VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTERAKTIVNÍ VLAKOVÝ JÍZDNÍ ŘÁD PRO TABLETY

INTERACTIVE TRAIN DIAGRAM FOR TABLET COMPUTERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **PETR KALUSEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20466/2017/xkalus03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Kalusek Petr**
Obor: Informační technologie
Téma: **Interaktivní vlakový jízdní řád pro tablety**
Interactive Train Diagram for Tablet Computers
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte existující informační systém určený pro porovnávání vlakových spojů poskytující tzv. grafikon vlakové dopravy [Uhlíř]. Analyzujte jeho výhody a nedostatky.
2. Seznamte se s principem tvorby aplikací pro mobilní zařízení (se zaměřením na tablety).
3. Na základě pokynů vedoucího navrhněte mobilní aplikaci určenou pro interaktivní porovnávání vlakových spojů spolupracující s informačním systémem z bodu 1. Navrhněte mechanismus automatizovaného získávání potřebných dat o spojích.
4. Navrženou aplikaci implementujte.
5. Rozšiřte vhodně databázi spojů.
6. Otestujte přívětivost implementovaného uživatelského rozhraní a navrhněte možná rozšíření.

Literatura:

- Uhlíř, J. *Interaktivní porovnávání vlakových spojů*. Brno, 2017. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=17511. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. 2017-06-12. Vedoucí práce Hynek Jiří.
- Johnson, J.: *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Burlington: Morgan Kaufmann Publishers/Elsevier, 2010, ISBN 978-0-12-375030-3.
- Rogers, Y.; Sharp, H; Preece, J.: *Interaction Design: Beyond Human - Computer Interaction*. 3rd Edition. New York: Wiley, 2011, ISBN 978-0-470-66576-3.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hynek Jiří, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L. S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá návrhem a vývojem mobilní aplikaci pro operační systém Android. Cílem práce je implementace aplikace pro interaktivní porovnávání vlakových spojů pomocí grafikonu vlakové dopravy. Aplikace vyhledá trasy mezi zastávkami včetně přestupů a zobrazí spoje na těchto trasách v grafikonu. Ten umožňuje filtraci dat a operace posunutí a přiblížení. Data o vlakových spojích byly získány pomocí skriptu z webových služeb.

Abstract

This bachelor's thesis deals with the design and development of a mobile application for Android operating system. The aim of the work is implementation of application for interactive comparison of train connections using a graphic timetable. Application finds paths between stations including transfers a shows the connections on these paths in graphic timetable. This graph supports data filtration, scrolling and zooming. Train connection data were scraped from web services using a script.

Klíčová slova

grafikon vlakové dopravy, jízdní řád, vizualizace dat, automatizované získání dat, mobilní zařízení, tablet, Java, Android, SQLite, Python, CasperJS

Keywords

graphic timetable, train timetable, data visualization, web scraping, mobile devices, tablet, Java, Android, SQLite, Python, CasperJS

Citace

KALUSEK, Petr. *Interaktivní vlakový jízdní řád pro tablety.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek

Interaktivní vlakový jízdní řád pro tablety

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Kalusek
16. května 2018

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Jiřímu Hynkovi za jeho rady, trpělivost a vedení při vypracovávání této práce a za jeho čas věnovaný konzultacím.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je poskytnout uživateli přehled o veškerém provozu na vlakových tratích pomocí nákresného jízdního řádu. Tento jízdní řád využívající grafikon vlakové dopravy se běžně používá pro plánování a řízení provozu na trati, ale široká veřejnost k nim dobrý přístup nemá.

Výstupem práce je mobilní aplikace pro operační systém Android se zaměřením na tablety, které poskytují dostatečnou plochu pro přehledné zobrazení grafikonu. Uživatel bude moci využít aplikaci pro vyhledání vlakových spojů mezi 2 stanicemi s následným porovnáním a vybráním vhodného spoje přímo pro jeho potřeby. Aplikace bude podporovat i zobrazení tras s přestupy s automatickým výběrem přestupních bodů a nabídnutím několika optimálních tras. Spoje se budou vyhledávat v lokální databázi, která bude aktualizovatelná novými daty. Aplikace tak bude použitelná i bez internetového připojení.

Tato bakalářská práce navazuje na bakalářskou práci [18] absolventa VUT FIT Bc. Jana Uhlíře, který již řešil zobrazování vlakového grafikonu pro porovnávání spojů a výsledkem jeho práce je webový informační systém, který má ale některé nedostatky, které se pokusím touto prací eliminovat.

Nejprve budou popsány základní informace o vlakovém grafikonu a o datech pro něj potřebných. Poté bude analyzován existující informační systém a určeny jeho nedostatky. Následně bude provedena analýza požadavků aplikace a možných technologií pro použití v aplikaci. Poté proběhne návrh a implementace aplikace a nakonec bude popsán průběh testování, jaké problémy se tím vyřešily a možné vylepšení do budoucna.

Kapitola 2

Vlakový grafikon

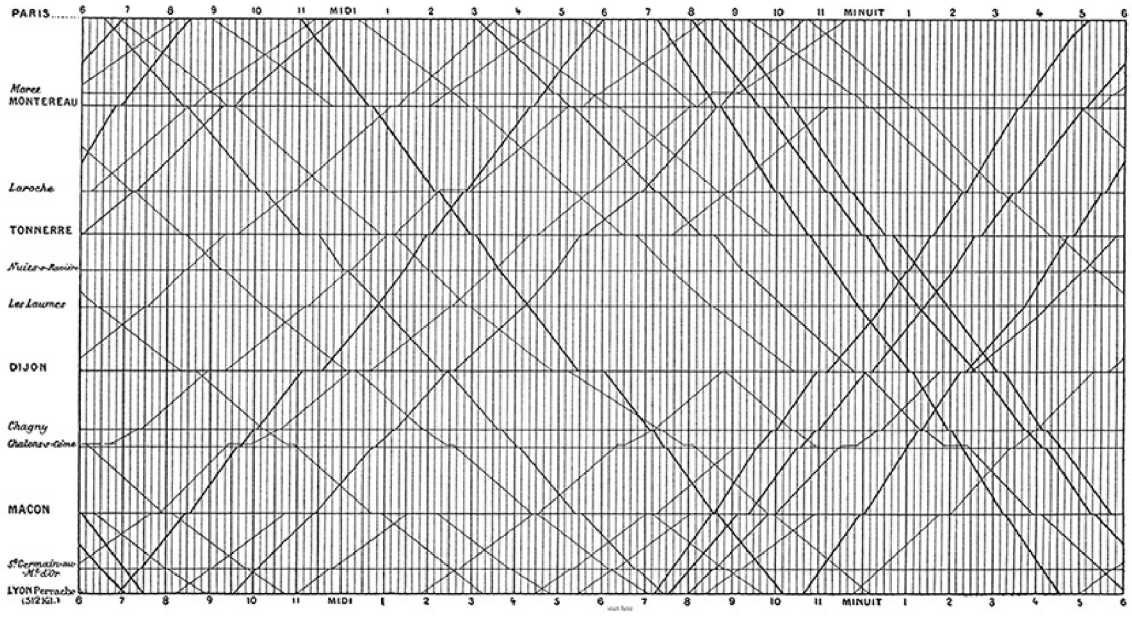
V této kapitole bude stručně popsána historie grafikonu vlakové dopravy, z čeho se tento graf skládá a jeho současné využití v České republice. Také bude uvedena motivace pro jeho použití v aplikaci.

2.1 Popis

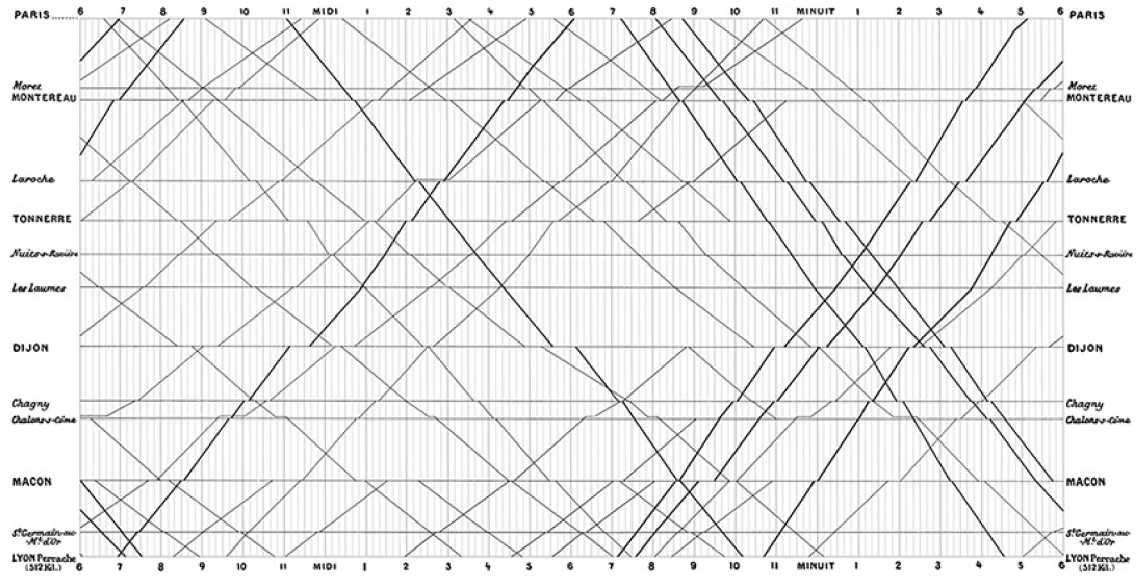
Grafikon je grafickou formou jízdních řádů, označované také jako nákresný jízdní řád. Základem je graf kartézské soustavy souřadnic s horizontální časovou osou a vertikální značící jednotlivé stanice s odstupy odpovídajícími reálným vzdálenostem mezi nimi. Pomocí lomených čar popisuje grafikon pohyb vlaků po trase. Čím je tato čára svislejší, tím rychlejší vlak je. Stání vozidel ve stanicích se projeví vodorovnou čárou.

