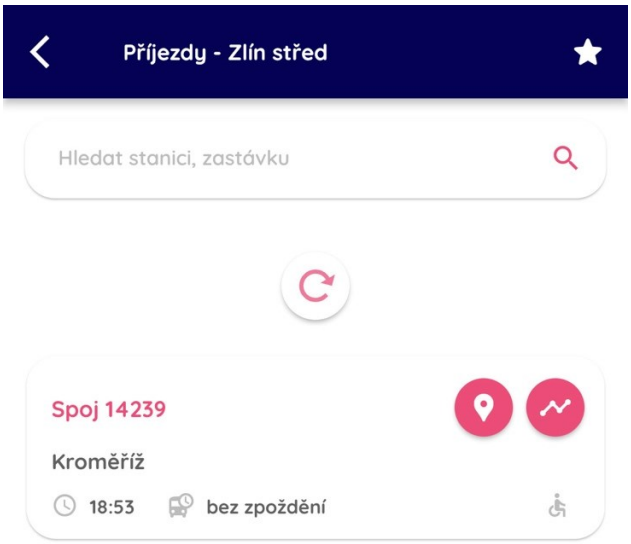
vyjelo a u odjezdových tabulí se naopak zobrazuje destinace, kam vozidlo míří. Hlavní informací u výsledků je číslo linky nebo spoje daného vozu, které je zde zobrazeno v kontrastní barvě oproti ostatním informacím. Další informace zde jsou zpoždění vozů a také čas odjezdu z vybrané stanice. Dále je u jednotlivých spojů také tlačítko s proklikem na jízdní řád spoje a linky, který již byl popsán v kapitole výše (tato komponenta je znovu použita). Tento jízdní řád také podporuje vizualizaci polohy, pokud je spoj aktivní. Mimo jízdní řád lze také zobrazit aktuální polohu vozidla, ale jen tehdy, když je aktivní a dostupné v datech. Pokud vozidlo dostupné není, tlačítko je zašedlé a neaktivní (Obrázek 39 - Výsledky vyhledávání).




Obrázek 39 - Výsledky vyhledávání

Když nastane nějaká chyba nebo je uživatel na obrazovce s výsledky delší dobu a výsledky již nejsou aktuální, zobrazí se dole po kliknutí na tlačítko polohy chybová hláška, která říká, že vozidlo již není aktivní. Chybová hláška se také zobrazí i v případě, když nastane nějaká chyba u jízdního řádu. Na této obrazovce není implementována automatická aktualizace dat, což znamená, že uživatel musí data manuálně obnovit, pokud je chce mít aktuální. To lze

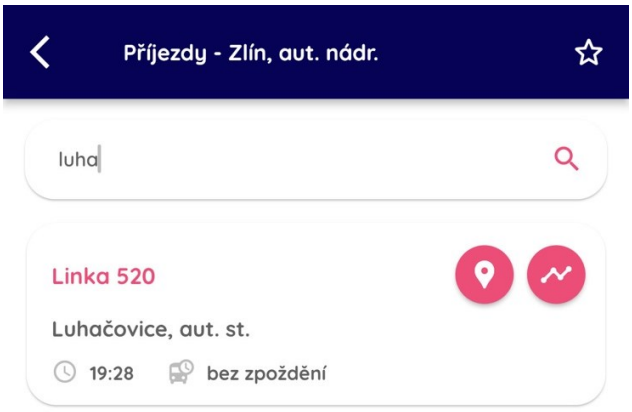
provést potažením z horní části obrazovky směrem dolů. Gesto zviditelní malý `loading_spinner`, který zmizí, jakmile se aktualizace dokončí (Obrázek 40 - Obnovení dat).




Obrázek 40 - Obnovení dat

Na obrazovce s výsledky se zobrazují všechna vozidla, která na zvolené stanici zastavují, mají ji na jízdním řádu jako destinaci nebo jako původní stanici. Je nutné zmínit, že výsledky vyhledávání jsou vždy na další zhruba 2 hodiny dopředu – i toto je bohužel ovlivněno daty od poskytovatele, v ideálním případě by si uživatel mohl také vybrat, v jaký čas se chce na odjezdy nebo příjezdy u stanice podívat.

Je zde také možnost filtrování podle názvu cílové nebo původní stanice. Uživatel má možnost zadat do vyhledávacího políčka název a díky tomu se potom aktualizují výsledky vyhledávání (Obrázek 41 - Filtrování výsledků). Toto vyhledávání se děje pouze na straně mobilního klienta, takže se nemusí čekat na odpověď ze serveru a děje se téměř okamžitě.



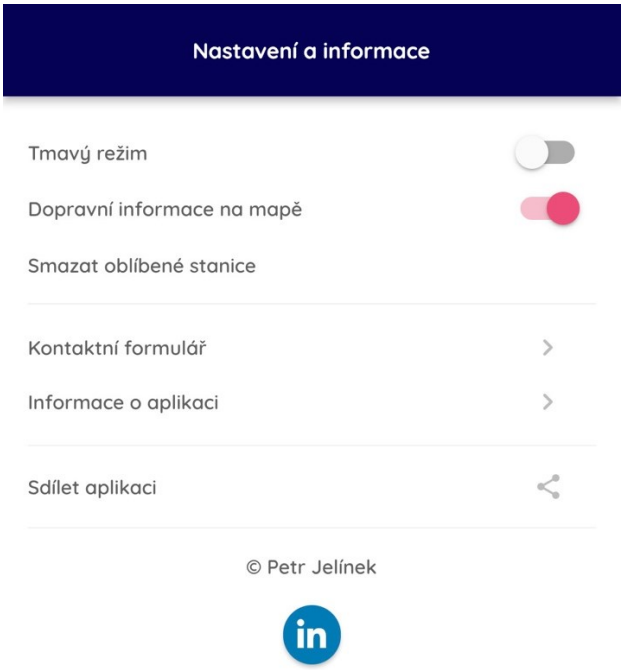

Obrázek 41 - Filtrování výsledků

## 7.4 Zobrazení polohy zastávky

Dříve bylo v aplikaci možné také zobrazit polohu hledané zastávky na mapě. Vzhledem k tomu, že tato data o poloze nejsou součástí dat, které se vrací ze strany poskytovatele dat, bylo nutné tato data brát z jiného zdroje. Jako zdroj bylo zvoleno API od *https://api.mapy.cz*, které umožnilo najít souřadnice hledaného bodu zájmu na mapě. V backend aplikaci byla tedy logika, která vzala název hledané zastávky a provedla dotaz na toto API, které vrátilo všechny body zájmu, které měly shodu se zvoleným názvem. Z odpovědi potom aplikace vybrala jen ty, které měly označení odpovídajícím zastávce nebo stanici. Problém zde však nastal v tom, že některá data neodpovídala zastávkám, které uživatel hledal (např. se vrátily informace o zastávce, která se nacházela u Prahy místo ve Zlínském kraji). Při testování bylo odhaleno takových nekonzistentních výsledků mnoho, proto bylo rozhodnuto tuto funkci dočasně vypnout.

