UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Orest Khanenya

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Návrh a implementace aplikace pro lokalizaci kolejového vozidla s využití mini počítače Raspberry Pi a webového servisu

Orest Khanenya

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Orest Khanenya**
Osobní číslo:	**I13272**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a implementace aplikace pro lokalizaci kolejového vozidla s využití mini počítače Raspberry Pi a webového servisu**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je návrh a implementaci aplikace pro lokalizaci kolejového vozidla s využití mini počítače Raspberry Pi a webového servisu.
V teoretické části práce bude popsán význam lokalizace kolejových vozidle a proveden přehled různých druhů lokalizace se zaměřením na železnice
V praktické části se student zaměří na vlastní návrh a implementaci aplikace, která bude umožňovat vizualizovat polohu a doplňkové informace o poloze kolejového vozidla na železniční síti pomocí mini počítače Raspberry Pi a již existujícího webového servisu
Výsledná aplikace bude otestována v reálném provozu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

UPTON, Eben a Gareth HALFACREE. Raspberry Pi: uživatelská příručka. Brno: Computer Press, 2013. ISBN 978-80-251-4116-8.

MONK, Simon. Programming the Raspberry Pi,: McGraw-Hill Education TAB, 2015. ISBN 978-1259587405

Vedoucí bakalářské práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 29.04.2017

Orest Khanenya

PODĚKOVÁNÍ

Chtěl bych poděkovat svému vedoucímu práce, Ing. Janu Fikejzovi, Ph.D., za cenné rady a odborný dohled, přátelům a rodičům za podporu po celou dobu studia.

ANOTACE

Cílem práce je návrh a implementace aplikace, která umožňují zobrazení aktuální polohy a doplňkové informace kolejového vozidla na železniční síti s využitím již existujícího webového servisu a jednodeskového počítače Raspberry Pi. Aplikace bude běžet pod operačním systémem Raspbian a implementována ve programovacím jazyce Java.

KLÍČOVÁ SLOVA

Raspberry Pi, JavaFX, Java, Webové služby

TITLE

Design and implementation of an application for localization of rolling stock using Raspberry Pi and a web service.

ANNOTATION

The aim of this work is design and implementation of application, targeted to display current location and complementary information about rolling stock on the rail network by using single board computer Raspberry Pi and existing web service. Application will be running for operation system Raspbian and implemented in programming language Java.

KEYWORDS

Raspberry Pi, JavaFX, Java, Web services

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
ARM	Advanced RISC Machine
CPU	Central Processing Unit
CSS	Cascading Style Sheets
GLONASS	Global Navigation Satellite System
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
LXDE	Lightweight X11 Desktop Environment
MVC	Model–View–Controller
NMEA	National Marine Electronics Association
NOOBS	New Out Of Box Software
PIXEL	Pi Improved Xwindows Environment, Lightweight
SDRAM	Synchronous Dynamic Random Access Memory
UPCE	Univerzita Pardubice
USB	Universal Serial Bus
XML	Extensible Markup Language

ÚVOD

Využívání informačních technologií se stalo nedílnou součástí našeho každodenního života. Nejinak tomu je i z pohledu využívání satelitní navigace. Ta nachází široké uplatní zejména v dopravě. Díky vzniku a pokračování v rozvoji globálního družicového polohového systému (GNSS) existuje možnost sledování polohy pozemních, vodních a vzdušných objektů (vlaků, lodí, letadel atd.). Mimo jiné satelitní polohové systémy mají možnost získávat i další doplňkové informace, jako je například rychlost a směr pohybu. V současné době jsou aktivně využívané družicové polohové systémy jako GPS, GLONASS a Galileo.

Kromě údajů o poloze často uživatele zajímají i doplňující informace spojené s dopravou. V situacích souvisících s jízdou vlakem takovou informací může být například seznam železničních zastávek, počet vozů, detailní informace o poskytovaných službách a další podobné údaje. V současnosti jsou často používané počítačové databáze, které umožňují uchovávat a zpracovávat takové velké množství dat. Pro přenos dat se často využívá síť Internet, kde jeho konečným cílem je buď webová stránka nebo zvláštní zpracovávaná aplikace.

Bakalářská práce se primárně zaměřuje na návrh a implementace aplikace pro zobrazení informace a aktuální polohy kolejového vozidla na železniční síti, s použitím přenosného terminálu postaveného na základě mikropočítače, ve kterém pro získání aktuální polohy použít vnější GPS přijímač a dotykový displej sloužící k zobrazení informací.

Teoretická část práce popisuje již existující webový servis, se kterým úzce spolupracuje navržená aplikace, hardwarové komponenty, technologie které byly použity, strukturu projektu a popis implementace vlastní aplikace.

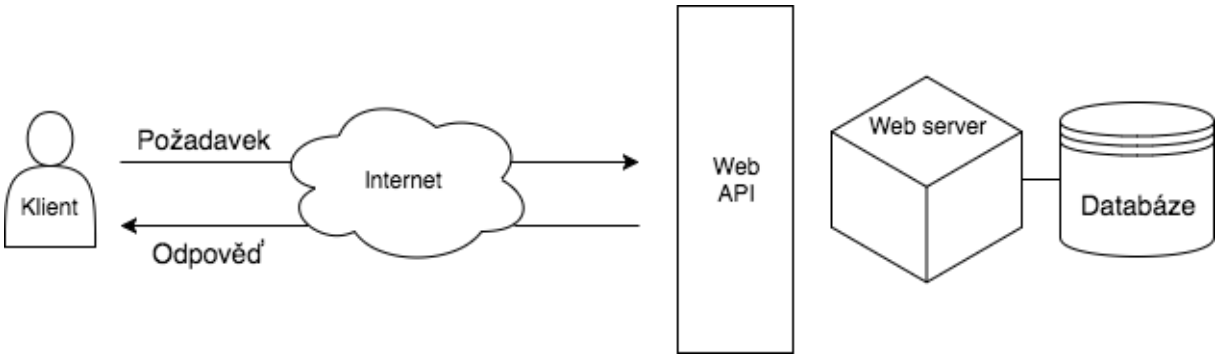
V praktické části bakalářské práce byla nejprve provedena analýza technického zadání, následně pak byl vytvořen pracovní plán, a nakonec návrh a implementace aplikaci pro její další použití na mikropočítači.

1 WEBOVÉ APLAKAČNÍ ROZHRANÍ

Tato kapitola obsahuje popis aplikačního rozhraní, které bylo použité ve vytvořené aplikace, princip jejích využití. Dále je popsán textový formát JSON, který je využit pro přenos informace v rámci webového aplikačního rozhraní (dále jen Web API).

1.1 Popis existujícího aplikačního rozhraní

Navržená aplikace funguje na principu klient-server. Pro získání a zobrazení požadované informaci, aplikace využívá již existující webové rozhraní, které vzniklo v rámci Univerzity Pardubice. Toto webové API tvoří prostředníka mezi klientskou aplikací a již navrženým na univerzitě datovým modelem [1]. Datový model představuje model železniční sítě pro území Pardubického a Královehradeckého kraje, pomocí kterého lze provádět simulaci provozu kolejových vozidel. Implementace modelu byla provedena pomocí Oracle Spatial s nadstavbou Network Data Model, která v tomto případě vytváří dvě aplikační vrstvy, kde první vrstva je Web API a druhá je klientská aplikace [2].



Obrázek 1 – Klient-server komunikace

Pro získání potřebné informací je nutné požádat Web API, odesláním HTTP požadavku. Jak již bylo zmíněno výše, Web API slouží prostředníkem mezí aplikací a modelem, proto po obdržení požadavku od klienta (aplikace) a jeho zpracování, klient obdrží odpověď od služby, která obsahuje požadované informace nebo potvrzení o úspěšném zpracování požadavku (záleží na typu dotazu).

Při vývoji aplikaci aktivně se používá tři možné dotazy:

1. {endpoint}/trains/webapi/train?number={number}
   Rozhraní vrací podrobnou informaci ohledně vlaku podle jeho čísla. Požadované parametry: číslo vlaku.
2. {endpoint}/trains/webapi/train?longitude={longitude}&latitude={latitude}&azimuth={azimuth}

Rozhraní vrací informaci ohledně vlaku podle jeho polohy. Požadované parametry:

- longitude - zeměpisná délka,
- latitude - zeměpisná šířka,
- azimuth - azimut.

3. {endpoint}/trains/webapi/train/path?stationFromID={stationFromID}&stationToID={stationToID}

   Rozhraní vrací souřadnice přilehlé k cestě mezi dvěma stanicemi. Požadované parametry:

   - stationFromID - identifikační číslo výchozí stanice,
   - stationToID - identifikační číslo cílové stanice.