2.2 Historie

Nejznámější první zmínkou o vlakovém grafikonu je graf na obrázku 2.1. Tento graf znázorňoval trať z Paříže do Lyonu a byl publikován v knize francouzského vědce E. J. Marey publikované roku 1878. Tento design získal chválu a ohlas a americký statistik a profesor Edward Tufte jej dokonce umístil na titulní stranu své knihy [17] o grafech a tabulkách, ve které také publikoval vylepšení přehlednosti grafikonu z obrázku 2.1. Tento vylepšený grafikon je zobrazen na obrázku 2.2.

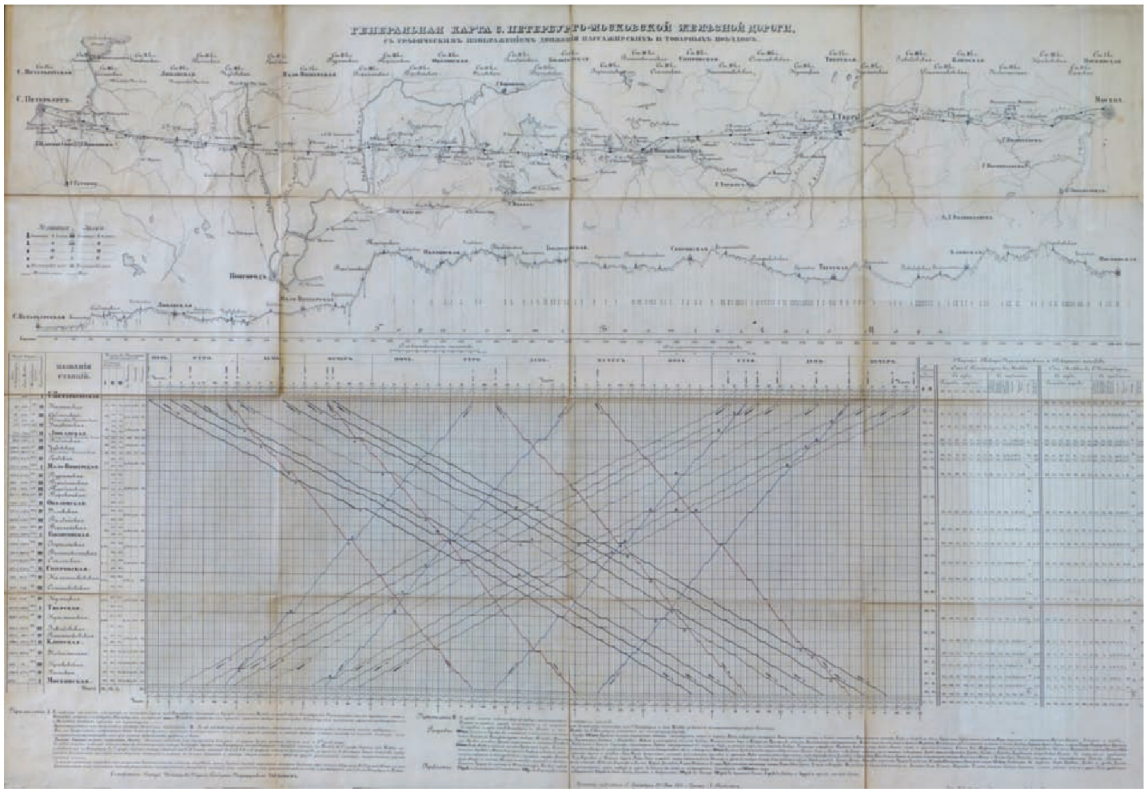


Obrázek 2.1: Étienne-Jules Marey, *La Méthode Graphique* (Paříž, 1878) [15]



Obrázek 2.2: Graf se šedým mřížkováním pro lepší přehlednost spojů [17]

Přestože je Marey považován za jeho autora, přisluhuje v jeho knížce [15] zásluhy panu Ibry, který jej navrhnul. Ale ani tento návrh nebyl první. Existují i dřívější použití z Ruska roku 1854 viditelné na obrázku 2.3, kdy jím pan Serjev znázornil trať mezi Moskvou a Petrohradem. Tufte zmínil v emailové korespondenci s autory článku [12] zkoumajícím historii grafikonu, že i jeho graf je natolik propracovaný, že musely existovat ještě dřívější použití, pravděpodobně použity při interním plánování železničních jízdních řádů.

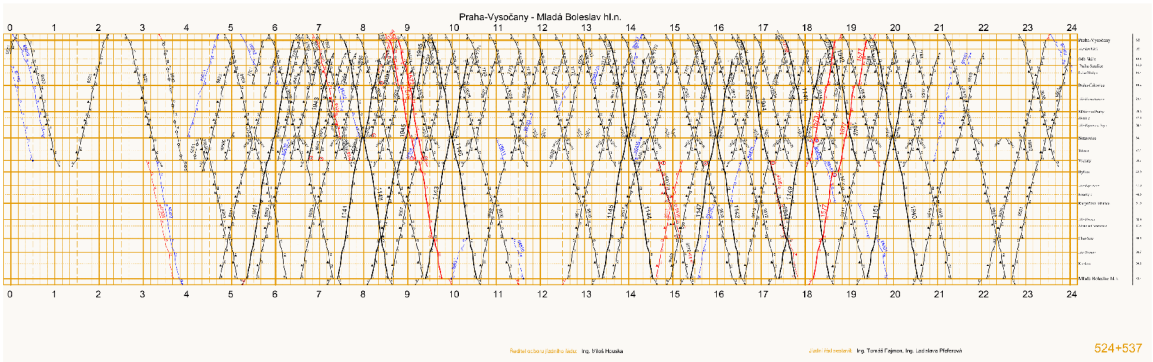


Obrázek 2.3: Serjev (Petrohrad, 1854) [12]

2.3 Současné využití

Nákresné jízdní řády jsou používány nejen při plánování vlakové dopravy a samotné tvorbě jízdních řádů, ale denně i při řízení a kontrole dopravy. Hlavními uživateli, kteří pracují s grafikony jsou zaměstnanci dopravních podniků – dispečeři, strojvedoucí, a výpravčí.

Tyto jízdní řády pro tratě České a Slovenské republiky jsou vydávány odborem jízdního řádu a zveřejňovány webovou službou gvd.cz. Příklad takto zveřejněného jízdního řádu je na obrázku 2.4.



Obrázek 2.4: Nákresný jízdní řád pro trať Praha-Vysočany – Mladá Boleslav hl.n.

2.4 Výhody pro použití v aplikaci

V grafikonu se zobrazují všechny vlaky, které po dané trase v daný den jezdí. Všechny informace o typu vlaků, rychlosti a počtu zastávek jsou ihned dostupné v grafu kartézské soustavy souřadnic. Uživatel nemusí procházet spoustu výsledků od klasických webových služeb, kdy by mohl nějaký vlak, který by pro něj byl optimální, přehlédnout.

Díky tomuto rychlému přehledu o veškerém provozu dokáže uživatel rozpoznat vhodné vlaky pro jeho cestování. Může porovnat rychlosti vlaků, jejich zastávky a stání v nich, a efektivně si naplánovat cestu včetně možných přestupů. Využití grafikonu přenechává naplánování cesty na uživateli, který jej může uzpůsobit právě jeho potřebám.

Například pokud by bylo pro uživatele vhodnější jet kousek cesty osobním vlakem a poté přesednout do rychlíku, v grafikonu by tuto možnost viděl ihned. Stejně jako případ, kdy by rychlejší vlak jel později a uživatel by je tak ve standardních vyhledávačích mohl přehlédnout. V grafikonu se navíc zobrazují i zpáteční spoje, čehož by uživatel mohl využít a provést pouze jedno vyhledání.

Kapitola 3

Data spojů vlakové dopravy

Data spojů veřejné dopravy by měly být podle vyhlášky č. 122/2014 Sb. [5] veřejně dostupné a ve formátu umožňujícím automatizované zpracování. Tato vyhláška se ale vztahuje pouze na dopravu autobusovou. Pro vlakové spoje existuje vyhláška 173/1995 Sb. [6], která se ale o formátu dat již nezmiňuje. Do dnešní doby tak nejsou tyto data dostupná ve formátu, který by umožňoval strojové zpracování, ale pouze grafické jízdní řády ve formátu PDF.

Tento způsob je velice nepraktický z několika důvodů. Skript pro stáhnutí těchto dat musí být specializovaný pro danou webovou službu. Veškerá jeho konfigurace závisí na aktuálním formátu webové stránky – na pořadí, hierarchii, identifikátorů a tříd HTML prvků. Pokud webová služba změní tento formát, je potřeba úprava skriptu, která by nemusela být zrovna jednoduchá.

3.1 Zdroje dat

Nejznámější webovou službou pro vyhledávání jízdních řádů je bezpochyby IDOS [13]. Provozuje jej firma CHAPS s.r.o., která zároveň shromažďuje tato data pro celou Českou republiku a jsou zodpovědní za jejich zvěřejnění. Data na této službě jsou tak vždy aktuální.

Druhou službou jsou Jízdní řády od společnosti Seznam, a.s., která je v současné době v beta verzi. Firma se potýkala se stejným problémem s dostupností dat jízdních řádů vlakové dopravy a při vyhledávání těchto spojů zobrazuje dokonce upozornění o možné neaktuálnosti dat, protože „český stát neplní svou zákonnou povinnost a nezveřejňuje aktuální data.“ [16]

Tyto webové služby poskytují data související s vlakovými spoji. Lze vyhledat buď spoje mezi zastávkami nebo přímo spoj pro daný vlak. V podrobnostech o spoji jsou poté informace o vlaku, všechny jeho zastávky s časy a ujetou vzdáleností na trase. Dále pak také informace o tom, v jaké dny vlak jezdí, nebo jaké jsou ve vozech služby.

Pro vlakový grafikon jsou vyžadovány data o zastávkách spoje a časech příjezdů a odjezdů pro tyto zastávky. Zastávka určuje vertikální souřadnice v grafu a časy horizontální. Pro vyhledávání spojů by se také hodila informace o tom, v jaké dny vlak jezdí, aby se zobrazovaly pouze vlaky, které v hledaný den skutečně jezdí. Ujetá vzdálenost vlaku se v grafu neprojeví, ale je možné ji zobrazit v podrobnostech o spoji.

Kapitola 4

Analýza současného informačního systému

Zakládám na existující práci [18], ze které vznikl informační systém pro interaktivní porovnávání vlakových spojů dostupný na jupw.cz. V této kapitole bude popsáno stručné shrnutí této práce a vytvořené aplikace.