# 8 NASTAVENÍ APLIKACE

Každá moderní aplikace umožní uživateli nějakou formu personalizace či nastavení, není tomu jinak ani v případě této aplikace a i zde si uživatel může nastavit několik věcí, jako jsou například viditelnost dopravních informací na mapě nebo barevný režim tak, aby vyhovovaly jeho osobním preferencím a aplikace se mu používala lépe a měl z jejího používání lepší pocit. Preference uživatele jsou uloženy v interní paměti aplikace a potažmo i zařízení, takže pokud aplikaci vypne a zapne ji znovu po nějakém čase, tato nastavení si aplikace bude pamatovat a uživatel je nemusí znovu nastavovat. Data nejsou uložena na serveru, protože uživatelé nemají v rámci aplikace žádný účet a nebylo by jednoduché je rozpoznat a přiřadit k nim jejich nastavení z databáze.




Obrázek 42 - Nastavení aplikace

## 8.1 Uživatelské preference

Pokud si uživatel nastaví nějakou proměnnou v nastavení aplikace (např. barevný režim aplikace), tato informace se uloží do paměti aplikace, potažmo i do trvalého úložiště zařízení, takže hodnota není jen v operační paměti a když uživatel aplikaci zapne po delší době, kdy ji nepoužíval, tato informace o jeho nastavení a preferenci bude v aplikaci stále známá. Tato funkce je možná díky knihovně, která byla do projektu přidána právě za účelem držení informací tohoto typu přímo na zařízení. Vzhledem k tomu, že aplikace nemá správu uživatelů (přihlášení, aktivní session, registrace atd.) a nekomunikuje v tomto ohledu s databází,

byla varianta ukládání informací do zařízení jako ta nejlepší možná volba. Knihovna, která podporuje práci s trvalým úložištěm, se jmenuje `shared_preferences` a je taktéž, jako ostatní knihovny, dostupná ke stažení z online repozitáře. U této knihovny není vyžadována další speciální konfigurace, stačí jen přidat knihovnu do konfiguračního souboru aplikace a poté knihovnu naimportovat. Tato knihovna pracuje s nativním perzistentním úložištěm na dané platformě (u iOS se jedná o `NSUserDefaults` a u Androidu to jsou `SharedPreferences`). Tato knihovna obaluje nativní úložiště pro jednoduchá data a přidává další funkcionalitu, aby se s tímto úložištěm dobře pracovalo. Data se do úložiště zapisují asynchronně, takže neposkytuje garanci toho, že se data zapíšou hned po zavolání funkce, tudíž se ani nedoporučuje zde ukládat kritická data. Data se do paměti ukládají v `key-value` formátu, to znamená, že každá data musí mít svůj specifický a ideálně unikátní klíč, jinak se data přepíší novou hodnotou [47]. Tato struktura je podobná například datové struktuře `HashMap`.

Tato funkce má za úkol inkrementovat počítadlo a uložit jeho stav do trvalé paměti. Při dalším kliknutí se vytvoří instance třídy `SharedPreferences`, na které jsou dostupné metody jako `getInt(key)`, `setInt(key, value)`, díky kterým lze z úložiště načítat nebo do něj ukládat vybraná data. Knihovna nabízí funkce také pro textové řetězce a další datové typy. Stejným způsobem aplikace ukládá uživatelské nastavení do paměti a při načítání se z paměti hodnoty čtou a používají dále pro konfiguraci a personalizaci funkcí (Obrázek 43 - Ukázka práce s knihovnou pro trvalé úložiště) [47].

```
_incrementCounter() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  int counter = (prefs.getInt('counter') ?? 0) + 1;
  print('Pressed $counter times.');
  await prefs.setInt('counter', counter);
}
```

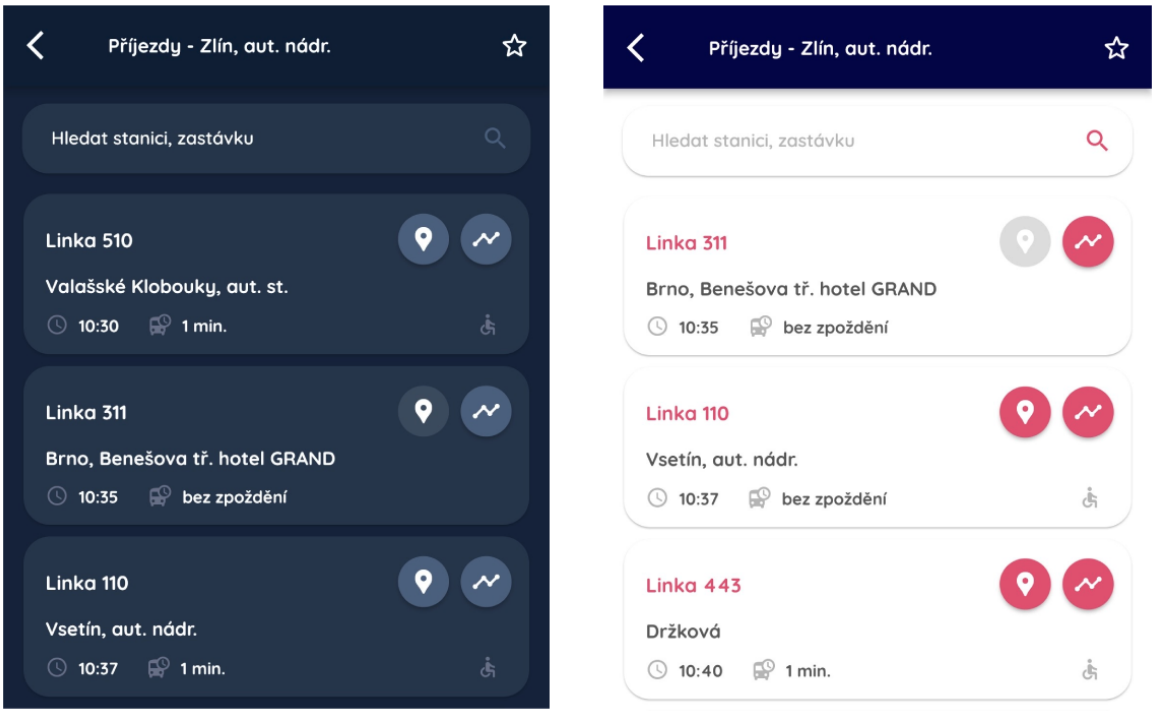
Obrázek 43 - Ukázka práce s knihovnou pro trvalé úložiště