Poznámka: endpoint = http://fei-hosting.upceucebny.cz:8080.

1.2 JSON

Při prací s Web API se obvykle používá určitá sada HTTP požadavků (příklad který byl uveden výše) stejně jako struktury HTTP odpovědí. Pro odpovědí se používá strukturovaný formát XML(Extensible Markup Language) nebo formát JSON(JavaScript Object Notation) . Web API, které se používá v rámci aplikace, vrací všechny odpovědi ve formátu JSON.

JSON je univerzální textový formát určený pro přenos dat, který je založen na zápisu objektových literálů v jazyce JavaScript. Vzhledem ke své stručnosti v porovnání s XML, formát JSON je vhodnější pro serializace komplexních datových struktur [3].

Příklad zápisu entity student pomocí formátu JSON:

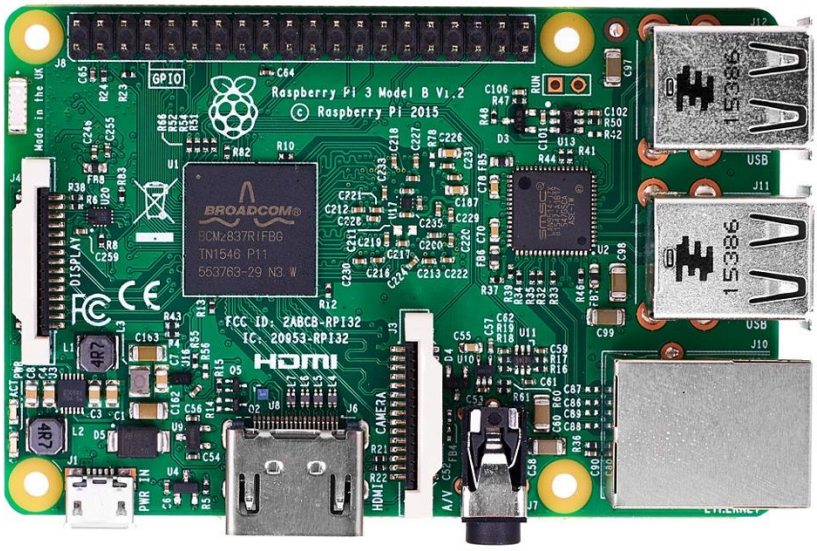
```
{
    "id": 68000,
    "name": "Jaromír Jágr",
    "address": {
        "street": "Nerudova 1840",
        "city": "Pardubice",
        "postalCode": 53002
    },
    "phoneNumbers": [
        "777 123 456",
        "777 654 321"
    ]
}
```

2 HARDWARE

Tato kapitola se věnuje hardwarové části práce, popisuje jednodeskový počítač Raspberry Pi, který je hlavním hardwarovou komponentou této práce, dotykový displej, pomocí něhož se provádí ovládání aplikace, a externí GPS přijímač který přidává možnost určení aktuální polohy.

2.1 Raspberry Pi

Raspberry Pi je mikropočítač s velikostí desky srovnatelné s velikostí platební karty. Původně byl navržen jako levný systém pro výuku informatiky, ale získal širší využití, než očekávali jeho autoři. V rámci projektu používá se model, který byl navrhnut na začátku roku 2016, a je to Raspberry Pi 3 model B. Hlavní rozdíl oproti předchozím verzím je využití 64-bitového CPU architektury ARM. Tento model je dvakrát výkonnější než jeho předchůdce, kromě toho počítač dostal integrovaný WIFI a Bluetooth modul, což eliminuje nutnost nákupu externích adaptérů, rozhraní LAN pořád je k dispozici [4].



Obrázek 2 - Raspberry Pi 3 model B[1]

[1] Zdroj: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Vlastnosti:

- architektura: ARMv8,
- procesor: 1.2GHz 64-bitový čtyřjádrový ARM Cortex-A53,
- paměť (SDRAM): 1 GB (sdílená s GPU),
- USB 2.0 porty: 4,
- interní paměť: MicroSDHC slot,
- síť: 10/100 Mbit Ethernet, 802.11n, Bluetooth 4.1.

Na oficiálních stránkách výrobce jsou k dispozici různé operační systémy, které oficiálně podporují mikropočítačí, většinou jsou na bázi GNU/Linuxu. Během projektu byl využít operační systém Raspbian, který byl vyvinut Raspberry Pi Foundation. Tento systém je derivátem operačního systému Debian, který je optimalizován pro využití na ARM procesorech Raspberry Pi. Původně byl systém poskytován spolu s málo modifikovaným desktopovým prostředím LXDE (Lightweight X11 Desktop Environment), které se ničím nelišilo od ostatních modifikací pro Linux distribuce. V roce 2016 Raspberry Pi Foundation prezentovala desktopové prostředí PIXEL (Pi Improved Xwindows Environment, Lightweight). Během dvou let prací, na základě LXDE bylo vyvíjené kompletně obnovené desktopové prostředí s přitažlivým vzhledem, panelem z vrcholu (původně v LXDE panel se nachází ve spodku obrazovky) a novým menu. Pro nejjednodušší instalace systému se používá nástroj NOOBS (New Out Of Box Software) [5].

2.2 Dotykový displej

Na začátku projektu byl použit 3,5-palcový displej od společnosti Adafruit. Bohužel tento displej byl špatnou volbou. Vzhledem k malé velikosti obrazovky se objevil problém s návrhem vzhledu aplikace. Také během jeho použití se vyskytovaly problém s vykreslováním obsahů obrazovky, které trvalo příliš dlouho. V důsledku toho, bylo rozhodnuto nahradit tento displej jiným.

Pro účely zobrazení informací a ovládání aplikace byl finálně použit 7-palcový dotykový displej, který oficiálně byl vydán společností Raspberry Pi Foundation [6]. Tento displej je speciálně vytvořen pro Raspberry Pi, s jeho pomocí lze mikropočítač využívat jako tablet, interaktivní informační stánek, řídicí systém nebo jako stolní počítač. Ve srovnání s původním displejem, tento displej má rychlejší odezvu dotyku a jeho 7-palcová obrazovka umožnila

vytvořit uživatelské přívětivé rozhraní, které je zcela určeno k ovládání pomocí prstů. Proces připojování tohoto displeje je velmi jednoduchý a nezabere víc než 10 minut [7].

Obrázek 3 - Raspberry Pi touch display[2]

Vlastnosti:

- Rozlišení: 800x480,
- Konektivita: DSI port,
- Typ panelu: LCD,
- Snímání dotyku: kapacitní.

2.3 GPS přijímač Navilock NL-602U

Pomocí využití již existujícího Web API je možnost získat informaci o vlaku, předáním jeho aktuální polohy. Pro získání souřadnice se používá GPS přijímač společnosti Navilock konkrétně Navilock NL-602U.

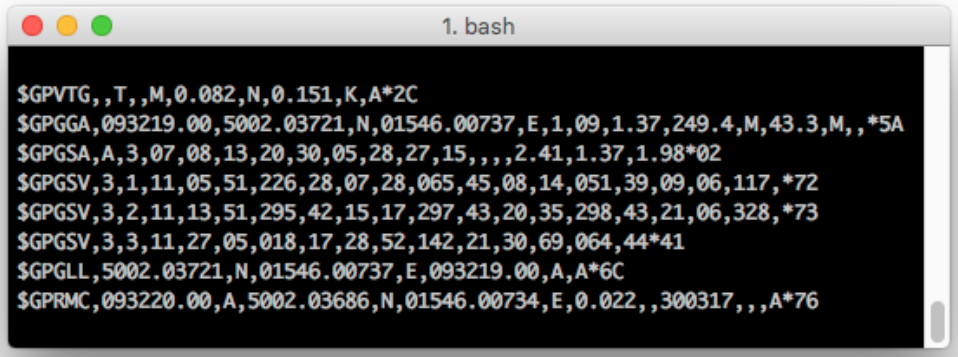
Vlastnosti:

- Konektor: USB 2.0,
- Chipset: u-blox 6 UBX-G6000-BT,
- Frekvence: GPS L1, 1575,4200 MHz a GLONASS L1, 1602,5625 - 1615,5000 MHz,
- Podpora protokolu NMEA 0183.

[2] Zdroj: https://www.wiltronics.com.au/product/8981/raspberry-pi-7-inch-touchscreen-display/

Obrázek 4 - GPS přijímač Navilock NL-602U[3]

Tento přijímač umožňuje snadnou instalaci a velmi jednoduché používání [8]. Komunikace se zařízením prochází přes vstupné/výstupné sériové rozhraní. Rychlost přenosu dat se rovna 4800 baud. Pro přenos dat přijímač používá textový komunikační protokol NMEA, který je nejpoužívanějším protokolem přenosu dat pro GPS přijímače [9]. Tento protokol byl navržen za účelem standardizace sériové komunikace námořních elektronických zařízení: např. sonary, echolokátory, gyrokompasy a přijímače GPS. Také tento protokol se používá ve vlacích. Přenos dat v protokolu probíhá zasíláním jednotlivých řádků dat nazývaných datové věty, které obsahují specifická data spadající do určité kategorie a jsou zcela nezávislé na ostatních větách (Obrázek 5).