4.1 Struktura aplikace a použité technologie

Informační systém se skládá ze dvou částí – přední a zadní.

Do přední části patří prvky, se kterými uživatel pracuje a které vidí. Tato část je tvořena sadou několika skriptů v jazyce JavaScript, které komunikují mezi sebou a se zadní částí aplikace. Grafikon zobrazující vlakové spoje je vykreslován s využitím knihovny Plotly, která je taktéž psána v jazyce JavaScript.

Zadní část pracuje na pozadí a tvoří jádro aplikace. Tvoří ji skripty v jazyce PHP, které zpracovávají uživatelské vstupy, provádí vyhledání spojů v databázi, a vrací data potřebná pro vykreslení grafu.

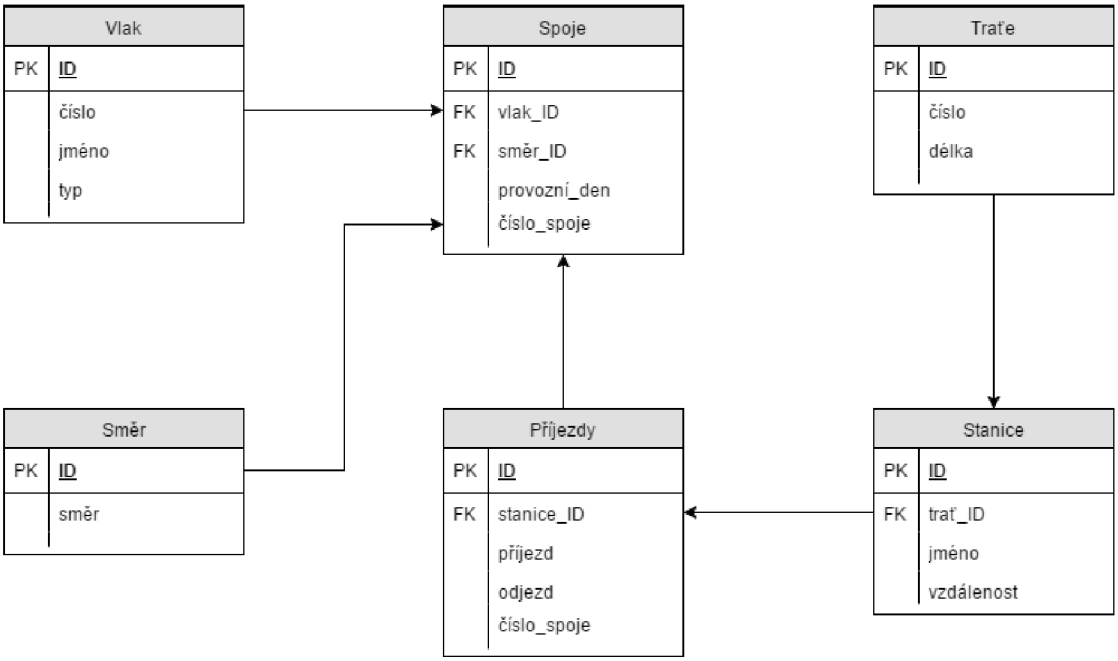
4.2 Databáze spojů

Aplikaci využívá klasickou relační databázi MySQL. Databáze obsahuje 6 tabulek a její ER diagram je možné vidět na obrázku 4.1. Autor ukládá data o jednotlivých tratích, jejich zastávkách a příjezdech na ně. Tyto příjezdy tvoří dohromady spoje daného vlaku v daném směru.

Autor optimalizoval databázi a zbavil se tak redundantních dat, které by vznikly při ukládání spojů zvlášť pro každý den. Vlaky totiž sdílejí jízdní řády podle typu dne – všední, víkendy a svátky. Existují i výjimky, kdy vlak nejede ve speciální dny, ale ty autor neřešil.

Graf, který zobrazuje vlakové spoje, má funkci pro současné zobrazení či filtrování směru jízdy tam a zpět. Kvůli této funkci musel autor přidat pro spoje atribut směru jízdy.

Pro naplnění databázi daty vytvořil autor skript v jazyce Python, který převádí data z textového souboru na SQL příkazy. K dispozici je pouze malý testovací vzorek spojů a není uvedeno jakým způsobem se data v textové podobě získala.



Obrázek 4.1: Schéma relační databáze informačního systému [18]

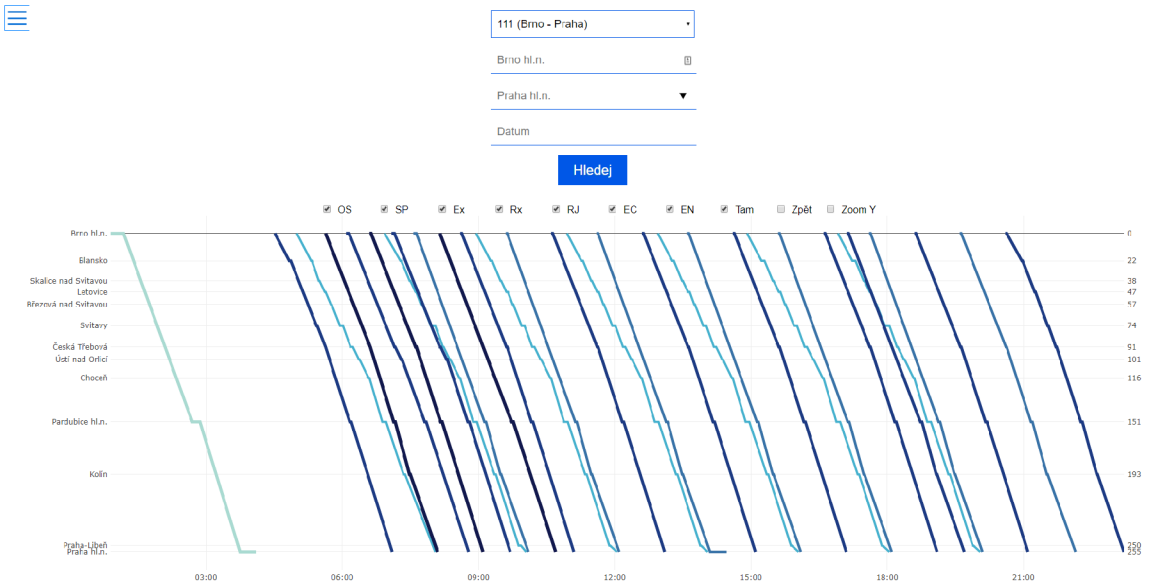
4.3 Uživatelské rozhraní

Uživatelské rozhraní viditelné na obrázku 4.2 je jednoduché a přehledné. Skládá se z vyhledávácího formuláře a interaktivního grafického prvku, který se nachází ve spodní části obrazovky, a ve kterém se vykresluje vlakový grafikon. Autor kladl důraz na jednoduché vyhledávání, takže grafické rozhraní neobsahuje žádné rušivé či matoucí prvky. Uživatel si nejprve vybere trať, na které chce spoje vyhledávat, a poté zadá počáteční a cílovou stanici a datum cesty. Pro lepší uživatelskou přívětivost jsou stanice napovídány z databáze spojů pro danou trať.

Po vyhledání se zobrazí nalezené vlakové spoje v grafu, který umožňuje filtraci zobrazovaných dat – rozlišení podle typu vlaku a směru jízdy. Graf se zobrazuje jako klasický vlakový grafikon s časovou horizontální osou a vertikální osou se stanicemi. Na levé straně grafu se zobrazují názvy zastávek a na pravé reálná vzdálenost na trati.

Graf taktéž podporuje ve výchozím nastavení přibližování a posun po časové ose. Při zapnuté speciální volbě podporuje taktéž přiblížení a posun po vertikální ose se stanicemi. Vyhledané spoje se zobrazují pouze pro vyhledaný den a posuvem na další se žádné spoje již nedovyhledají.

Menu v levé části obrazovky umožňuje sdílet konfiguraci vyhledávacího formuláře včetně nastavení filtrace dat a obnovení stránky do původní konfigurace.



Obrázek 4.2: Uživatelské rozhraní informačního systému [18]

4.4 Nedostatky

Největším nedostatkem je nutnost vybrat si trať, což znamená, že aplikace nepodporuje vyhledání spoje s přestupem. Uživatel si tak může vyhledat a zobrazit pouze provoz na jedné trati.

Z hlediska uživatelského rozhraní je nešťastný způsob zobrazování doplňujících informací o vlaku. Okénko s těmito informacemi se totiž zobrazuje pro všechny spoje, které protínají vertikální osu pozice kurzoru, a ne jen ten, na který ukazuje kurzor přesně. Při větším počtu spojů je tak graf nepřehledný.

Dalším nedostatkem je obsah databáze vlakových spojů, která obsahuje pouze 2 tratě a spoje mezi nimi pro 1 den. Aplikace v daném stavu ale není určena pro produkční nasazení.

Aplikaci chybí responzivní design a tak není dobře použitelná na menších zařízeních, zejména pak na mobilních telefonech. To je v dnešní době velký nedostatek, jelikož právě mobilní zařízení jsou čím dál častějším způsobem přístupu na internet.

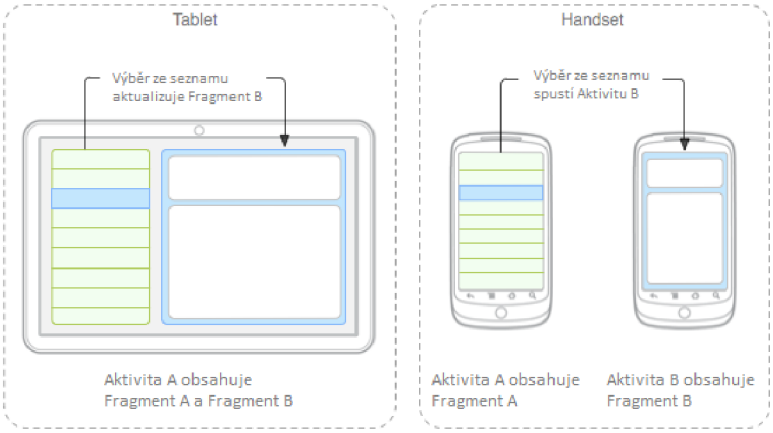
Kapitola 5

Teorie návrhu grafického rozhraní pro tablety

Vyvíjení grafického rozhraní pro tablety s sebou přináší značná úskalí a změny oproti jiným zařízením.