## 8.2 Tmavý a světlý režim

První položkou v nastavení je barevný režim aplikace – uživatel si má možnost vybrat mezi tmavým a světlým režimem. Každý barevný režim má definované barevné schéma v konfigurační třídě v aplikaci, které je použito napříč celou aplikací – od barev textů, přes tlačítka až po samotnou mapu a ikony reprezentující vozidla (Obrázek 44 - Tmavý (vlevo) a světlý

(vpravo) režim aplikace). Barvy, potažmo celé barevné schéma, bylo pečlivě vybráno tak, aby obsahovalo příjemné barvy a aplikace nebyla nijak křiklavá anebo výrazná. Hodnota tohoto nastavení je uložena v perzistentním úložišti na zařízení, takže si aplikace pamatuje volbu uživatele a i po restartu aplikace je tato hodnota stále v paměti.

Z výzkumů mezi potencionálními uživateli vyplývá, že tmavý režim je obecně více oblíbený a češtěji používaný než světlý. Ve výchozím stavu je však aplikace nastavena na světlý režim a uživatel si musí tmavý režim zapnout sám, nicméně v dalších verzích by nastavení mělo být opačné a jako výchozí hodnota nastavení by měl být tmavý režim.



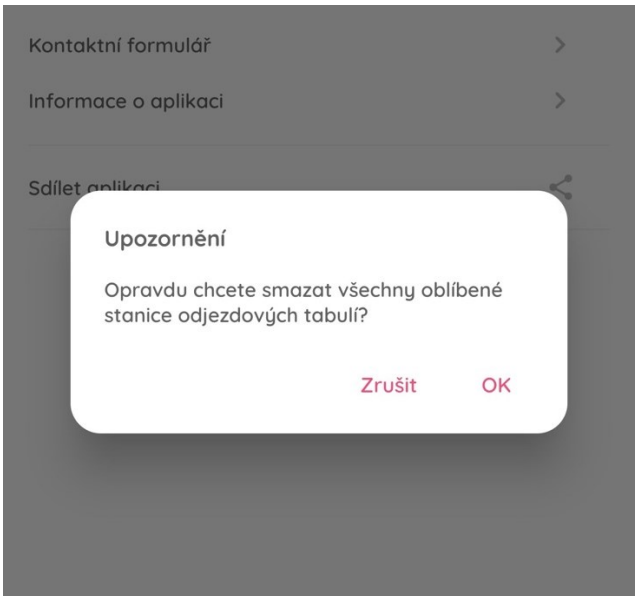

Obrázek 44 - Tmavý (vlevo) a světlý (vpravo) režim aplikace

## 8.3 Dopravní informace na mapě

Jak již bylo popsáno v kapitolách výše, aplikace nabízí na mapě aktuální dopravní informace o vytížení cest, uzavírkách a podobně. Tato data jsou dostupná přímo od Google v rámci knihovny, která implementuje funkce pro práci s Google Maps ve Flutter aplikaci. Pokud uživatel data o dopravě na mapě vidět nechce, stačí, aby tuto variantu v nastavení vypnul a jeho preference bude opět uložena v perzistentní paměti aplikace. Některým uživatelům se může zdát mapa i s těmito daty dost nepřehledná, takže pokud je vytížení dopravy na zvoleném místě nezajímá, je pro ně toto nastavení ideální (Obrázek 28 - Dopravní informace na mapě).

## 8.4 Mazání oblíbených stanic

Pokud by chtěl uživatel najednou odstranit všechny oblíbené stanice, které si v aplikaci uložil, má možnost tak učinit v nastavení. Po kliknutí na tuto možnost bude uživateli zobrazen uprostřed obrazovky widget `AlertDialog` (Obrázek 45 - Dialogové okno s varováním a potvrzením operace mazání), který ho varuje, že se chystá odstranit všechny uložené oblíbené stanice a táže se ho, zdali chce opravdu pokračovat. Uživatel má možnost se tedy ještě rozmyslet a pokud si je jistý, může svou volbu potvrdit, což zapříčiní následné smazání oblíbených stanic z trvalého úložiště zařízení.