Obrázek 5 - Příklad dat přijatých z přijímače (NMEA)

Každá datová věta vždy začíná symbolem dolaru ($), dále následuje název věty a jednotlivé položky které odděleny čárkou. V rámci aplikace jsou použité hodnoty z řádků GPGGA a GPGSV, význam kterých je jasně popsán pro každý řádek. Podrobný popis práce s údaji, bude vysvětlen v části, která věnovaná implementaci aplikace.

[3] Zdroj: http://www.navilock.de/produkte/G_61840/merkmale.html

3 POUŽITÉ TECHNOLOGIE

Aplikace byla vyvinuta ve vyšším programovacím jazyce Java. Hlavním důvodem pro volbu jazyka Java byla možnost vyvíjet aplikace na běžném počítači a její další kompilace. Vzhledem k multiplatformnímu jazyku Java je možné aplikaci snadno distribuovat na Raspberry Pi bez jakýchkoliv oprav. Tato vlastnost výrazně zjednodušuje proces implementace a menší opravy mohou být okamžitě otestovány na běžném počítači, pokud je nutné otestování aplikace přímo na zařízení stačit jen provést nasazení *jar*[4] souboru. Druhý důvod je framework JavaFX, který slouží pro vytváření grafického uživatelského rozhrání.

Pro vývoj aplikaci bylo využito vývojové prostředí IntelliJ IDEA od společnosti JetBrains, která poskytuje vývojářské nástroje pro programátory. Jako nástroj pro správu, řízení a automatizaci sestavení (tzv. buildů) aplikace se používá Apache Maven (dále jen Maven). Hlavní výhoda použití Maven v projektu je možnost kontroly vnějších knihoven z jednoho místa v rámci projektu. Díky využití Maven není nutno samostatně stahovat použité knihovny, stačí jen jednou přidat závislost do konfiguračního souboru *pom.xml* a Maven je automaticky stáhne. Pokud je potřeba aktualizovat knihovnu stačí jenom upravit její verzi v konfiguračním souboru na požadovanou podobu. Struktura konfiguračního souboru *pom.xml* je velmi jednoduchá a je definovaná pomocí značkovacího jazyka XML. Pro přidání požadované knihovny do projektu, stačí jen vložit závislost (dependency) do seznamu všech závislostí (dependencies) [10].

Příklad konfiguračního souboru pom.xml:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project>
3.      <modelVersion>4.0.0</modelVersion>
4.
5.      <groupId>cz.upce</groupId>
6.      <artifactId>khanenyao</artifactId>
7.      <version>1.0-SNAPSHOT</version>
8.
9.      <dependencies>
10.         <dependency>
11.             <groupId>com.google.code.gson</groupId>
12.             <artifactId>gson</artifactId>
13.             <version>2.8.0</version>
14.         </dependency>
15.         <dependency>
16.             <groupId>com.jfoenix</groupId>
17.             <artifactId>jfoenix</artifactId>
18.             <version>1.2.0</version>
19.         </dependency>
20.     </dependencies>
21. </project>
```

[4] jar je kompresní souborový formát, používaný platformou Java k distribuci programů a knihoven.

3.1 Důvod použití JavaFX

Do roku 2007 byl ve světě Java hlavním nástrojem pro tvorbu grafického uživatelského rozhraní knihovna Swing. První verze Swingu byla distribuována roce 1995 spolu s první verzí jazyka Java. Knihovna poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí Swingu je možno vytvářet okna, dialogy, tlačítka, rámečky, rozbalovací seznamy atd. V prosinci roku 2008 se objevil nový nastroj grafického rozhraní - JavaFX. Od této doby začal Swing pomalu ustupovat. V dnešní době JavaFX téměř nahradil zastaralý Swing, jako nástroj pro tvorbu GUI rozhraní v Javě.

Vzhledem k výše uvedenému byla pro tvorbu grafického uživatelského rozhraní v rámci projektu zvolena moderní softwarová platforma JavaFX. Hlavní výhodou nad Swing je podpora značkovacího jazyka FXML který slouží deklarativnímu popisu GUI. Jako výsledek aplikace má odděleni pohledu (*view*) od funkční části (*controller*). JavaFX podporuje kaskádové styly (CSS), které umožňují jednoduché nastavení vzhledu aplikace. Kaskádové styly mohou existovat oddělené od *view* i od programu ve vlastním souboru *css*. Navíc, díky JavaFX, aplikace má schopnost pracovat s HTML 5 a dostává podporu dotykových zařízení [11]. Všechny výše uvedené funkce byly v plné míře využity při psaní aplikací.

Od roku 2014 JavaFX je v součásti s JRE/JDK. Bohužel to neplatí pro vestavené zařízení od verze JDK 8u33 pro ARM, ale lze zajistit podporu pomocí open source rozhraní – OpenJDK [12].

Příklad jednoduché JavaFX aplikace, bez využití view, controller a FXML:

```
1.  public class HelloJavaFX extends Application {
2.      public void start(Stage primaryStage) {
3.          Button btn = new Button();
4.          btn.setText("Say 'Hello JavaFX'");
5.          StackPane root = new StackPane();
6.          root.getChildren().add(btn);
7.          Scene scene = new Scene(root, 300, 250);
8.          primaryStage.setTitle("Hello JavaFX!");
9.          primaryStage.setScene(scene);
10.         primaryStage.show();
11.     }
12.     public static void main(String[] args) {
13.         launch(args);
14.
```

3.2 Knihovna JFoenix

Přestože JavaFX umí pracovat s kaskádovými styly, byla pro modernější vzhled aplikace, využita další knihovna JFoenix [13]. Vývojáři z projektu JFoenix provedli skvělou práci a vytvořili výborné rozšíření, které přidává nové komponenty do JavaFX podle Google Material Design [14].

Maven závislost pro přidání JFoenix do projektu:

```
<dependency>
    <groupId>com.jfoenix</groupId>
    <artifactId>jfoenix</artifactId>
    <version>1.2.0</version>
</dependency>
```

3.3 Knihovna Google Gson

Vzhledem k tomu že při využiti Web API požadovaná informace přichází ve formátu JSON, bylo nutné zajistit podporu práce s tímto formátem v jazyce Java. Samozřejmě je možné napsat jednoduchý *parser* pomocí kterého pak lze zpracovávat požadované hodnoty, ale díky velkému počtu knihoven pro jazyk Java je rychlejší a efektivnější využít již existující řešení pro práci s JSON. V tomto projektu byla využita knihovna Gson [15].

Knihovna Gson je vyvinuta ve společnosti Google. Původně tato knihovna byla použita v několika vnitřních projektech společnosti. Po nějaké době se rozhodlo vydat knihovnu do open-source, pro její další rozvoj. Gson je malá knihovna která je napsaná v jazyce Java a umožňuje převést objekty Javy do formátu JSON, stejně jako vytvářet objekty na základě formátu JSON.

Základní třída knihovny má stejný název - Gson. Pro vytváření instanci třídy je nutné použít jeden ze dvou možných způsobů:

```
Gson gson = new Gson();
Gson gson = new GsonBuilder().create();
```

První způsob vytvoří instanci třídy s výchozím nastavením, zatímco druhý způsob umožňuje uplatnit některé nastavení. Základní metody, které slouží pro serializace a deserializace Java objektů jsou toJson a fromJson.

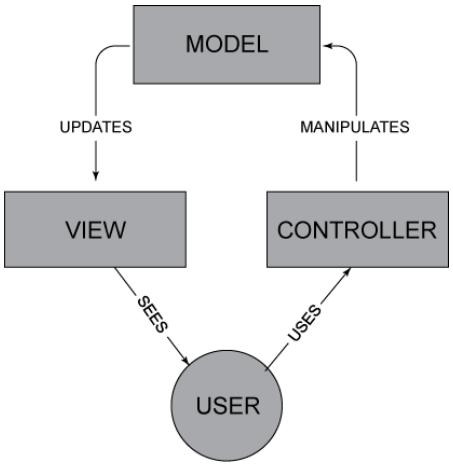
Příklad serializace a deserializace pomocí Gson:

```
1.  public class GsonExample {
2.
3.      private static class Student {
4.          public int id;
5.          String name;
6.
7.          public Student(int id, String name) {
8.              this.id = id;
9.              this.name = name;
10.         }
11.
12.         @Override
13.         public String toString() {
14.             return "[Information about Studnet]\nStudnent name: "+this.name+"\nStude
    nt id number: "+this.id;
15.         }
16.     }
17.
18.     public static void main(String[] args) {
19.         // Gson gson = new Gson();
20.
21.         // serializace
22.         Gson gson = new GsonBuilder().setPrettyPrinting().create(); // přidání struk
    turovaného výstupu
23.         Student student = new Student(43875, "Orest Khanenya");
24.         String JSON = gson.toJson(student);
25.         System.out.println(JSON);
26.         /*
27.             {
28.               "id": 43875,
29.               "name": "Orest Khanenya"
30.             }
31.          */
32.
33.         // deserializace
34.         String jsonString = "{\"id\": 68000, \"name\": \"Jaromír Jágr\"}";
35.         Student st68000 = gson.fromJson(jsonString, Student.class);
36.         System.out.println(st68000);
37.
38.         /*
39.             [Information about Studnet]
40.             Studnent name: Jaromír Jágr
41.             Student id number: 68000
42.          */
43.     }
44.
45. }
```

4 NÁVRH A IMPLEMENTACE APLIKACE

Tato kapitola je zaměřena na návrh a implementaci vlastní aplikaci s využitím technologií, které byly popsány v předchozí části práci. Kapitola popisuje strukturu projektu, jeho klíčové body a problémy jejich řešení.

4.1 Architektura aplikace



Obrázek 6 – Architektura MVC[5]

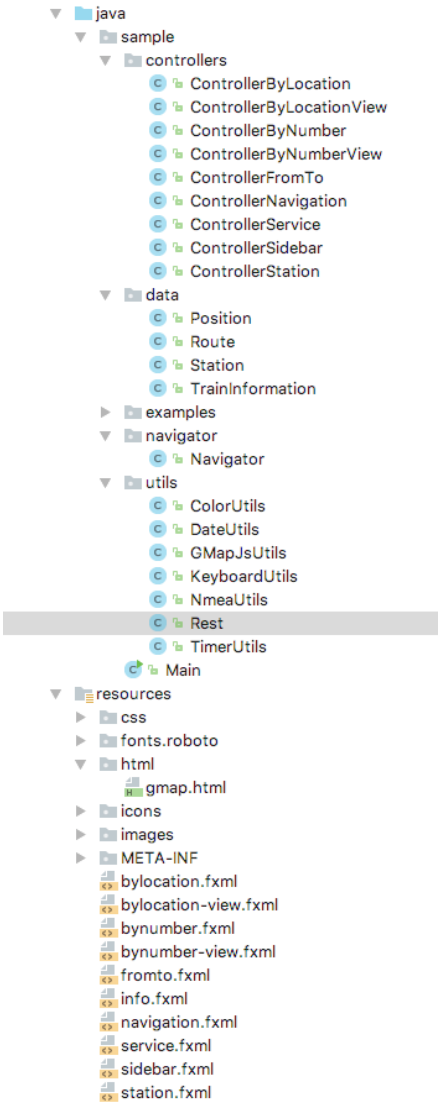
Vzhledem k využití JavaFX jako nástroje pro tvorbu grafického uživatelského rozhraní, které v procesu vývoje aplikaci umožňuje oddělení prezenční vrstvy (*view)* od zbytku programu. Kvůli tomu od začátku vývoje se používá návrhový vzor MVC (Model–View–Controller). Na základě tohoto návrhového vzoru je aplikační kód rozdělen do tří částí:

- Model (model) poskytuje data a reaguje na činnosti řadiče změnou svého stavu,
- View (pohled) zodpovědný za zobrazení datového modelu uživatelovi, reaguje na změny modelu,
- Controller (řadič) interpretuje akce uživatele, oznamuje model o nutnosti aktualizace stavu.

[5] Zdroj: https://www.ibm.com/developerworks/library/mo-prototype-watson/

4.2 Struktura projektu

Struktura projektu je rozdělená do několika částí. Každá část reprezentuje skupinu, jejíž členové plní podobné úkoly.



Obrázek 7 - Struktura projektu

První taková skupina je *controllers*, která zastupuje všechny řadiče dané aplikace. Členové teto skupiny jsou umístěný v adresáři controllers. Hlavním cílem kontrolérů je řízení stavů pohledů. Název třídy, která patří do této skupiny, se skládá pomocí prefixu Controller a názvu pohledu, který tento kontrolér řídí. Jako příklad lze uvést: kontrolér *ControllerNavigation* který řídí stavem pohledu *navigation.fxml*.

Seznam kontrolérů:

- *ControllerByLocation* – řídí stavem pohledu *bylocation.fxml,*
- *ControllerByLocationView* - řídí stavem pohledu *bylocation-view.fxml,*
- *ControllerByNumber* - řídí stavem pohledu *bynumber.fxml,*
- *ControllerByNumberView* - řídí stavem pohledu *bynumber-view.fxml,*
- *ControllerFromTo* - řídí stavem pohledu *fromto.fxml,*
- *ControllerNavigation* – hlavní kontrolér aplikace, odpovědný za řízení stavu *navigation.fxml,*
- *ControllerService* - řídí stavem pohledu *service.fxml,*
- *ControllerSidebar* - řídí stavem pohledu *sidebar.fxml,*
- *ControllerStation* - řídí stavem pohledu *station.fxml*.

Pohledy tvoří druhou skupinu. Definice pohledů se provádí ve souborech s příponou *fxml*, název který odpovídá jejich účelům. Hlavním cílem pohledů je vytváření GUI. Každý pohled zastupuje určitou část grafického rozhraní aplikace. Větší pohledy představují takzvané layouty, které označují rozmístění základních prvků na obrazovce. Pokud je pohled dynamický, mění svůj stav během běhu aplikace přičemž pro jeho správu stavu se používá příslušný kontrolér. Použitím kontrolérů lze kombinovat pohledy, čímž se vytváří komplexní vzhled aplikace. Všechny pohledy jsou umístěny v adresáři *resources*.

Seznam pohledů:

- *bylocation.fxml* - je layout, v rámci kterého uživatel zadává udají,
- *bylocation-view.fxml* - je layout, určený k zobrazení výsledku,
- *bynumber.fxml* - je layout, v rámci kterého uživatel zadává udají,
- *bynumber-view.fxml* - je layout, určený k zobrazení výsledku,
- *fromto.fxml* – malý pohled který slouží ke zobrazení informací ohledné výchozí a cílové stanici,
- *info.fxml* – střední velikosti pohled, slouží pro zobrazení všeobecné informací,
- *service.fxml* - malý pohled který je využít jako šablona pro zobrazení informace ohledné vlakové služby,
- *sidebar.fxml* - malý pohled který je využít pro umístění komponentů v rámci vyjížděcího menu aplikace,

- *station.fxml* - malý pohled který je využít jako šablona pro zobrazení informaci ohledné stanice.

Třetí skupinu tvoří třida *Navigator*, která umístěná v adresáři se stejným názvem. Tato třída bude probraná v další části práci

Vzhledem k použití návrhového vzoru MVC, všechny použité modely se oddělují od kontrolérů a jejich pohledů. Proto byla založena čtvrtá skupina – data. Všechny členové teto skupiny se používají k ukládaní informaci, která byla obdržená z Web API. Třídy teto skupiny umístěny v adresáři data.

Seznam modelů:

- *Position* – uschová souřadnice,
- *Route* – uschová souřadnice která je přilehlá k cestě mezi dvěma stanicemi,
- *Station* - uschová informací ohledné stanice,
- *TrainInformation* – uschovává informací obdrženou z Web API, v rámci teto třídy se používá třídy *Station* a *Position*.

Poslední skupina v rámci projektu je *utils*. Členové teto skupiny jsou pomocnými třídami, které usnadňují provádění úzce zaměřených úkolů. Vetší část členů teto skupiny, mají pouze jednu metodu, co potvrzuje jejich úzkou specializaci. Adresářem skupiny je složka *utils*.

Seznam utilit:

- *ColorUtils* – tato třida obsahuje všechny použité barvy,
- *DateUtils* – provádí formátování dat,
- *GMapJsUtils* – usnadňuje práci s Google mapy,
- *KeyboardUtils* – realizuje moznost přidáni akce při zadržení klávesy,
- *NmeaUtils* – zpracování a převádění souřadnic obdržených s GPS,
- *TimerUtils* – realizuje univerzální způsob práce s časovačem,
- *Rest* – slouží k odesíláni požadavků na Web API.

4.3 Navigace

V aplikaci existuje několik různých *layoutů*, mezi kterými bylo nutné realizovat možnost výměny vzhledu během běhu aplikace. Pro tyto účely bylo vytvořeno hlavní grafické rozhraní aplikace, které je definované v souboru *navigation.fxml*. Pro řízení stavu rozhraní se používá třída *ControllerNavigation*.

navigation.fxml:

```
<AnchorPane fx:controller="sample.controllers.ControllerNavigation">
   <children>
      <JFXToolbar>
         <left>
            <JFXToolbar>
               <left>
                  <JFXHamburger />
               </left>
               <center>
                  <Label>
                     <font>
                        <Font size="12.0" />
                     </font>
                  </Label>
               </center>
            </JFXToolbar>
         </left>
      </JFXToolbar>
      <AnchorPane fx:id="contentPane" />
      <JFXDrawer fx:id="drawer" />
   </children>
</AnchorPane>
```

Vztah mezi kontrolérem a rozhraním je definován v souboru *navigation.fxml* pomocí atributu *fx:controller*. Zobrazení všech dalších layoutů bude probíhat přes komponentu *AnchorPane*, který označen atributem *fx:id* s hodnotou *contentPane*. Pro kompletní spojení rozhraní a kontroléru v třídě *ControllerNavigation* se vytváří komponent *AnchorPane* s názvem *contentPane*.