V porovnání s počítači jde o zmenšení zobrazované plochy a hlavně změnu ovládacího zařízení. Oproti myši s kurzorem, který přesný, zde uživatel používá prsty, které při dotyku s obrazovkou zaberou mnohem větší plochu. K tomu musí být přizpůsobena velikost ovládacích prvků jako jsou odkazy, tlačítka, ale také velikost písma, které by mohlo být špatně čitelné.

V porovnání s mobilními telefony, pro které jsou hlavně cíleny mobilní aplikace, znamená větší obrazovka a rozlišení více prostoru pro aktivity. Vyskytuje se zde rozložení do více panelů jak je vidět na obrázku 5.1. Obrazovky vyvinuty pro mobilní telefony se mohou sjednotit do jedné v podobě panelů. Obrazovka je tak logicky rozdělena například na menu a obsah.



Obrázek 5.1: Ukázka použití více panelů

Samotné tablety mohou mít velice rozdílnou nejen velikost a rozlišení obrazovek, ale také počet pixelů na palec (DPI). Při vývoji pro Android je proto k dispozici několik velikostí,

které se dají implementovat a navrhnout pro ně specializované rozložení grafických prvků a také několik verzí ikonek a obrázků.

Při navrhování vzhledu textu je nutné dát pozor na velikost písma. Malé velikosti, které jsou běžné na stolním počítači by mohly být špatně čitelné. A vzhledem k nutnosti větších ovládacích prvků by to mohlo vypadat nepěkně. Zároveň je důležité dbát na šířku prvku, ve kterém se zobrazuje text, abychom se vyvarovali stavům na obrázku 5.2. Při velké šířce vznikají prázdná místa, která zabírají drahocenné místo. Hodně široké řádky jsou také obtížné na čtení a uživatel by se tak mohl mezi řádky ztratit.

Obrázek 5.2: Ukázka přílíš širokých textových prvků

Při navrhování grafického rozhraní je také důležité si uvědomit, jak bude uživatel zařízení používat, respektive jak jej bude držet. Časté a bezpečné uchopení 2 rukama jako na obrázku 5.3 nechává jen málo prostoru kam prsty dosáhneme. Je proto dobré mít ovládací prvky po stranách, kde budou v dosahu.

Obrázek 5.3: Příklad držení tabletu 2 rukama

Kapitola 6

Analýza požadavků aplikace

Hlavním cílem je poskytnout uživateli přehled o veškerém provozu na vlakových tratích pomocí nákresného jízdního řádu. Pro přehled o tomto provozu se hodí využít nákresný jízdní řád – grafikon vlakové dopravy. V současné době navíc neexistuje mobilní aplikace, která by jej využívala. V této kapitole budou definovány požadavky, za kterých se má tohoto cíle dosáhnout. Zároveň budou popsány nejpravděpodobnější cíloví uživatelé aplikace a co to pro aplikaci znamená.

6.1 Požadavky na aplikaci

Uživatelé si budou chtít vyhledat optimální vlakové spoje mezi stanicemi, z čehož vyplývají následující požadavky:

- vyhledání tras a spojů i s přestupy,
- filtrace spojů podle typů vozů,
- pochopitelný vlakový grafikon,
- intuitivní zobrazení detailních informacích o spoji vč. časového řádu.

Vzhledem k aplikační platformě bude problémům čelit zejména grafické rozhraní aplikace. Tablety jsou ideálním zobrazovacím rozhraním vzhledem k dostatečné velikosti obrazovky a možnosti ovládat graf prsty. Nicméně aplikace bude také cílena na uživatele mobilních telefonů, pro které se bude muset GUI optimalizovat. Z vývojářského hlediska bude potřeba zajistit následující požadavky:

- dostatečný počet grafických uspořádání podle velikosti obrazovky pro optimální zobrazení,
- rychlé vyhledání spojů v databázi včetně přestupních míst,
- dostatečná a aktualizovaná databáze vlakových spojů České republiky.

Aby byla aplikace plynulá a rychlá i na starších zařízeních, je nutné navržení vlastní databáze spojů a optimalizace nejen vyhledávání spojů, ale i vykreslování grafu a grafických přechodů na přední části aplikace viditelné uživateli.

6.2 Grafické uživatelské rozhraní

Aby uživatel aplikaci nejen vyzkoušel, ale později se k ní i vracel, nestačí aby byla jen funkční. Velký důraz je také kladen na grafickou stránku. Aplikace bude vyhovovat standardům Material Design [10] určenou pro operační systém Android, což zaručí, že uživatelé budou ovládací prvky znát a nebudou v aplikaci ztraceni.

První část uživatelského rozhraní se bude skládat ze vstupních prvků pro zadání počátku a cíle trasy, a výběr datumu. Vyhledávací prvky budou napovídat možné stanice z databáze a prvek pro datum bude mít vyskakovací kalendář pro jednoduché a rychlé zvolení požadovaného dne.

Druhou, hlavní, částí je vizualizace vlakového grafikonu, který bude zobrazovat vyhledaná data. Graf bude umožňovat filtrování spojů podle různých kriterií, např. podle typu vlaku. Bude se zobrazovat především tzv. landscape módu, kdy je displej položený na šířku. Ta je u grafikonu důležitější než výška, protože čím větší jsou rozestupy na časové ose, tím bude graf lépe čitelný. Tento pohled bude doporučen, ale nebude vyžadován. Graf se bude zobrazovat přes celou obrazovku, kdy počáteční vyhledávací formulář zmizí a bude jej možné zobrazit tlačítkem zpět.

Graf půjde také přibližovat a oddalovat. Budou se zobrazovat spoje na daný den s možností přesahu do následujícího dne. Po kliknutí na spoj, bude možné zobrazit doplňující informace, včetně všech stanic a časů.

6.3 Databáze vlakových spojů

Pro dobrou použitelnost aplikace je zapotřebí mít dostatek dat vlakových spojů – ty je nutné získat vlastním skriptem, který bude popsán v kapitole 9.2.

Takto získaná data se přetransformují do vhodného formátu a uloží do databáze aplikace. Databáze tedy bude součástí instalace aplikace a nebude nutnost mít pro vyhledání spoje internetové připojení. S tím bohužel přichází problém (ne)aktuálnosti dat.

6.4 Problém aktuálnosti dat

Při hledání spojů je nutné, aby data byla aktuální, jinak by hledání nemělo smysl a mohlo by způsobit nepříjemné problémy. Abychom předešli těmto problémům a uživatelé mohli aplikaci používat, bude zapotřebí vydávat aktualizace aplikace s aktualizovanými daty. Databáze bude obsahovat veškeré data, která budou z vybraných webových služeb volně dostupná a která se podaří pomocí skriptu získat.

6.5 Cíloví uživatelé

Aplikace je určená pro širokou veřejnost, kteří mají zájem vyhledávat a porovnávat vlakové spoje. Zaměřením na tablety se tato cílová skupina zužuje, ale aplikace bude použitelná i na mobilních telefonech. Pro zařízení s menšími obrazovkami se například skryje filtr a bude zobrazitelný pomocí tlačítka. Tak vznikne více prostoru pro zobrazení potřebných dat. V dnešní době mají mobilní telefony navíc běžně přítomen větší displej, na kterém se vlakový grafikon bude bez problémů zobrazovat.

Jelikož je aplikace určena pro zařízení s různou velikostí a orientací displeje, bude nutné udělat návrhy a zpracovat grafické pohledy pro všechny kombinace pro zajištění optimálního zobrazení pro všechny cílové uživatele.

Kapitola 7

Analýza možných technologií

V této kapitole se podíváme na zajímavé knihovny, které by se daly v aplikaci využít. Řekneme si důvody proč je použít s malou ukázkou kódu.

Aplikace bude psána v programovacím jazyce Java, který je dominantní ve vývoji aplikací pro Android.

7.1 Práce s aplikačními daty

Android nabízí několik tříd a knihoven, které slouží pro konzistenci dat napříč celou aplikací.

7.1.1 Data binding

Data binding je oficiální knihovnou společnosti Google umožňující propojení grafických komponent přední části aplikace definovaných v souborech formátu XML a datových objektů ze souborů Java. Zajišťuje automatickou aktualizaci atributů a zobrazeného textu na stejnou hodnotu. Zároveň podporuje navázání funkcí na různé události uživatele – například kliknutí. Data binding ulehčuje spoustu práce, kdy programátor nemusí řešit poslání hodnoty z formuláře do funkcí a zajišťování konzistentního stavu.

Atributy z modelu se mohou v souborech formátu XML definujících vzhled komponenty odkazovat jednoduchým odkazem:

```
android:text="@{user.lastName}"
```

Podobně jako atributy se mohou také navazovat funkce na uživatelskou událost pomocí reference na metodu:

```
android:onClick="@{handlers::onClick}"
```

7.1.2 ViewModel

ViewModel [11] je třída navržená pro uchování všech dat spojených s uživatelským rozhraním a aplikační logikou. Má na starosti získání a uchování potřebných dat skrze všechny fragmenty v rámci aktivity. Pro aktivitu je sdílená jediná instance tohoto modelu, která je zachována i při přetváření aktivity a o data se tak nepřijde.

ViewModel také obstarává komunikaci mezi aktivitou nebo fragmentem a aplikační vrstvou aplikace, ale nikdy by neměl držet reference na samotný fragment nebo aktivitu. Ti mohou ViewModel sledovat a reagovat na změny dat v něm například prostřednictvím LiveData.

7.1.3 LiveData

LiveData [9] je třída, která se chová jako schránka pro ostatní objekty. Umožňuje pozorování změny uchovaného objektu – využitelné například pro aktualizaci dat grafického prostředí při změně zdrojových dat. Dá se také využít pro návratovou hodnotu dat z databáze, jelikož přímé volání databáze na hlavním vlákně je zakázáno.

7.2 Práce s databází

Android nabízí relační databázi SQLite pro uložení aplikačních dat. Pro práci s databází je nutné buď s ní obtížně pracovat na nízké úrovni nebo použít některou implementaci objektově relačního zobrazení (ORM – Object-relational mapping).

ORM umožňuje vytvoření tříd, tzv. entit, které reprezentují pohled na databázi, a práci s nimi pro ukládání a výběr dat z databáze. Tato technika zjednodušuje vývojáři práci, který může tak pracovat přímo s objekty daného programovacího jazyka - v našem případě Javy.