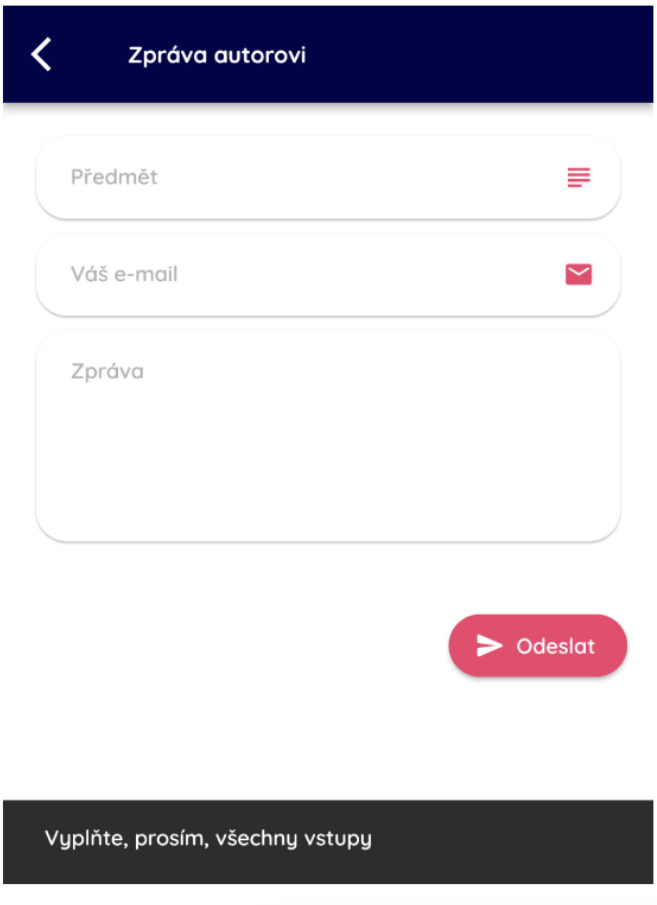



Obrázek 45 - Dialogové okno s varováním a potvrzením operace mazání

## 8.5 Zpětná vazba

Uživatelé mohou v nastavení přes aplikaci poslat také zpětnou vazbu autorovi aplikace, kde mohou napsat jakékoliv připomínky, nahlásit chybu nebo se zeptat na dotaz ohledně fungování nebo stavu aplikace. Po kliknutí v nastavení se otevře nová stránka s formulářem, který se po vyplnění odešle na server a odsud potom na email, kde může autor na podnět, připomínku nebo dotaz reagovat. Formulář obsahuje taktéž jednoduchou validaci, jako například to, že uživatel nezadal a neodeslal prázdný text zprávy či předmětu atd. Informace o validační chybě se zobrazí ve spodní části obrazovky (Obrázek 46 - Kontaktní formulář s validační chybou). O výsledku akce je uživatel informován pomocí `snackBar` widgetu, který se v dolní části zobrazí tehdy, když se vrátí ze serveru odpověď, ať už o úspěšném dokončení, anebo naopak o chybě, která při odesílání nastala.

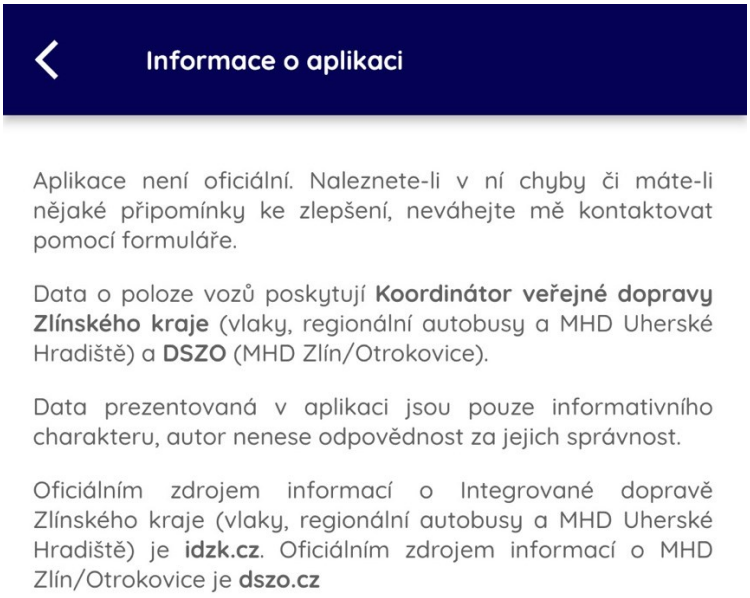
Na straně serveru je odesílání e-mailů na osobní e-mail autora implementováno tak, že po obdržení dat z mobilní aplikace (jedná se o standardní HTTP POST dotaz) a následné sestavení samotné zprávy, se backend připojí na SMTP (Simple Mail Transfer Protocol) server *smtp.seznam.cz* a poté e-mail odešle přes tento server. Toto řešení je implementováno popsaným způsobem převážně z důvodu ušetření nákladů za případný vlastní e-mail server a jednoduchosti řešení. Ze strany uživatele není nic poznat, takže tzv. „user experience" zůstává stále stejná v rámci aplikace a není nutné, aby uživatel chodil do jiné aplikace, odkud by e-mail s připomínkou mohl odeslat, což by mohlo spoustu lidí od odeslání hodnotné zpětné vazby odradit.




Obrázek 46 - Kontaktní formulář s validační chybou

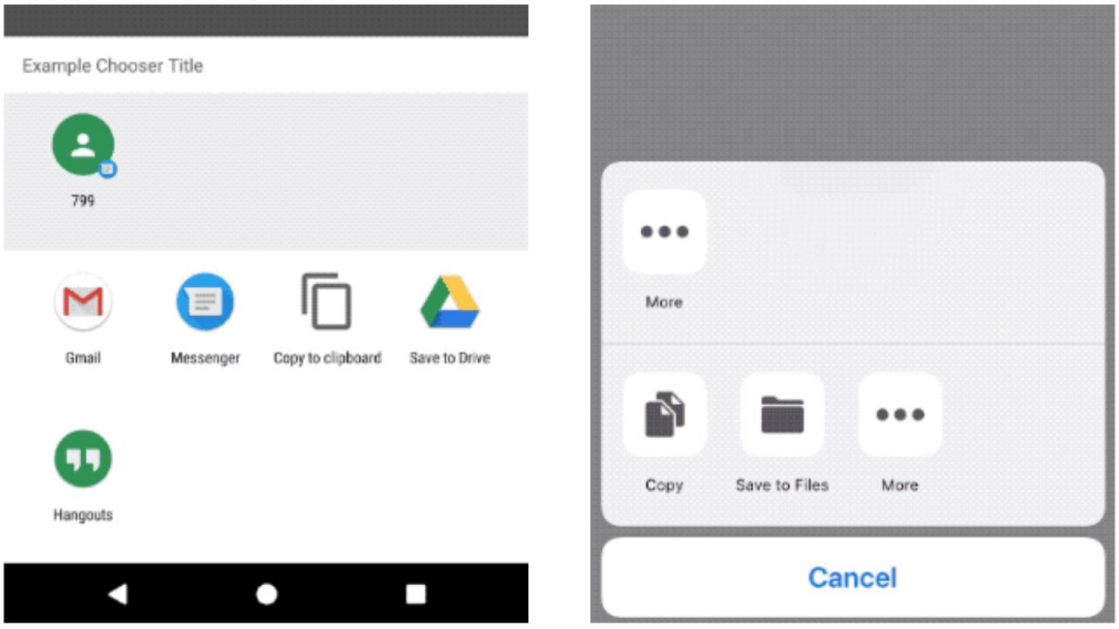
## 8.6 Informace o aplikaci, autorovi a sdílení aplikace

V neposlední řadě se na stránce s nastavením nachází také informace o aplikaci, informace o autorovi aplikace a poskytovatelích dat a také možnost sdílení aplikace dalším lidem. V informacích o aplikaci se nachází krátké shrnutí funkce a cíle aplikace, zmínění poskytovatelů dat a odkazy na jejich webové stránky a také zřeknutí se odpovědnosti za poskytovaná data (Obrázek 47 - Informace o aplikaci).