```
@FXML
private AnchorPane contentPane;
```

Aby bylo možné měnit obsah komponenty *contentPane* mimo třídu *ControllerNavigation*, byla vytvořena třída *Navigator* za účelem přepínání mezi layouty v rámci aplikace. Tato třída obsahuje instanci třídy *ControllerNavigation*, metodu *setMainController*, která slouží k nastavení hlavního kontroléru aplikaci a metodu *load*, která pomocí instanci třídy *ControllerNavigation* a volaní metody *setPane* provádí změnu obsahů komponent *contentPane*.

```
1.  public class Navigator {
2.
3.      public static final String NAVIGATION = "/navigation.fxml";
4.      public static final String BY_NUMBER = "/bynumber.fxml";
5.      public static final String BY_LOCATION = "/bylocation.fxml";
6.
7.      private static ControllerNavigation mainController;
8.
9.      public static void setMainController(ControllerNavigation mainController) {
10.         Navigator.mainController = mainController;
11.     }
12.
13.     public static void load(String fxml) {
14.         try {
15.             mainController.setPane(
16.                     FXMLLoader.load(
17.                             Navigator.class.getResource(
18.                                     fxml
19.                             )
20.                     )
21.             );
22.         } catch (IOException e) {
23.             e.printStackTrace();
24.         }
25.     }
26.     ...
27. }
```

Také třida *Navigator* obsahuje cesty k souborům, které definují grafické rozhraní, pro jejich další snadné použití. Hlavní třídou aplikace je *Main*. Tato třída je potomkem třídy *Application* a je zodpovědná za spuštění aplikace (využívá se metoda start) a zobrazení všech layoutů, které umísťuje v sobě scéna aplikace. Pro tvorbu scény byla vytvořena metoda *createScene*, která akceptuje kaskádové styly, které jsou definované v souboru *style.c*ss.

```
1.  private Scene createScene(Pane mainPane) {
2.      Scene scene = new Scene(mainPane);
3.      scene.getStylesheets().setAll(
4.          getClass().getResource("/css/style.css").toExternalForm()
5.      );
6.
7.      return scene;
8.  }
```

Jak je vidět metoda *createScene* jako vstupní parametr přijímá objekt třídy *Pane*, pomocí kterého se vytváří scéna. V tomto případě *mainPane* je hlavní layout aplikace a využívá metodu– *loadMainPane*.

```
1.  private Pane loadMainPane() throws IOException {
2.      FXMLLoader loader = new FXMLLoader();
3.
4.      Pane mainPane = (Pane) loader.load(
5.          getClass().getResourceAsStream(
6.              Navigator.NAVIGATION
7.          )
8.      );
9.
10.     Controller mainController = loader.getController();
```

```
11.
12.     Navigator.setMainController(mainController);
13.
14.     Navigator.load(Navigator.BY_NUMBER);
15.
16.     return mainPane;
17. }
```

Použití a zobrazování scény se provádí v metodě start:

```
1.  @Override
2.  public void start(Stage stage) throws Exception{
3.      stage.setScene(
4.          createScene(
5.              loadMainPane()
6.          )
7.      );
8.
9.      stage.setResizable(false);
10.     stage.show();
11. }
```

4.4 Sidebar

Po vytváření zakladu pro provádění navigace mezi layouty vzniká potřeba implementace grafického prvku ovládání, který v spolupráci s třídou *Navigator* bude provádět změny uvnitř komponentu *contentPane*. Pro tyto účely byla vybrána komponenta JFXDrawer, kterou poskytuje knihovna JFoenix. Tento komponent představuje dynamicky vyjíždějící menu, obsah kterého je umístěn do další komponenty VBox. Vzhled je definován v souboru *sidebar.fxml*. Obsahem komponenty VBox jsou tlačítka, pomocí kterých uživatel provádí přesměrování na příslušný layout. Volání vyjíždějícího menu se provádí pomoci komponentu JFXHamburger umístěnou v navigačním panelu aplikace.

```
1.  <VBox fx:controller="sample.controllers.ControllerSidebar">
2.     <children>
3.        <ImageView>
4.           <image>
5.              <Image url="images/bckg.jpg" />
6.           </image>
7.        </ImageView>
8.        <JFXButton fx:id="byNumberBtn" accessibleText="BY_NUMBER" />
9.        <JFXButton fx:id="byPositionBtn" accessibleText="BY_LOCATION" />
10.       <JFXButton fx:id="exitBtn" />
11.    </children>
12. </VBox>
```

Jak je vidět z výše uvedeného kódu *sidebar.fxml* má svůj vlastní kontrolér *ControllerSidebar*, který řeší ukončení běhu aplikace v případě stlačení tlačítka *exitBtn*. Hlavní věc, na kterou je třeba dávat pozornost je atribut *accessibleText* u tlačítek *byNumberBtn* a *byPositionBtn*. Hodnota atributu je použita v logice přesměrování, která je popsaná ve třídě *ControllerNavigation*.

Logika přesměrování je definovaná uvnitř metody *initialize* a je velmi jednoduchá. Výběr layotu prohází přes hodnotu z atributu *accessibleText* a přesměrování se pak řeší pomocí třídy *Naviagtor* a její metody *load*.

```
1.  VBox box = FXMLLoader.load(getClass().getResource("/sidebar.fxml"));
2.  drawer.setSidePane(box);
3.
4.  for (Node node : box.getChildren()) {
5.      if (node.getAccessibleText() != null) {
6.          node.addEventHandler(MouseEvent.MOUSE_CLICKED, (e) -> {
7.              switch (node.getAccessibleText()) {
8.                  case "BY_NUMBER":
9.                      Navigator.load(Navigator.BY_NUMBER);
10.                     drawer.close();
11.                     break;
12.                 case "BY_LOCATION":
13.                     Navigator.load(Navigator.BY_LOCATION);
14.                     drawer.close();
15.                     break;
16.             }
17.         });
18.     }
19. }
```

4.5 Získaní dat z webového rozhraní

Pro účely odeslání požadavku Web API byla vytvořena třída *Rest*. Tato třída obsahuje jedinou metodu s názvem *get* a to za účelem odesíláni HTTP dotazu a přijetí odpovědi z Web API. Jako vstupní data se očekává sestavený řetězec URL. Jako výsledek své činnosti metoda vrací JSON odpověď přijatou z Web API, v textové podobě.

```
1.  public static String get(String strUrl) {
2.      try {
3.
4.          URL url;
5.          url = new URL(strUrl);
6.          HttpURLConnection conn = (HttpURLConnection) url.openConnection();
7.          conn.setRequestMethod("GET");
8.          conn.setRequestProperty("Accept", "application/json");
9.
10.         if (conn.getResponseCode() != 200) {
11.             throw new RuntimeException("Failed : HTTP error code : "
12.                     + conn.getResponseCode());
13.         }
14.
15.         BufferedReader br = new BufferedReader(new InputStreamReader(
16.                 (conn.getInputStream())));
17.
18.         String output;
19.         StringBuilder response = new StringBuilder();
20.
21.         while ((output = br.readLine()) != null) {
22.             response.append(output);
23.         }
24.
```

```
25.         conn.disconnect();
26.         return response.toString();
27.     } catch (MalformedURLException e) {
28.         e.printStackTrace();
29.     } catch (IOException e) {
30.         e.printStackTrace();
31.     }
32.
33.     return null;
34. }
```

Pro rychlou a snadnou práci s odpovědí se provádí deserializace JSON do objektu Java. Pro tyto účely byl připraven model, který je prezentován třídou *TrainInformation*. Uvnitř třídy je metoda *getTrainByNumber*, která slouží ke získávání dat na základě čísla vlaku. Zároveň existuje metoda *getTrainByCoords*, která vrací informaci vzhledem k aktuální poloze. Teto dvě metody se od sebe liší pouze svým požadavkem a vstupními parametry, jinak je to úplně totožný princip práce. V důsledku předem vytvářenému modelu a knihovně Gson, je pak příjem a ukládání dat v rámci aplikace velmi jednoduchým úkolem.