Dále budou popsány možné knihovny ORM pro použití v aplikaci.

7.2.1 Room

Room je knihovna vyvinutá společností Google. Základem knihovny jsou anotace, které se používají v objektech jazyka Java. Služba DAO (Data Access Object) provádí nad databází naše dotazy a vrací z ní celé objekty nebo jen jejich atributy. Room poskytuje potřebné minimum pro pohodlnou práci s databázemi.

Entity se označují anotací `@Entity` a jejich jednotlivé atributy, které reprezentují sloupec v databázi anotacemi `@PrimaryKey` pro ID a `@ColumnInfo(name="column_name")` pro vše ostatní.

Room nepodporuje v entitách asociace, kdy by při spojení tabulek cizími klíčemi mohly existovat zanořené entity – například železniční trať by mohla obsahovat vlaky, které po ní jezdí. Toto by šlo vyřešit komplexním SQL dotazem, který nám vrátí co potřebujeme, ale musíme se dotazovat přesně na věci, které potřebujeme.

DAO, označené anotací `@Dao`, obsahuje metody reprezentující databázové dotazy. V případě výběru dat z databáze musíme napsat vlastní dotaz, kterým metodu anotujeme. Například `@Query("SELECT * FROM user WHERE uid IN (:userIds)")`.

7.2.2 Realm

Realm je databáze postavená od základu a optimalizovaná pro mobilní zařízení. Je psán v C++ a běží přímo na hardware, takže je velice rychlý a mnohdy i rychlejší než nativní SQLite.

O spoustu věcí se Realm stará automaticky, což s sebou ale přináší i značné nevýhody - například nemožnost definování metod ve třídách modelů. Na druhou stranu má Realm poměrně rozsáhlou dokumentaci, což je určitě velká výhoda. Projekt jako takový je open-source.

7.2.3 OrmLite

OrmLite je obecná ORM knihovna pro Javu, ne jen pro Android. Nabízí standardní funkce bez zbytečností, takže má malou velikost.

Mimo jiné poskytuje jednoduché metody pro podmíněný výběr z databáze v kódu, což by se hodilo při vyhledávání vlakových spojů. Projekt je taktéž open-source a dokumentace je i přes velice jednoduchý vzhled poměrně obsáhlá.

Kapitola 8

Návrh aplikace

V této kapitole bude popsán návrh aplikace - jak bude vypadat grafické prostředí, celková architektura a databáze, co se použije za technologie, a jak se bude vykreslovat samotný graf spojů.

8.1 Architektura aplikace

Celkový kód a architekturu aplikace jsem rozdělil do 2 hlavních částí na:

1. grafické uživatelské rozhraní (GUI) a
2. aplikační logiku.

Tento proces oddělení zodpovědností umožní nemíchat do sebe rozdílné věci a zajistí jednoduché rozšiřování či úpravy aplikace v budoucnu.

8.1.1 Jádro

Základní stavební kameny Android aplikací spočívají v aktivitách (*activity*) a fragmentech (*fragment*), které slouží ke spojení operačního systému a naší aplikace.

Aktivita je vizuální komponentou, která popisuje jedno okno aplikace, většinou zaplňující celou obrazovku. Uživatel interaguje prostřednictvím aktivity s aplikací.

Jednotlivé logické celky na obrazovce se mohou dále dělit na fragmenty. Ty zlepšují modularitu a znovupoužitelnost komponent. Mohou popisovat jak část grafického rozhraní jako například vyhledávací formulář, tak jen nějakou operaci bez grafické komponenty.

Tyto 2 prvky mají určitý životní cyklus a operační systém je může nejen při neaktivitě ukončit, což by způsobilu ztrátu dat v nich uložených. Příkladem ukončení může být přetváření aktivity z důvodu otočení obrazovky. Proto by měly obsahovat pouze kód spojený s prací s grafickým rozhraním nebo přímo operačním systémem.

Pro aplikaci jsem navrhl pouze 1 aktivitu, která obstará integraci s uživatelem a bude zobrazovat různé fragmenty. První fragment bude obsahovat vyhledávací formulář a po jeho vyplnění a odeslání se tento fragment skryje a spustí druhý s vykresleným vlakovým grafikonem. Operace spuštění fragmentů se ukládají do zásobníku, který umožňuje se v aplikaci vrátit použitím tlačítka zpět, které anuluje poslední akci.

8.1.2 Práce s daty

Data využívaná v grafickém rozhraní by měly být uloženy v **modelu**, který umožňuje perzistenci dat, aby v případě ukončení aplikace či změny orientace obrazovky uživatel o tyto data nepřišel. Model je komponenta starající se o data, která je nezávislá na pohledu a grafických komponent.

ViewModel popsaný v kapitole 7.1.2 je implementací tohoto přístupu. Stará se o komunikaci s **aplikační vrstvou**, která může zajišťovat hlavní logiku aplikace a dodávat data do modelu. Model je tak oddělený od pohledu a není ovlivněn konfiguračními změnami jako je například zmíněné přetváření při rotaci obrazovky a o data v něm tak nepřijdeme.

8.1.3 Aplikační logika

Veškerá logika aplikace bude uložena v aplikační vrstvě, kterou tvoří tzv. **služby**. Ty budou v našem případě zajišťovat vyhledání dostupných stanic nebo vlakových spojů. Vyhledávací služba převezme data vyhledávacího formuláře z modelu a provede vyhledání možných cest mezi stanicemi a poskládání jednotlivých částí tratí dohromady. Samotná práce s databází bude v **repozitáři**, která nad ní bude volat dotazy, případně spojovat více dotazů do jedné odpovědi.

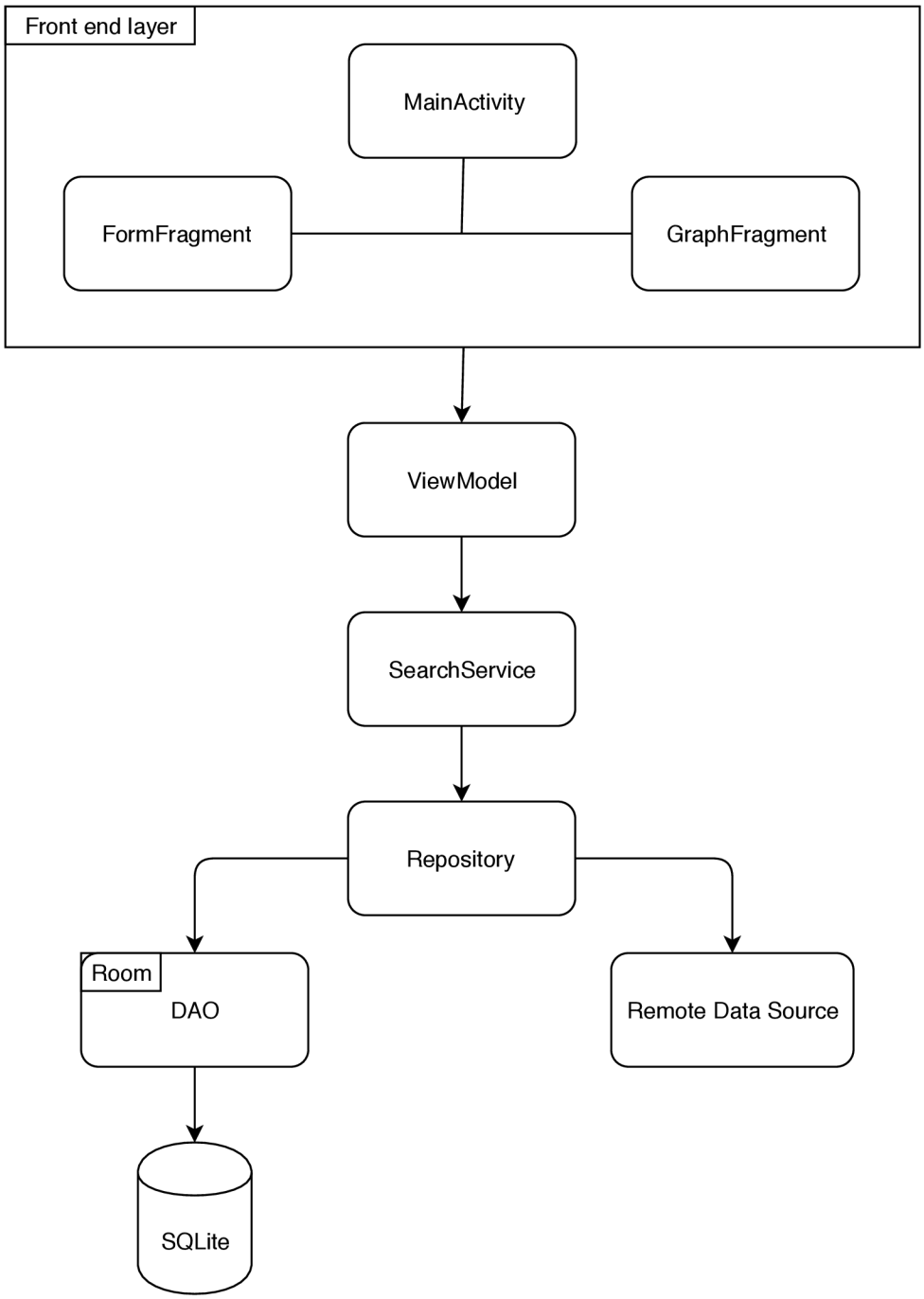
8.1.4 Diagram vrstev aplikace

Diagram 8.1 ukazuje návrh této architektury aplikace s tím, jak mezi sebou jednotlivé vrstvy komunikují.

1. vrstva – aktivity a fragmenty
    - Tato vrstva zobrazuje a obsluhuje grafické komponenty, skrz které uživatel interaguje s aplikací.
    - FormFragment obsahuje vyhledávací formulář a GraphFragment vlakový grafikon s jeho filtrem.
    - Spouštění těchto fragmentů zajišťuje MainActivity.
2. vrstva – ViewModel
    - Model v této vrstvě obsahuje veškerá data, se kterými se v aplikaci pracuje.
    - Zajišťuje, že během přetváření aktivity se o tato data nepřijde.
    - Volá službu, která provede vyhledání vlakové trasy a spojů a uloží tyto data zpět do modelu.
3. vrstva – SearchService
    - Služba, která provádí vyhledání dat pro zobrazení v grafikonu.
    - Po vyhledání převede data z databáze na objekty vhodné pro vykreslování.
    - Celá služba pracuje v novém vlákně, aby se nezatěžovalo hlavní vlákno vykreslující aplikaci.
4. vrstva – Repository
    - Repozitář, který zprostředkovává volání do databáze.