Obrázek 47 - Informace o aplikaci

Sdílení aplikace je vytvořeno za pomocí knihovny `share`, která podobně jako knihovna pro perzistentní úložiště obaluje nativní dialog pro sdílení (u Androidu to je `ACTION_SEND` a u iOS `UIActivityViewController`) a poskytuje k nim další funkcionalitu a podporu pro Flutter. Proto na zařízení s operačním systémem iOS bude toto okno pro sílení vypadat jinak než u Androidu (Obrázek 48 - Sdílení na Androidu (vlevo) a na iOS (vpravo)).




Obrázek 48 - Sdílení na Androidu (vlevo) a na iOS (vpravo) [48]

# 9 TESTOVÁNÍ FUNKCIONALITY APLIKACE

Aby byla aplikace pokud možno co nejplynulejší a nejlepší pro používání, při vývoji podléhala také velkému množství testování, ověřování veškeré funkcionality a reagování na podněty potencionálních uživatelů aplikace při uživatelském a tzv. A/B testování.

Před samotným vývojem mobilní aplikace proběhlo designování a návrh vzhledu aplikace a uživatelského rozhraní. Tento návrh byl vytvořen a postupně laděn v grafickém editoru Figma, který je zdarma dostupný ať už jako webová nebo desktopová verze. Pro každou novou funkci byl nejprve vytvořen návrh, který byl otestován na potencionálních uživatelích a po úspěšném vyhodnocení následně také implementován do aplikace. Pokud měli potencionální uživatelé (testeři) aplikace nějaké připomínky ať už k samotnému vzhledu či jednoduchosti používání, byly tyto podněty brány v potaz a následně reflektovány do vzhledu aplikace. Jeden z největších cílů aplikace je totiž právě vysoká přehlednost a jednoduchost používání, aby uživatelé i starších věkových skupin neměli problém se v aplikaci zorientovat.

## 9.1 Testování a ověřování během vývoje

Po vyvinutí a během vývoje dané funkcionality byla vždy dílčí část aplikace vývojářem řádně otestována i zkontrolována tak, aby vyvinutá funkcionalita splnila požadavky a ověřilo se, že je všechno v pořádku, jak z hlediska prezentovaných dat, tak z pohledu prezentační logiky a zobrazení všech komponent a widgetů v aplikaci.

Ověřování funkcionality probíhalo hned několika způsoby – jeden z nich bylo klasické manuální testování, což znamená, že při vývoji byla daná funkce (např. zobrazení vozidla na mapě) testována přímo na simulátoru zařízení. Díky funkci Hot Reload, kterou nabízí Flutter, bylo toto testování značně zjednodušeno – po uložení upraveného zdrojového kódu se změny totiž hned projevily na simulátoru a bylo tak možné ověřit, zda aplikace dělá to, co je očekáváno, bez nutnosti restartu nebo čekání na sestavení aplikace. Tímto způsobem se dala testovat například prezentační logika (vykreslení značek na mapě, správné zobrazení a rozložení widgetů…) a také vrácená data od poskytovatele a potažmo i z vlastní API atd.

Další testování probíhalo vůči reálnému světu, což znamená, že sestavená aplikace byla nainstalována na testovací zařízení a s ním byl potom sledován reálný provoz na nádraží a v terénu (například poloha vlaku), čímž se ověřovalo, že informace o zpoždění a poloha vozu odpovídá realitě.

Tím, že aplikace pracuje s real-time daty, je v aplikaci také mnoho tzv. edge cases (okrajový případ), což znamená, že k nim může dojít pouze při extrémních provozních parametrech – například bylo nutné ošetřit situaci, kdy vozidlo vjelo do místa, kde není GPS signál apod. V aplikaci bylo potřeba tyto okrajové případy co nejvíce podchytit a ošetřit je tak, aby se zachoval stejný uživatelský dojem z používání aplikace a v ideálním případě uživatel ani chybu nepoznal. Na tyto okrajové případy se došlo právě díky testování, nejvíce jich však bylo odhaleno hned ze začátku vývoje, ještě před samotným uživatelským testování aplikace. Obecně bylo toto testování velmi užitečné k odhalení chyb a jejich opravě ještě před tím, než by na chybu mohli narazit uživatelé.

V neposlední řadě bylo při vývoji testováno také zobrazení na různých zařízeních podle velikosti, výkonu atd. Aplikace je podporována jak pro Android, tak iOS, což znamená, že je možné si ji nainstalovat na tablet, mobilní telefon a dokonce i počítač. Proto bylo nutné, aby se na různých velikostech obrazovky veškeré komponenty zobrazovaly správně, aby se komponenty a widgety přes sebe nijak nepřekrývaly a uživatel tak nepřišel o důležité informace, které by se díky špatnému rozložení widgetů nezobrazily. Toto testování probíhalo jak na reálných zařízeních, tak na simulátorech, protože ne všechna zařízení (velikosti) byla dostupná ve fyzické formě.

## 9.2 Automatizované testování a pipeline

Automatizované testování aplikace je v tuto chvíli pouze na straně serveru, tedy backend část. Automatizované testování mobilní aplikace by bylo samozřejmě také realizovatelné, například přímo pomocí testovacího frameworku ve Flutteru, díky čemuž by se daly psát unit testy (jednotkové testy), integrační testy nebo například tzv. widget testy, jejichž cílem je otestovat, že daný widget vypadá a prezentuje data tak, jak je specifikováno [49].