```
1.  public static TrainInformation getTrainByNumber(long number) {
2.      String response = Rest.get("http://fei-
    hosting.upceucebny.cz:8080/trains/webapi/train?number="+number);
3.
4.      TrainInformation trainInf = GSON.fromJson(response, TrainInformation.class);
5.
6.      return  trainInf;
7.  }
```

4.6 Spolupráce s GPS modulem

Jak už bylo zmíněno, pro přenos dat mezi přijímačem a mikropočítačem se používá sériové rozhrání. Po připojení přijímače přes USB kabel, v systému, se objeví virtuální sériový port */dev/ttyACM0*. Získat informaci z virtuálního portu je možné použitím terminálového příkazu *cat /dev/ttyACM0*. Spolupráce s přijímačem v rámci aplikace je řešeno v metodě *getActualPosition* třídy *ControllerByLocation*. Hlavním cílem teto metody je určení aktuální polohy.

```
1.  private void getActualPosition() {
2.      new Thread() {
3.          public void run() {
4.              String s;
5.              Process p;
6.              Position tempPos = new Position();
7.              float azimuthTemp = 0;
8.              try {
9.                  p = Runtime.getRuntime().exec("cat /dev/ttyACM0");
10.                 BufferedReader br = new BufferedReader(
11.                     new InputStreamReader(p.getInputStream())
12.                 );
13.                 while ((s = br.readLine()) != null) {
14.                     if (s.contains("$GPGGA")) {
15.                         tempPos = NmeaUtils.getPositionFromNmea(s);
```

```
                    } else if (s.contains("$GPGSV")) {
                        azimuthTemp = NmeaUtils.getAzimuthFromNmea(s);
                        p.destroy();
                    }
                }
            } catch (Exception e) {}
            Position newPosition = tempPos;
            float azimuth = azimuthTemp;
            Platform.runLater(() -> {
                latitudeTxtFld.setText(String.valueOf(newPosition.getLatitude()));
                longitudeTxtFld.setText(String.valueOf(newPosition.getLongitude()));

                azimuthTxtFld.setText(String.valueOf(azimuth));
            });
        }
    }.start();
}
```

Přestože komunikace se zařízením je velmi rychlá, všechny činnosti se provádějí uvnitř anonymního vlákna. Po přijatí informace o aktuální poloze se pomocí třídy *NmeaUtils* provádí zpracování dat. Jakmile jsou data zpracované a jsou k dispozici, přepne se z anonymního vlákna na vlákno JavaFX ve kterém se provádí aktualizace grafických komponent. Provádět přepínaní do vlákna JavaFX je nutné z důvodu zakázané aktualizace komponentů z jiných vláken programu.

4.7 NmeaUtils

Tato třída se zabývá převodem souřadnic přijatých z GPS přijímače, které byly předané pomocí protokolu NMEA.

Zeměpisné souřadnice bodu mohou být vyjádřené v různých formátech. Formát souřadnic je obvykle definován následujícím způsobem: *DD* – stupně, *MM* – minuty, *SS* – vteřiny. Celkem existují tři možnosti vyjádření:

- Stupně (50,25345) – formát *DD.ddddd*,
- Stupně Minuty (50 14,326) – formát *DD MM.mmm*,
- Stupně Minuty Vteřiny (50 14 25,415) – formát *DD MM SS.sss*.

Nejčastějším formátem pro zápis souřadnic u navigačních zařízení je *DD MM.mmm*. Google, Seznam a většina geografických informačních systémů pracují s formátem *DD.ddddd*. Kvůli tomu se v rámci aplikace objevila nutnost v převodu souřadnic z *DD MM.mmm* formátu do *DD.ddddd*. Tento převod se provádí pomocí vzorku:

$$DD.ddddd = DD + (\frac{MM.mmm}{60}) \quad (1)$$

Souřadnice se rozdělují na zeměpisnou šířku a zeměpisnou délku. Zeměpisná šířka určuje polohu k severu nebo jihu směrem od rovníku. Severní šířka se označuje symbolem *N*, jižní symbolem *S*. Zeměpisná délka určuje polohu k východu nebo západu od nultého Greenwichského poledníku. Východní délka značí se písmenem *E*, západní symbolem *W*. Z pohledu aplikační logiky symboly *N* a *E* říkají, že souřadnici jsou kladné, *W* a *S* jsou záporné.

```
1.   public class NmeaUtils {
2.      private static final float SECONDS = new Float("0.01666666667"); //1/60
3.
4.      public static Position getPositionFromNmea(String nmeaStr) {
5.          List<String> items = Arrays.asList(nmeaStr.split(","));
6.          float lat = (items.get(3).equals("N"))? convertNmeaLon(items.get(2)) : conve
    rtNmeaLon(items.get(2))*(-1);
7.          float lon = (items.get(5).equals("E"))? convertNmeaLat(items.get(4)) : conve
    rtNmeaLat(items.get(4))*(-1);
8.
9.          return new Position(lat,lon);
10.     }
11.
12.     private static float convertNmeaLon(String lon) {
13.         return Float.valueOf(lon.substring(0,2)) + Float.valueOf(lon.substring(2,lon
    .length())) * (SECONDS);
14.     }
15.
16.     private static float convertNmeaLat(String lat) {
17.         return Float.valueOf(lat.substring(0,3)) + Float.valueOf(lat.substring(3,lat
    .length())) * (SECONDS);
18.     }
19.
20.     public static float getAzimuthFromNmea(String nmeaStr) {
21.         List<String> items = Arrays.asList(nmeaStr.split(","));
22.         float azimuth = Float.valueOf(items.get(6));
23.         return azimuth;
24.     }
25. }
```

Metoda *getPositionFromNmea* očekává na vstup datovou větu GPGGA, která obsahuje poslední nalezené souřadnice a další doplňující informace. Z této datové věty se pak vytahuje zeměpisná délka a šířka. Metody *convertNmeaLon* a *convertNmeaLat* se zabývají převodem souřadnic podle vzoru, který je byl probrán výše. Poslední metodou je *getAzimuthFromNmea*. Na vstup se očekává datová věta GPGSV, ze které se vytahuje hodnota azimutu.

4.8 Zobrazení na mapě

Jedním z obtížnějších úkolů je zobrazení kolejového vozidla na mapě. Bohužel, knihovny JavaFX a JFoenix nemají přepravené komponenty pro prací s mapou. Původně, pro řešení tohoto úkolu se plánovalo využít knihovnu GMapsFX. Tato knihovna přidává do JavaFX podporu Google map, bez nutnosti využití základního JavaScript API. Ale během prací s touto knihovnou se objevily problémy, které souvisejí s vyžitím zastaralého API ze strany knihovny.

Kvůli tomu problému byla práce s mapou vyřešená vlastním přístupem, bez využití knihovny GMapsFX.

Nejprve byla připravená *html* stránka *gamp.html*, v které se provádí inicializace mapy pomocí *JavaScriptu*. Kromě inicializace mapy, *gmap.html* obsahuje sadu funkcí napsané v jazyce *JavaScript*. Tyto funkce řeší správu mapy, jako změna centra nebo měřítka mapy. Připravenou mapu lze otevřít v libovolném webovém prohlížečů. Pro zobrazení webového obsahů v JavaFX existuje skvělá třida *WebWiew*.

Pro účely zobrazení a správy mapy byla vytvořená třída *GMapJsUtils*:

```
1.  public class GMapJsUtils {
2.      private JSObject doc;
3.      private WebView webView;
4.      private WebEngine webEngine;
5.      private String MAP_HTML = "/html/gmap.html";
6.      private boolean ready;
7.
8.      public GMapJsUtils(int prefWidth, int prefHeight) {
9.          webView = new WebView();
10.         webView.setPrefSize(prefWidth,prefHeight);
11.         webEngine = webView.getEngine();
12.         webEngine.load(getClass().getResource(MAP_HTML).toExternalForm());
13.         initMap();
14.         initCommunication();
15.     }
16.
17.     private void initMap() {
18.         ready = false;
19.         webEngine.getLoadWorker().stateProperty().addListener(new ChangeListener<Wor
    ker.State>() {
20.             @Override
21.             public void changed(final ObservableValue<? extends Worker.State> observ
    ableValue,
22.                                 final Worker.State oldState,
23.                                 final Worker.State newState) {
24.                 if (newState == Worker.State.SUCCEEDED) {
25.                     ready = true;
26.                 }
27.             }
28.         });
29.     }
```

Jednoduché zobrazení mapy je však nedostatečné. Pro přístup k funkcím *JavaScriptu*, které jsou v souboru *gmap.html*, je nutné získat instanci třídy *JSObject* a využít metodu *initCommunication*, která se používá v konstruktoru třídy *GMapJsUtils*.