- V případě potřeby uskutečnění většího počtu dotazů, je sloučí a vrátí pouze jednu odpověď.

5. vrstva DAO a Remote Data Source
   - DAO obsahuje dotazy v jazyku SQL, které se provádějí na databází.
   - Remote Data Source ukazuje možnost webového serveru, pomocí kterého by se prováděla aktualizace dat v databázi.



Obrázek 8.1: Návrh architektury aplikace

8.2 Grafické prostředí

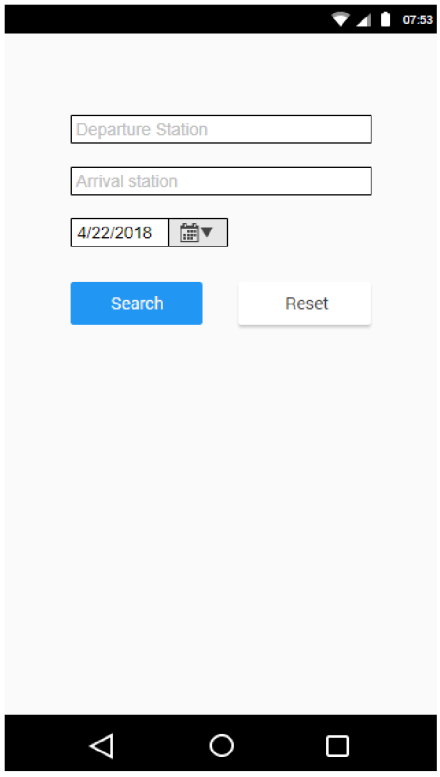
Základní deklarace jednotlivých grafických komponent Android aplikací se nachází v souborech formátu XML popisujících aktivity a fragmenty. Je zde sice možnost komponenty vytvářet za běhu programu v kódu aplikace, ale není ideální z několika důvodů. Hlavním je oddělení deklarace grafického prostředí a kódu aplikace pro lepší přehlednost a možnost snadnějšího nalezení případné chyby. U souborů formátu XML lze také definovat více variant uspořádání podle velikost obrazovky, způsobu orientace nebo i pro různé jazyky.

8.2.1 Návrh

Vzhledem k rozdílným velikostem a orientacím obrazovek u mobilních telefonů a tabletů, je potřeba vytvořit odpovídající počet návrhů a souborů formátu XML popisující vzhled pro daný případ.

Vyhledávací formulář

Základní uživatelské prvky tvoří formulář, jehož návrh je viditelný na obrázku 8.2, do kterého uživatel vyplní parametry hledání vlakových spojů. Formulář obsahuje textové vstupy pro počáteční a cílovou stanici, které budou napovídat dostupné stanice z databáze pro ulehčení výběru, a dále vstup pro čas a datum odjezdu, který po kliknutí otevře dialogové okno obsahující kalendář a hodiny. Po potvrzení formuláře se vyhledají vlakové spoje a obrazovka přejde do zobrazení grafu vlakového grafikonu.



Obrázek 8.2: Návrh vyhledávacího formuláře na mobilním zařízení

Vlakový grafikon

Výstup hledání spojů se bude zobrazovat v grafikonu. Ten bude potřeba vykreslit a vypočítat celý ručně bez použití jakékoliv knihovny, protože žádná použitelná neexistuje.

Graf bude skládat ze 2 popisujících os:

1. osa X znázorňující čas a
2. osa Y, na které budou zobrazeny jednotlivé zastávky vlaků.

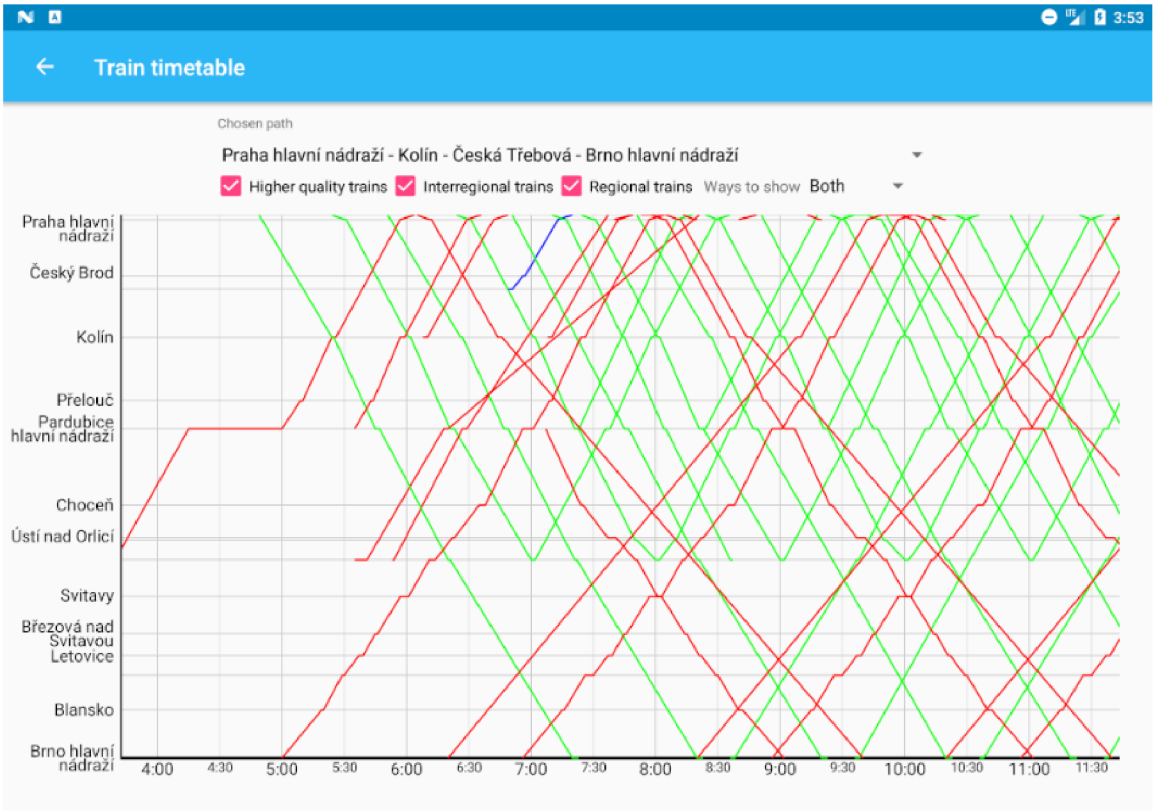
Na ose X se bude zobrazovat čas ve formě minut, hodin, až dnů. Bude možné graf posouvat podél této osy v rámci jednoho dne, který se v grafu vykreslil.

Na ose Y se budou zobrazovat vlakové stanice v takovém poměru jako jsou vzdálenosti mezi nimi pro zachování funkce grafikonu znázorňující rychlost daného vlaku podle sklonu křivky. V případě několika stanic blízko sebe, kdy by došlo k překrývání, bude mít větší prioritu viditelnosti stanice s větším počtem zastavujících vlaků, případně se zobrazí jen jedna a ostatní stanice budou skryty až do přiblížení grafu.

Celkový graf bude možné přiblížit a oddálit pro lepší viditelnost a jednotlivé vlakové spoje budou barevně odlišeny podle typu vlaku barvami:

- zelenou barvou vlaky vysoké kvality (SC, EC, ...)
- červenou barvou meziregionální vlaky (R, ...)
- modrou barvou regionální vlaky (Os, SP, ...)

Zobrazené vlakové spoje bude možné označit, čím se zobrazí podrobné informace o vlaku spolu s časovým řádem. Na obrázku 8.3 je možné vidět vykreslený grafikon z aplikace.



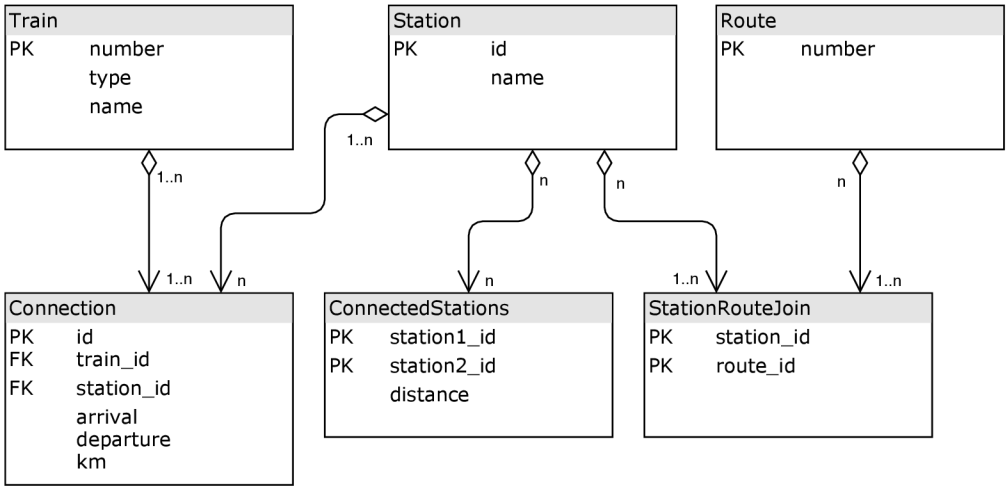
Obrázek 8.3: Obrazovka s vlakovým grafikonem vykreslená v aplikaci

8.3 Databáze vlakových spojů

Aplikace vyžaduje použití relační databáze, ze které se budou číst veškeré vyhledávané a zobrazované data. Uloženy budou všechny následující informace:

- tratě,
- stanice,
- vztahy mezi stanicí a tratěmi,
- propojené stanice,
- vlaky,
- spoje s příjezdy a odjezdy do stanic.

8.3.1 Struktura databáze



Obrázek 8.4: Schéma relační databáze

Na obrázku 8.4 je možné vidět navržené schéma databáze. Základními dvěmi tabulkami jsou `Route` a `Station`. `Route` popisuje železniční tratě jejími názvy – kromě 1 jsou to vždy čísla, a taktéž tabulka `Station` popisuje jednotlivé stanice jejími názvy.