Backend aplikace obsahuje testy (nejsou zahrnuty v produkčním kódu), které ověřují funkčnost a správné chování na straně serveru. Největší část testů tvoří unit testy, které ověřují funkcionalitu vždy pouze jedné části kódu (např. jedna metoda). Pokud se v této metodě volají další metody (případně metody v jiné službě nebo volání API poskytovatele dat), je lepší toto volání metod tzv. mockovat, což znamená to, že se při volání této metody vrátí předem určený výsledek a reálná metoda se nezavolá. V aplikaci je testována vždy happy a sad path (například metoda sčítání: happy path – 2 plus 2 rovná se 4, sad path – 2 plus 3 nerovná se 4). Pokud se metoda dále větví, například pomocí podmínek, je také nutné otestovat všechny možné výsledky a větve metody. Tyto testovací metody jsou vždy

idempotentní, což znamená, že při stejném vstupu musí vždy vrátit stejný výstup. Další část testů jsou integrační testy, které ověřují fungování všech komponent integrovaných dohromady (controller zavolá service, ta volá databázi atd.) a následně se otestuje očekávaný výsledek. V aplikaci jsou tyto typy testů využity primárně pro controllery.

Testy by se daly pustit mimo lokální prostředí také například v automatizované „pipeline" (CI – continuous integration) na serveru (GitHub, GitLab, Bitbucket…), která by se spustila například ve chvíli, kdy by do repozitáře přibyl nový commit se změnami v kódu nebo se vytvořil merge request. Díky tomu by se potom dalo ověřit, že napsané testy prochází a pokrytá část aplikace těmito testy je v pořádku i kdyby vývojář testy nepustil manuálně na lokálním prostředí. Definice samotné pipeline se ve většině případů píše do konfiguračního `YAML` souboru pomocí klíčových slov pro danou platformu, který je potom součástí repozitáře. Například na GitLabu se potom díky tomuto souboru (`gitlab-ci.yml`) automaticky vytvoří definice popsané pipeline. V tomto souboru se tedy nastavuje, jaké fáze (stage) a úkoly (job) pipeline bude mít – první stage může například aplikace sestavit, druhá stage ji otestovat a třetí třeba vizualizovat pokrytí kódu testy na základě vygenerovaných reportů. Tento proces je čistě na vývojáři a na tom, jaké věci chce automatizovat. V rámci pipeline (CD – continuous delivery), lze automatizovat třeba i samotné nasazení aplikace do cloudu, což také může ušetřit spoustu času, pokud má třeba aplikace více prostředí. V rámci automatizovaných pipeline jde dělat mnoho užitečných věcí, které ve výsledku velmi zjednoduší a také zefektivní celý proces vývoje [50].

## 9.3 Uživatelské testování

Aplikace byla během vývoje a následně také po něm podrobena testování nejen vývojářem, ale také testery a potenciálními uživateli aplikace, tedy dalšími lidmi, kteří mohou mít na danou věc jiný pohled než pohled jednoho vývojáře, a obohatit tak svými poznatky výsledný produkt k lepšímu. Dále také proto, aby byla aplikace, pokud možno co nejvíce odladěná a pro uživatele intuitivní na používání. Uživatelské testování se dělilo na několik typů a částí.

### 9.3.1 Observační testování

Toto testování mělo za cíl zjistit, jak se aplikace uživatelům reálně používá. Observační testování probíhalo tak, že na testovací zařízení s operačním systémem Android byla nainstalována sestavená verze aplikace a uživatelům byl zadán jakýsi úkol (v rámci aplikace úkol vypadal například jako najít na mapě vlakové nebo autobusové spojení s určitým číslem,

zobrazit jízdní řád a určit polohu vozidla nebo třeba vyhledat stanici a podívat se, jaká vozidla na ni přijedou či naopak z ní odjedou). Uživatelé byli při tomto úkolu pečlivě sledováni a kontrolováni, jakým způsobem aplikaci za účelem splnění zadané úlohy používají a jestli nemají například problém najít nějakou funkci, jestli jsou pro ně tlačítka a text v ideální velikosti nebo také jaký mají celkový dojem z aplikace a jejího vzhledu.

Následně byly výsledky tohoto testování vyhodnoceny a na jejich základě, pokud byly negativní či měli uživatelé připomínky k nějaké funkci, případně vzhledu, byla daná část v aplikaci předělána tak, aby pokud možno co nejvíce uživatelům vyhovovala, měli z ní dobrý dojem a lépe se jim používala. Jednou z věcí, která byla díky uživatelskému a observačnímu testování změněna, bylo chování klávesnice u vstupních polí – když uživatel zadal text a chtěl klávesnici skrýt kliknutím na okolní prostor, klávesnice se neskryla. Na toto chování byli uživatelé zvyklí z jiných aplikací a byli překvapení, že to zde provést nelze. Proto tedy na základě tohoto podnětu bylo chování klávesnice pozměněno tak, že pokud je klávesnice na obrazovce aktivní a uživatel klikne na jakoukoliv část obrazovky v aplikaci, klávesnice se skryje a ze vstupního pole zmizí fokus.

### 9.3.2 A/B Testování

Další část uživatelského testování probíhala formou A/B testování, což je metoda, která se často používá například u marketingových kampaní, reklam nebo designu. Tato metoda spočívala v tom, že byly vytvořeny dvě (nebo i více) verze daného produktu (v tomto případě například dvě verze barevných schémat, velikostí tlačítek a fontů, různé rozmístění ovládacích prvků apod.) [51], které byly následně distribuovány mezi skupinu uživatelů (testerů) a v těchto skupinách se sledovalo, která verze si ve výsledku vedla lépe, ať už přímo z uživatelských ohlasů, na základě pozorování uživatelského chování při používání aplikace nebo plnění zadaného úkolu, který měli v rámci aplikace splnit. Díky tomuto testování bylo upraveno rozmístění a vzhled tlačítek na informačních kartách na mapě nebo také u výsledků vyhledávání u odjezdových a příjezdových tabulí.