```
1.  private void initCommunication() {
2.          webEngine.getLoadWorker().stateProperty().addListener(new ChangeListener<Wor
    ker.State>() {
3.              @Override
4.              public void changed(final ObservableValue<? extends Worker.State> observ
    ableValue,
5.                                  final Worker.State oldState,
6.                                  final Worker.State newState) {
```

```
                if (newState == Worker.State.SUCCEEDED) {
                    doc = (JSObject) webEngine.executeScript("window");
                }}
        });
    }
```

Za účelem volaní funkcí *JavaScriptu* z Java uvnitř třídy *GMapJsUtils* je definovaná metoda *invokeJS*:

```
private void invokeJS(final String str) {
        if (ready) {
            doc.eval(str);
        } else {
            webEngine.getLoadWorker().stateProperty().addListener(new ChangeListener
<Worker.State>() {
                @Override
                public void changed(final ObservableValue<? extends Worker.State> ob
servableValue,
                                    final Worker.State oldState,
                                    final Worker.State newState) {
                    if (newState == Worker.State.SUCCEEDED) {
                        doc.eval(str);
                    }
                }
            });
        }
    }
```

Příkladem použití metody *invokeJS* je změna centra mapy. Pro tento účel, v souboru *gmap.html* je napsaná *JavaScriptová* funkce *setMapCenter*. Pomocí *JavaScriptového* API od Googlu, tato funkce provádí změnu centrálního bodu mapy.

```
function setMapCenter(lat, lng) {
            actual = new google.maps.LatLng(lat, lng);
            map.setCenter(new google.maps.LatLng(lat, lng));
        }
```

Pro snadnější použití teto funkce, je ve třídě *GMapJsUtils* vytvořena metoda *setMapCenter*, která pomocí metody *invokeJS* volá *JavaScriptovou* funkci *setMapCenter* ze souboru *gmap.html*.