Propojující tabulka `StationRouteJoin` určuje všechny tratě, na kterých daná stanice leží. Tato tabulka je důležitá pro zjištění, zda hledané stanice leží na stejné trati nebo k propojení různých tratí přes přestupní body.

Tabulka `ConnectedStations` určuje všechny páry stanic, které jsou spojené jakoukoliv tratí. Obsahuje také atribut `distance` znázorňující vzdálenost mezi těmito stanicemi. Tato tabulka se bude používat pro spojení dvou stanic na stejné trati a nalezení kompletní cesty mezi nimi včetně vzdáleností. Vzdálenosti se budou používat pro vykreslení stanic na vertikální osu grafikonu.

Posledními tabulkami jsou `Train` a `Connection`. `Train` obsahuje identifikační údaje vlaků – číslo, typ, název. Tyto údaje se budou zobrazovat v grafikonu v detailu spojení. Samotné spojení vlaku je uloženo v tabulce `Connection`, které obsahuje jednotlivé příjezdy a odjezdy do stanic včetně celkové vzdálenosti spoje, která se bude taktéž zobrazovat v detailu spoje.

8.4 Získání a zpracování databázových dat

Pro databázi potřebujeme získat data nejen o jednotlivých spojích, ale také vlacích, tratích, a všech jejich zastávkách. Neexistuje žádné veřejné rozhraní, kterému bychom mohli jen poslat dotaz a server by nám tyto data vrátil. Je potřeba analýza a následná extrakce dat ze zdrojového kódu webových stránek.

Data popisující tratě a jejich strukturu jsem získal z webové stránky www.zelpage.cz [4]. K extrakci těchto dat se použije skriptovací jazyk Python. V několika skriptech se získají identifikační údaje vlaků, čísla tratí a údaje o všech jejich zastávkách. Poté proběhne propojení těchto zastávek skrz tratě, aby došlo k vytvoření reálné mapě tratí.

Nakonec se pro nalezené vlaky vyhledají vlakové spoje. Pomocí typu, čísla a jména vlaku dojde k vyhledání na některé z věřejně dostupných webových služeb a z detailu spoje se získají informace o spoji.

8.5 Vykreslování grafů spojů

Vlakový grafikon je klíčovým prvek celé aplikace. Jeho vykreslování je tak velice důležitou a netriviální částí implementace.

Samotný graf je potřeba rozdělit do spousty základních tvarů nebo textů, které je možné vytvořit a vykreslit. Jednotlivé vlakové spoje musí být možné zobrazit a skrýt na základě filtru. Graf bude muset také podporovat posouvání a změnu přiblížení, což dále tento problém prohlubuje.

Celkový graf jsem rozložil na tyto body:

- plátno podporující posuv a přiblížení, na které se bude kreslit,
- osa X obsahující časové jednotky, které zároveň budou sloužit jako souřadnice,
- osa Y obsahující stanice – jejich souřadnice určí kilometrová vzdálenost stanic; zároveň bude podporovat skrytí překrývajících se textů,
- vlakové spoje, které budou tvořeny úsečkami.

Kapitola 9

Implementace aplikace

Aplikace byla implementována ve vývojovém prostředí Android Studio aktuální verze 3.1.2. Kód byl psán programovacím jazykem Java a jeho verze 8. V aplikaci ale nebylo možné použít všechny funkce, které byly přidány v této verzi vzhledem k nastavené minimální verzi systému Android, na kterém bude aplikace spustitelná. Pro kompatibilitu s až 95 % [8] zařízení byla vybrána jako nejnižší verze 4.4, které odpovídá verze API 19. Těmito verzemi se číslují jednotlivé vydání systému Android. Pro chybějící funkce – například třída `java.util.stream`, by byla potřeba verze API 24, což by znamenalo ztrátu 58.2 % všech zařízení Android, které nemají verzi 7.0.

9.1 Použitá technologie pro práce s databází

Pro práci s databází jsem vybral knihovnu Room, která poskytuje vše, co je pro tuto práci potřeba. Pro inicializaci databáze jsem využil knihovny **RoomAsset** [2], která obaluje knihovnu Room a dokáže vytvořit databázi ze souboru. Při prvotním spuštění se tak vytvoří databáze s počátečními daty vlakových tratí a spojů. Pro výběr dat z databáze byly napsány dotazy v jazyce SQL.

9.2 Skripty pro získání dat

V této sekci budou stručně popsány skripty, které byly potřeba pro získání veškerých dat o struktuře vlakových tratí a jízdních řádů vlaků.

9.2.1 Technologie

Python

Pro většinu skriptů jsem využil skriptovací jazyk Python za pomoci knihoven **requests**[1] pro stažení webové stránky a **lxml**[2] pro následné čtení a procházení zdrojového kódu.

Příklad načtení webové stránky a čtení zdrojového kódu:

```
page = requests.get(url) # získání webové stránky
tree = html.fromstring(page.content) # vytvoření stromu zdrojového kódu
rows = tree.xpath('//table/tr[@onmouseover]') # hledání elementů pomocí XPath
```

[1] Requests – http://docs.python-requests.org/en/master/
[2] lxml – http://lxml.de/

CasperJS

Pro provedení hledání vlakových spojů, kdy bylo zapotřebí vyplnit a odeslat formulář, byla využita technologie CasperJS[3], která staví na PhantomJS[4]. Oba nabízí webový prohlížeč bez grafického prostředí, který se dá ovládat pomocí jazyka JavaScript. Umožňují tak automatizaci a testování webových stránek. CasperJS navíc nabízí možnost vytvoření kroků, které se postupně zpracovávají.

Dokumentaci lze nalézt na stránkách CasperJS [1]. Příklady funkcí, které byly ve skriptu použity:

```
// vytvoření kroku pro jednotlivé prvky ze seznamu
casper.eachThen(trains, function (train) {});

// otevření web. stránky a navázání kroku po otevření
casper.open(url).then(function() {});

// vyplnění formuláře bez potvrzení
casper.fill('selector', {'inputName': data}, false);

// spuštění kódu JavaScript na otevřené web. stránce
casper.thenEvaluate(function() {});

// v~tomto případě se jedná o~odeslání formuláře
document.querySelector('input[type="submit"]').click();
```

9.2.2 Vlaky

O vlacích je potřeba získat jejich identifikační údaje, podle kterých se budou následně vyhledávat jízdní řády. Těmito údaji jsou – typ, číslo a jméno vlaku a lze je získat z následujících zdrojů. Já je získal z 2. zdroje.

1. zdroj – Wikipedia – Pojmenování vlaků ČD [19]
   - Nevhodný zdroj.
   - Všechny údaje jsou přehledně v jedné tabulce,
   - ale data jsou bohužel neúplná a neaktuální.
   - Tabulka obsahuje 447 vlaků, ale jízdní řády se povedlo získat jen pro 321 z nich.
2. zdroj – Zelpage – Řazení vlaků [3]
   - Jednotlivé vlaky jsou rozděleny do kategorií podle typů a následně ještě do podkategorií.
   - Z toho plyne složitější získávání dat, ale tyto data by měly být častěji aktualizované.
   - Tabulka navíc obsahuje 8606 vlaků a jízdní řády se povedlo získat pro 5118 z nich.

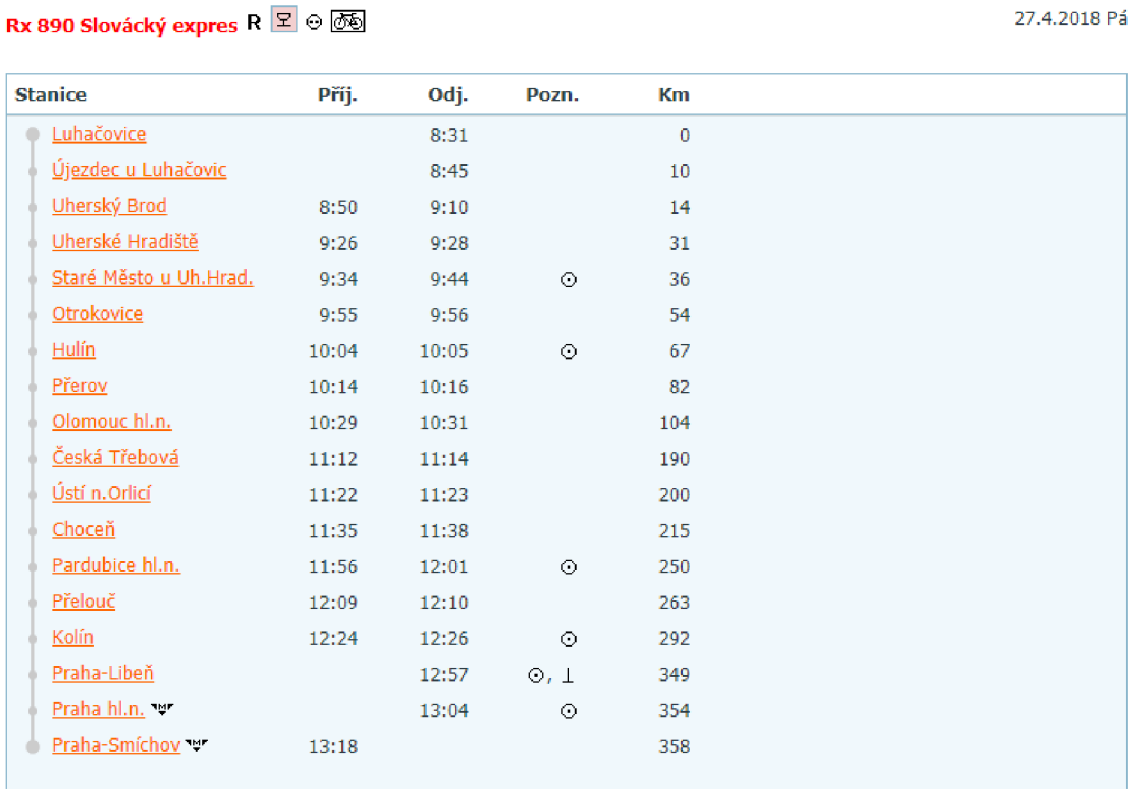
Výstupem těchto skriptů je textový soubor se seznamem vlaků v tomto formátu:

[3] CasperJS – http://casperjs.org/
[4] PhantomJS – http://phantomjs.org/

```
číslo;typ;název
70;rj;Gustav Mahler
```

9.2.3 Vlakové spojení

Pro všechny identifikované vlaky jsem nejprve provedl vyhledávání spoje na stránkách IDOS [14] pomocí technologie **CasperJS**. Ve skript se nejprve provede vyhledání potvrzením formuláře s typem, číslem a jménem vlaku. Z nalezených výsledků se ověří, zda první výsledek odpovídá zadaným údajům a pokud ano, uloží se odkaz směřující na podrobnosti spojení – příklad je na obrázku 9.1.