# ZÁVĚR

Aplikace nabízí uživatelům možnost sledování polohy vozů veřejné dopravy ve Zlínském kraji, mohou si zde vybrat, zdali chtějí sledovat vozy linek MHD Zlín nebo vozy regionálních linek, včetně MHD Uherské Hradiště nebo chtějí mít na mapě zobrazeny všechny dostupné vozy. U vybraného detailu vozu jsou zobrazeny informace o samotném vozidle (technické vybavení vozu jako je klimatizace, obrazovka, bezbariérový přístup atd.), jeho přesné zpoždění, cílová a poslední navštívená stanice a jízdní řád vybrané linky, který vizualizuje aktuální polohu mezi zastávkami. Uživatel díky tomu může vidět, kolik zastávek ještě zbývá do jeho destinace. Aplikace mimo jiné nabízí i filtrování a hledání vozů přímo na mapě na základě linky a typu vozidla. Díky možnosti personalizace aplikace si uživatel může vybrat, jestli chce jako mapový podklad leteckou či výchozí mapu, dále také možnost zobrazení dopravních informací od společnosti Google nebo jinou barvu podkladů na základě barevného režimu aplikace.

Mobilní aplikace obsahuje funkci virtuálních odjezdových/příjezdových tabulí, ve kterých si mohou uživatelé zvolit, zda chtějí zobrazit odjezdy či příjezdy pro zvolenou zastávku/stanici. Ve výsledcích se promítají informace o vozidle, jeho přesné zpoždění, jízdní řád a pokud je vozidlo aktivní a dostupné na mapě, tak si mohou zobrazit jeho aktuální polohu. Uživatelé mohou ve výsledcích hledat a filtrovat vozidla podle jejich destinace/původní stanice, což jim umožní rychlejší nalezení tíženého spoje. U virtuálních tabulí si uživatelé mohou také uložit do oblíbených výsledky stanic, které vyhledávají častěji, takže je nemusí při každém hledání psát znovu.

V aplikaci si uživatel může vybrat ze dvou barevných režimů – tmavý a světlý, které jsou spolu s dalšími preferencemi uloženy v paměti zařízení a aplikace si tak pamatuje, jak ji má uživatel nastavenou. Celá aplikace je tvořena v Material Design stylu, který nabídne uživatelům známé a povědomé prostředí z jiných aplikací, převážně od firmy Google, což ji dělá pro uživatele více intuitivní a jednodušší pro používání (například pro uživatele starší věkové kategorie).

V dalších verzích aplikace je možné velké množství dalších rozšíření a aktualizací jako je například webová verze celé aplikace či lepší podpora MHD Zlín, a to konkrétně přidání zastávek do virtuálních odjezdových a příjezdových tabulí. Dále také nastavení výchozího barevného režimu na tmavý a mnoho dalšího.

# SEZNAM POUŽITÉ LITERATURY

[1] JELÍNEK, Petr. Aplikace ve Flutteru. *Medium* [online]. 2020, 24.6.2020 [cit. 2022-01-12]. Dostupné z: https://medium.com/@tenpetr/aplikace-ve-flutteru-6c308a35cac0

[2] OBINNA, Onuoha. How does JIT and AOT work in Dart?. *Medium* [online]. 2020, 7.4.2020 [cit. 2022-01-12]. Dostupné z: https://learnlanga.medium.com/how-does-jit-and-aot-work-in-dart-cab2f31d9cb5

[3] GILDER, Tom. Flutter's Key Difference: Owning Every Pixel. *Medium* [online]. 2019, 11.3.2019 [cit. 2022-01-12]. Dostupné z: https://medium.com/flutter-community/flutters-key-difference-owning-every-pixel-e2135b44c8a

[4] Material Components widgets. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://docs.flutter.dev/development/ui/widgets/material

[5] Cupertino (iOS-style) widgets. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://docs.flutter.dev/development/ui/widgets/cupertino

[6] StatelessWidget class. *Flutter - Dart API docs* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

[7] StatefulWidget class. *Flutter - Dart API docs* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

[8] Hot reload. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://docs.flutter.dev/development/tools/hot-reload

[9] Visual Studio Code. *Visual Studio Code - Code, Editing. Redefined* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://code.visualstudio.com

[10] Android Studio. *Android Developers* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://developer.android.com/studio

[11] Xcode 13. *Xcode 13 Overview - Apple Developer* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://developer.apple.com/xcode

[12] Pub.dev. *Dart packages* [online]. 2022 [cit. 2022-01-12]. Dostupné z: https://pub.dev

[13] What is Java technology and why do I need it?. *Java | Oracle* [online]. Oracle [cit. 2022-01-12]. Dostupné z: https://www.java.com/en/download/help/whatis_java.html

[14] Differences between Java EE and Java SE. *Oracle Help Center* [online]. Oracle, 2012 [cit. 2022-01-12]. Dostupné z: https://docs.oracle.com/javaee/6/first-cup/doc/gkhoy.html

[15] JEMEROV, Dmitry a Svetlana ISAKOVA. Kotlin in action. Shelter Island, NY: Manning Publications Co., [2017]. ISBN 9781617293290.

[16] Null safety. *Kotlin Programming Language* [online]. JetBrains, 2021, 8.7.2021 [cit. 2022-01-12]. Dostupné z: https://kotlinlang.org/docs/null-safety.html

[17] Extensions. *Kotlin Programming Language* [online]. JetBrains, 2021, 19.7.2021 [cit. 2022-01-12]. Dostupné z: https://kotlinlang.org/docs/extensions.html

[18] Data classes. *Kotlin Programming Language* [online]. JetBrains, 2021, 12.7.2021 [cit. 2022-01-12]. Dostupné z: https://kotlinlang.org/docs/data-classes.html

[19] A Comparison Between Spring and Spring Boot. *Baeldung* [online]. Baeldung, 2021, 24.3.2021 [cit. 2022-01-12]. Dostupné z: https://www.baeldung.com/spring-vs-spring-boot

[20] WALLS, Craig. Spring Boot in action. Shelter Island, NY: Manning Publications, [2016]. ISBN 9781617292545.