```
public void setMapCenter(double lat, double lng) {
        invokeJS("setMapCenter(" + lat + ", " + lng + ")");
    }
```

Pro další prací s mapou v rámci aplikace JavaFX je využita metoda *getMap* třídy *GMapJsUtils*:

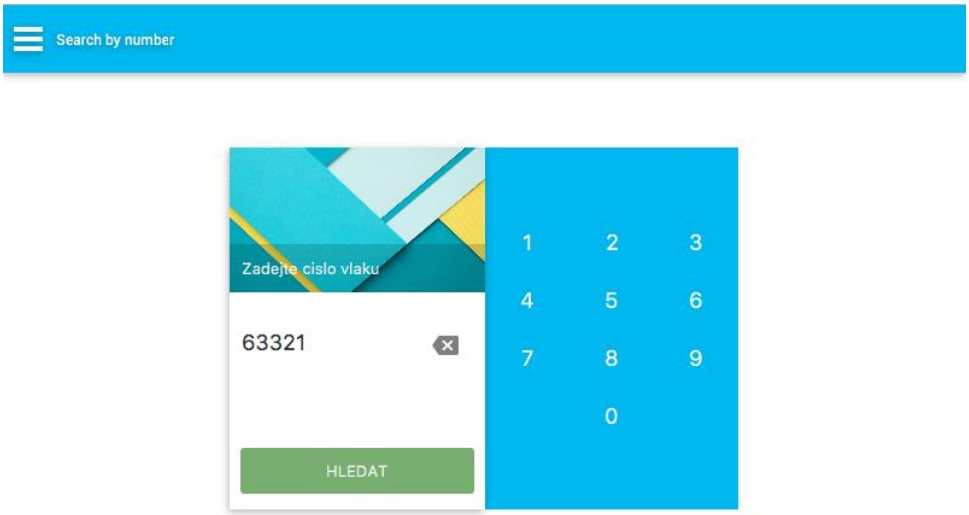
```
public WebView getMap() {
        return this.webView;
    }
```

5 UŽIVATELSKÉ ROZHRANÍ

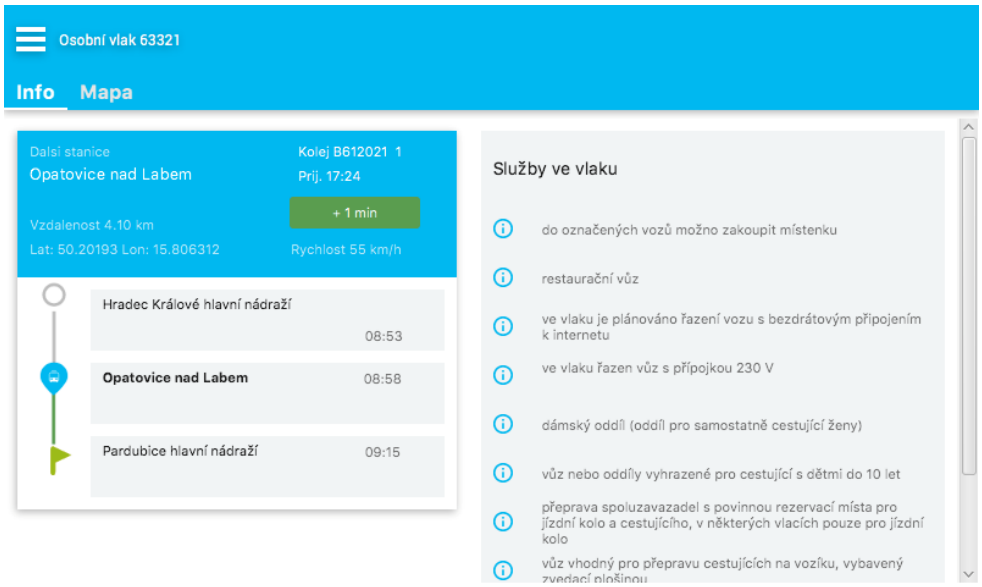
Než uživatel spustí aplikaci, je nutné zajistit správné VPN spojení. Je to způsobeno tím, že Web API je dostupné pouze z vnitřní sítě univerzity. Pro připojeni se používá CLI klient openconnect. Tento klient je zcela kompatibilní se serverem Cisco AnyConnect, který se využívá na univerzitě. Instalace klienta se provádí pomoci následujícího příkazu *apt-get install openconnect*.

Pro rychlejší připojení se používá vlastní *bash script*, který automatizuje proces připojení. Tento scriptu eliminuje nutnost zadaní prohlašovacích údajů. Všechny přihlašovací údaje se nacházejí uvnitř scriptu.

Po zajištění připojení ke VPN, uživatel může provést spuštění aplikace. Pro teto účely na pracovní ploše je spustitelný soubor s názvem „start“. Po spouštění aplikace se zobrazí obrazovka, v rámci které provádí se zadaní čísla vlaku (Obrázek 8). Po zpracovaní požadavku aplikace zobrazí veškerou nalezenou informací (Obrázek 9).

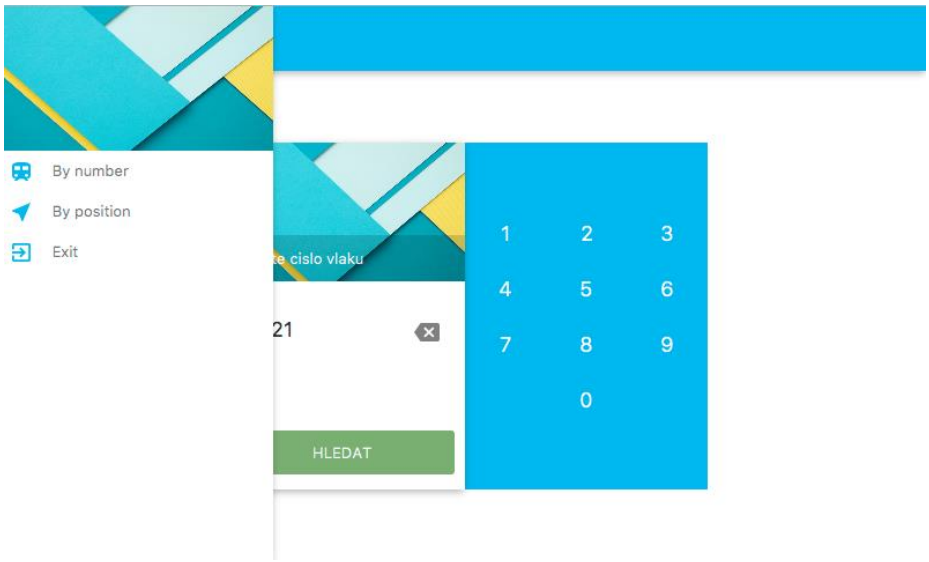


Obrázek 8 - Zadání čísla vlaku



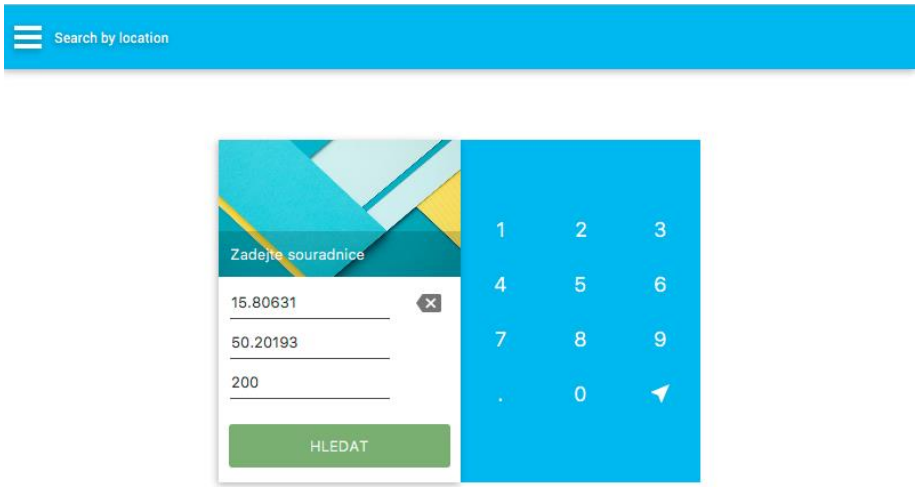
Obrázek 9 - Zobrazení informace o vlaku

Další možnost je zobrazení informace podle souřadnic. Uživatel otevře boční menu, pomocí tlačítka, které se nachází v navigačním panelu (levý horní roh). V otevřeném menu, zvolí možnost „By position“ (Obrázek 10).

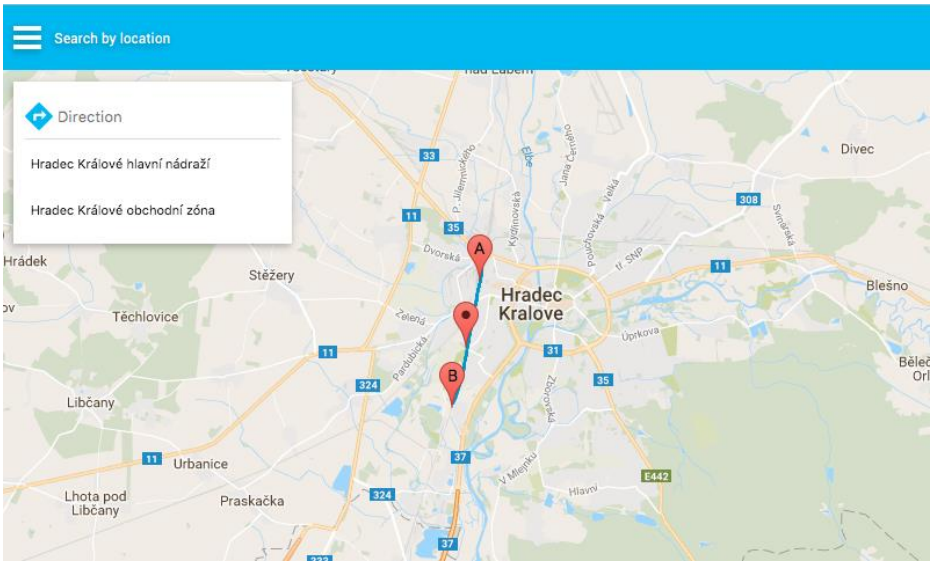


Obrázek 10 - Boční menu aplikace

Aplikace přesměruje uživatele na obrazovku, kde se očekává na vstup souřadnice, které uživatel může zadat buď ručně nebo použitím GPS (Obrázek 11). Po zpracovaní požadavku aplikace zobrazí aktuální polohu vlaku na mapě a dodatkové informace (Obrázek 12). Zobrazení polohy lze taky provést v rámci modulu zobrazení informace podle čísla vlaku zmáčknutím tlačítka „Mapa“ (Obrázek 9).



Obrázek 11 - Zadaní souřadnic



Obrázek 12 - Zobrazení polohy vlaku na mapě

6 ZÁVĚR

Cílem bakalářské práce bylo navrhnout aplikaci určenou pro použití mikropočítače Raspberry Pi, s využitím existujícího webového aplikačního rozhraní pro získaní a zobrazení informací o poloze kolejových vozidlech.

V teoretické části práce, byly vysvětleny použité webové služby, které byly využity v rámci navržené aplikace. V jednotlivých kapitolách byla popsána hardwarová část projektu a taky technologie které byly využity pro vlastní vývoj aplikace. V teoretické části práce jsou podrobně vysvětleny jednotlivá téma, která bezprostředně souvisejí s návrhem a implementací aplikace. Dále je v teoretické části, popsána struktura projektu, architektura aplikace a jsou uvedeny příklady použití aplikace.

V rámci praktické části byla provedena konfigurace všech použitých zařízení, návrh a implementace aplikace s využitím programovacího jazyka Java. Veškeré cíle bakalářské práce byly úspěšně splněny. Aplikace úspěšně komunikuje s webovým rozhraním a přijímá a zpracovává informace získané z modulu GPS. Navržené aplikace byla finálně testována a to pomocí podpůrného softwarového nástroje simulujícího provoz kolejových vozidel v rámci existujícího modelu železniční sítě.

7 POUŽITÁ LITERATURA

[1] FIKEJZ, Jan. *Systém na podporu dispečerského řízení železniční dopravy analyzující data o poloze kolejových vozidel získaných ze systémů GNSS*. Pardubice, 2016. Disertační. Univerzita Pardubice. Vedoucí práce Antonín Kavička.

[2] OBORNÍK, Jakub. *Návrh a implementace mobilní aplikace pro lokalizaci kolejového vozidla na železniční síti s využitím komunikačního rozhraní*. 2016. Bakalářská práce. Univerzita Pardubice. Vedoucí práce Jan Fikejz URI: http://hdl.handle.net /10195/66867

[3] SAI SRINIVAS SRIPARASA. *JavaScript and JSON essentials: successfully build advanced JSON-fueled web applications with this practical, hands-on guide*. Birmingham, UK: Packt Publishing, 2013. ISBN 9781783286034.

[4] UPTON, Eben. a Gareth. HALFACREE. *Raspberry Pi user guide*. Chichester, England: John Wiley, c2012. ISBN 978-1-118-46446-5.

[5] Kolektiv autorů. *NOOBS - Raspberry Pi Documentation*. www.raspberrypi.org. URL: https://www.raspberrypi.org/documentation/installation/noobs.md

[6] Kolektiv autorů. *Raspberry Pi Touch Display - Raspberry Pi.* www.raspberrypi.org. URL: https://www.raspberrypi.org/products/raspberry-pi-touch-display/

[7] JOEMAN. *Raspberry Pi 7" Touchscreen Display*. www.element14.com. URL: www.element14.com/community/docs/DOC-78156/l/raspberry-pi-7-touchscreen-display

[8] Kolektiv autorů. *NL-602U ublox6 USB receiver Operation manual (61840).* www.navilock.de. URL: http://www.navilock.de/produkte/G_61840/dokumente.html

[9] Kolektiv autorů. *NMEA data*. www.gpsinformation.org URL: www.gpsinformation. org/dale/nmea.html

[10] Kolektiv autorů. *Welcome to Apache Maven*. Apache Maven Project [online]. Last Published: 2017-04-27. URL: http://maven.apache.org/index.html

[11] DEA, Carl. *JavaFX 8: introduction by emample. Second edition*. ISBN 978-1-4302-6461-3.

[12] BLAUKOPF, Daniel. *OpenJFX on the Raspberry Pi*. openjdk.java.net. Dec 10, 2015. URL: https://wiki.openjdk.java.net/display/OpenJFX/OpenJFX+on+the+Raspberry+Pi

[13] Kolektiv autorů. *JFoenix*. github.com. URL: https://github.com/jfoenixadmin/JFoenix

[14] Kolektiv autorů. *Material design*. material.io. URL: https://material.io/guidelines/material-design/introduction.html

[15] Kolektiv autorů. *google-gson*. github.com. URL: https://github.com/google/gson