[21] Spring Initializr. *Spring Initializr* [online]. VMware, 2022 [cit. 2022-01-12]. Dostupné z: https://start.spring.io

[22] What is the cloud? | Cloud definition. *Cloudflare* [online]. Cloudflare, 2022 [cit. 2022-01-12]. Dostupné z: https://www.cloudflare.com/learning/cloud/what-is-the-cloud

[23] How Heroku Works. *Heroku Dev Center* [online]. Salesforce.com, 2020, 9.6.2020 [cit. 2022-01-12]. Dostupné z: https://devcenter.heroku.com/articles/how-heroku-works

[24] Build apps for free on Heroku. *Cloud Application Platform | Heroku* [online]. Salesforce.com, 2022 [cit. 2022-01-12]. Dostupné z: https://www.heroku.com/free

[25] Heroku Pricing. *Cloud Application Platform | Heroku* [online]. Salesforce.com, 2022 [cit. 2022-01-12]. Dostupné z: https://www.heroku.com/pricing

[26] Buildpacks Go Cloud Native. *Heroku Blog | Heroku* [online]. Salesforce.com, 2018, 3.10.2018 [cit. 2022-01-12]. Dostupné z: https://blog.heroku.com/buildpacks-go-cloud-native

[27] IBM CLOUD EDUCATION. Containerization. *IBM* [online]. IBM, 2021, 23.6.2021 [cit. 2022-01-15]. Dostupné z: https://www.ibm.com/cloud/learn/containerization

[28] Docker vs Virtual Machines (VMs) : A Practical Guide to Docker Containers and VMs. *Operate and Manage Kubernetes easily with Weaveworks* [online]. WEAVEWORKS, 2020, 16.1.2020 [cit. 2022-01-15]. Dostupné z: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms

[29] IBM CLOUD EDUCATION. Docker. *IBM* [online]. IBM, 2021, 23.6.2021 [cit. 2022-01-15]. Dostupné z: https://www.ibm.com/cloud/learn/docker

[30] Sample application. *Docker Documentation* [online]. Docker, 2021 [cit. 2022-01-15]. Dostupné z: https://docs.docker.com/get-started/02_our_app

[31] Kubernetes Documentation. *Kubernetes Documentation | Kubernetes* [online]. The Kubernetes Authors, 2021, 20.7.2021 [cit. 2022-01-15]. Dostupné z: https://kubernetes.io/docs/home

[32] About. *Git* [online]. [cit. 2022-01-15]. Dostupné z: https://git-scm.com

[33] PRESTON-WERNER, Tom. Semantic Versioning 2.0.0. *Semantic Versioning 2.0.0* [online]. [cit. 2022-01-15]. Dostupné z: https://semver.org

[34] MARTIN, Robert C. Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall, c2009. ISBN 9780132350884.

[35] MARTIN, Robert C. Clean architecture: a craftsman's guide to software structure and design. London, England: Prentice Hall, [2018]. ISBN 0134494164.

[36] FOWLER, Martin. Refaktoring: zlepšení existujícího kódu. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0299-1.

[37] Build and release an Android app. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-19]. Dostupné z: https://docs.flutter.dev/deployment/android

[38] Build and release an iOS app. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-19]. Dostupné z: https://docs.flutter.dev/deployment/ios

[39] Adaptive icons. *Android Developers* [online]. 2022 [cit. 2022-01-19]. Dostupné z: https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

[40] About Android App Bundles. *Android Developers* [online]. 2022 [cit. 2022-01-19]. Dostupné z: https://developer.android.com/guide/app-bundle

[41] MORGAN, Brett. Adding Google Maps to a Flutter app. *Google Codelabs* [online]. 2021, 3.9.2021 [cit. 2022-01-22]. Dostupné z: https://codelabs.developers.google.com/codelabs/google-maps-in-flutter#0

[42] Maps SDK for Android Usage and Billing. *Google Developers* [online]. 2022, 21.1.2022 [cit. 2022-01-22]. Dostupné z: https://developers.google.com/maps/documentation/android-sdk/usage-and-billing

[43] Getting started with Google Maps Platform. *Google Developers* [online]. 2022, 21.1.2022 [cit. 2022-01-22]. Dostupné z: https://developers.google.com/maps/gmp-get-started#create-project

[44] SDK Platform release notes. *Android Developers* [online]. 2022 [cit. 2022-01-22]. Dostupné z: https://developer.android.com/studio/releases/platforms

[45] Google_maps_flutter 2.1.1. *Pub.dev* [online]. 2022 [cit. 2022-01-22]. Dostupné z: https://pub.dev/packages/google_maps_flutter

[46] KUMAR, Abhishek. How to create custom widget based Info Window for Google Maps in Flutter?. *CODING NOTES* [online]. 2020, 6.6.2020 [cit. 2022-01-22]. Dostupné z: https://www.abhishekduhoon.com/2020/06/how-to-create-widget-based-google-maps.html

[47] Shared_preferences 2.0.12. *Pub.dev* [online]. 2022 [cit. 2022-01-29]. Dostupné z: https://pub.dev/packages/shared_preferences

[48] Flutter_share 2.0.0. *Pub.dev* [online]. 2022 [cit. 2022-01-29]. Dostupné z: https://pub.dev/packages/flutter_share

[49] Testing Flutter apps. *Flutter documentation | Flutter* [online]. 2022 [cit. 2022-01-29]. Dostupné z: https://docs.flutter.dev/testing

[50] GitLab CI/CD. *GitLab Documentation* [online]. [cit. 2022-01-29]. Dostupné z: https://docs.gitlab.com/ee/ci

[51] A/B Testing Guide. *VWO | #1 A/B Testing Tool in the World* [online]. Wingify, 2022 [cit. 2022-01-29]. Dostupné z: https://vwo.com/ab-testing

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| AOT | Ahead-Of-Time |
| API | Application Programming Interface |
| APK | Android Application Package |
| AWS | Amazon Web Services |
| CD | Continuous Delivery |
| CI | Continuous Integration |
| CPU | Central Processing Unit |
| CSS | Cascading Style Sheets |
| DTO | Data Transfer Object |
| GCP | Google Cloud Platform |
| GPS | Global Positioning System |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ID | Identifier |
| IDE | Integrated Development Environment |
| JDBC | Java Database Connectivity |
| JDK | Java Development Kit |
| JIT | Just-In-Time |
| JRE | Java Runtime Environment |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| KISS | Keep It Simple, Stupid! |
| LTS | Long-Term Support |
| MHD | Městská Hromadná Doprava |

| | |
|---|---|
| MVC | Model View Controller |
| NPE | Null Pointer Exception |
| ORM | Object-Relational Mapping |
| RAM | Random Access Memory |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| SMTP | Simple Mail Transfer Protocol |
| VM | Virtual Machine |
| VS | Visual Studio |

# SEZNAM OBRÁZKŮ

# SEZNAM PŘÍLOH