Obrázek 9.1: Podrobnosti vlakového spojení; zdroj IDOS

Tyto odkazy dále zpracovávám ve skriptu v jazyce Python, kde z podrobností získám údaje o jednotlivých zastávkách spojení – jméno stanice, časy příjezdu, odjezdu a ujetou vzdálenost.

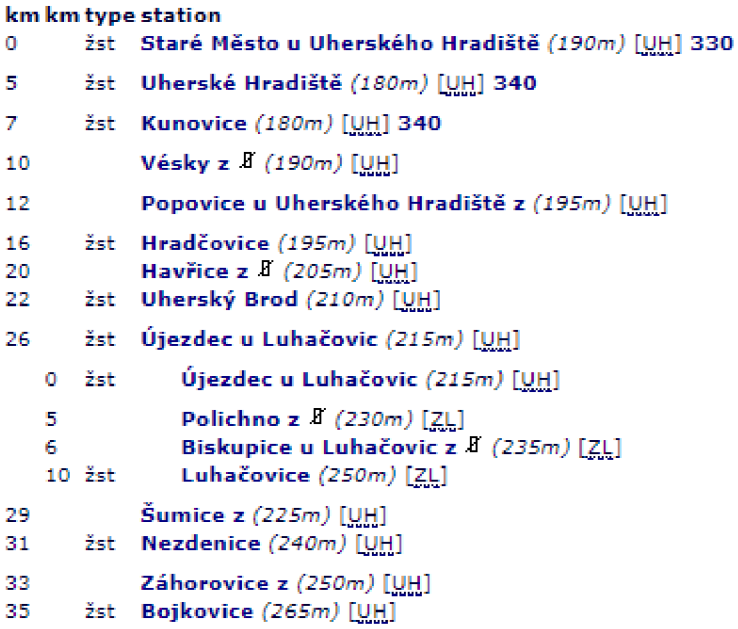
9.2.4 Tratě

Z webové stránky ŽelPage [4] jsem za pomoci skriptu získal názvy všech tratí v České republice a zároveň odkazy na podrobnosti o trati, které obsahují jednotlivé stanice, jejichž zpracování je popsáno v následující kapitole.

9.2.5 Zastávky

Posbírané odkazy na detaily tratí se postupně ve skriptu navštěvují a jsou z nich vytaženy údaje o zastávkách ležících na trati. Formát těchto dat je možné vidět na obrázku 9.2. Z tohoto detailu jsem získal informace o názvech zastávek a o tom, na jaké trati leží.

Na obrázku 9.2 je také možné vidět, že některé tratě se mohou větvit do odboček. Ve skriptu se toto zpracuje, aby na sebe zastávky správně navazovaly.



Obrázek 9.2: Zastávky ležící na dané trati; zdroj ŽelPage [4]

9.2.6 Propojení zastávek – vytvoření mapy tratí

Při získávání zastávek trati propojuji ve skriptu sousední stanice, z čehož vznikají grafy daných tratí. Všechny tyto zastávky zároveň ukládám do globálního seznamu, díky čemuž dojde k propojení všech tratí a vytvoření kompletního grafu železničních tratí a jejich zastávek.

Tento graf bude sloužit pro vyhledání přesné cesty a identifikaci všech zastávek mezi dvěmi zastávkami na stejné trati.

9.2.7 Mapování názvů zastávek tratí a spojů

Vzhledem k různým zdrojům zastávek a spojů došlo k rozdílům v názvech – např. Praha hlavní nádraží a Praha hl. n. Tyto rozdíly jsem musel identifikovat a vytvořit dvojice ve formátu `původní_název;požadovaný_název`.

K vyhledání stejných nebo nejlépe vyhovujícím názvům jsem použil knihovnu FuzzyWuzzy[5] pro jazyk Python. Knihovna využívá pro hledání podobných textových řetězců Levenštejnovu vzdálenost [7] a obsahuje několik různých metod porovnání. Pro co nejlepší přesnost jsem provedl vyhledání pomocí 3 metod pro maximálně 5 nejpřesnějších řetězců. Z těchto výsledků jsem vytvořil průměr a 2 nejlépe vyhovující jsem v požadovaném formátu

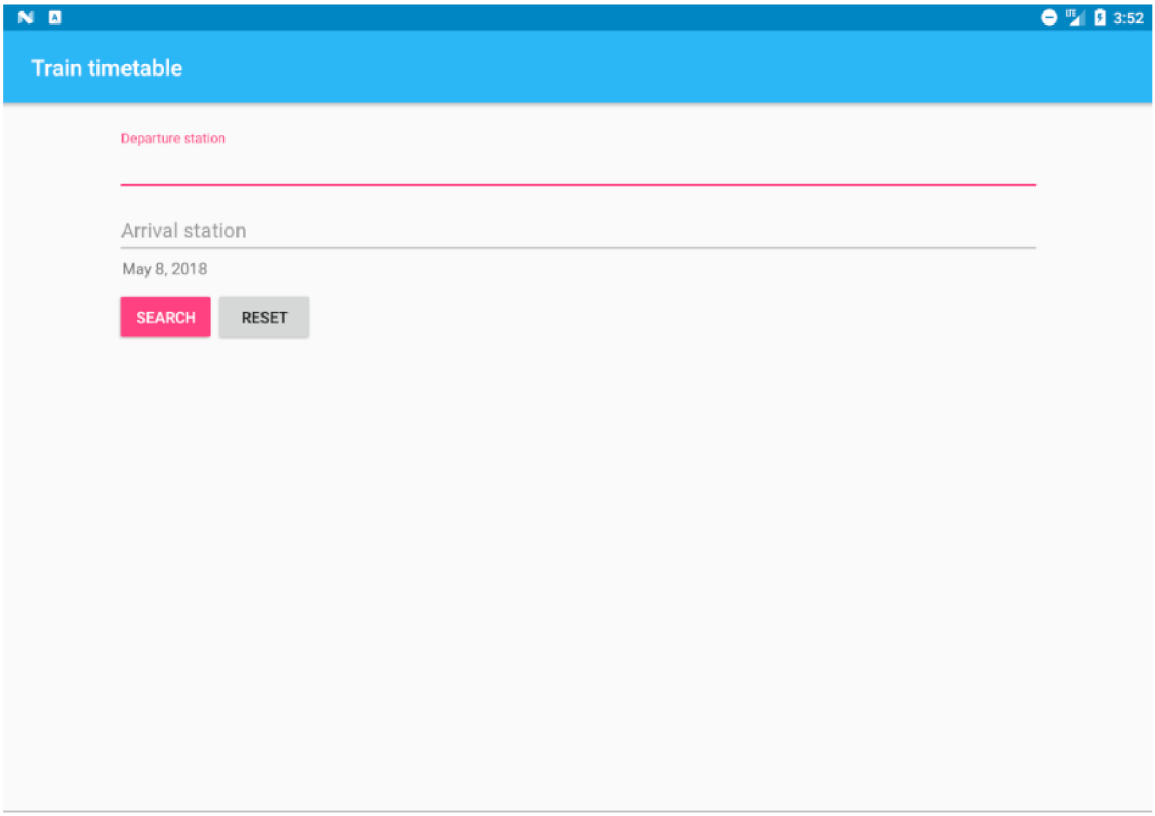
[5] FuzzyWuzzy – https://github.com/seatgeek/fuzzywuzzy

vypsal do souboru. Následně jsem ručně prošel všechny tyto výsledky a určil správný název. Ve většině případů byla hned první možnost ta správná. Zároveň jsem během tohoto mapování zjišťoval přebytečné znaky v požadovaném názvu a vytvořil druhý mapovací soubor pro zastávky získané z tratí.

9.3 Vyhledávací formulář

Vyhledávací formulář se skládá ze 2 textových polí pro počáteční a koncovou stanici, z výběru datumu cesty a dvou tlačítek pro potvrzení hledání a vymazání formuláře. Tlačítko pro výběr datumu otevře dialog obsahující kalendář pro výběr dne. Implementovaný formulář je možné vidět na obrázku 9.3.

Na obě textová pole pro stanice jsou navázány našeptávače dostupných stanic, ze kterých si uživatel musí stanici vybrat. Tím se zaručí, že zadané řetězce budou správné a do aplikace nám nepřijdou nevalidní vstupní hodnoty. Dostupné stanice jsou takové, ve kterých zastavuje nějaký vlak a zároveň mají sousední stanici.



Obrázek 9.3: Obrazovka s vyhledávacím formulářem vykreslená v aplikaci

9.4 Hledání možných cest

Po potvrzení formuláře je potřeba nejprve najít možné cesty složené z navazujících stanic. Toto hledání se liší podle toho, zda zadané stanice leží na stejné trati nebo je nutné mezi nimi najít nejprve přestupní body, na kterých se mění číslo tratě.

9.4.1 Stanice ležící na stejné trati

Pro vytvoření posloupnosti stanic mezi zadanými stanicemi, které leží na stejné trati, používám následující algoritmus:

1. Naleznou se všechy stanice náležící dané trati pomocí tabulky `StationRouteJoin` – z toho seznamu se budou vybírat vyhovující stanice.
2. Do seznamu možných cest se vloží počáteční stanice.
3. Poté probíhá následující cyklus dokud se nenarazí na koncovou stanici:
   (a) Ze seznamu možných cest se vybere 1. cesta a z této poslední zastávka, která se označí jako navštěvovaná.
   (b) Pomocí tabulky `ConnectedStations` se vyhledají sousední stanice pro tu navštěvovanou.
   (c) Ze sousedních stanic se odstraní ty, které nejsou obsaženy v seznamu vyhovujících stanic.
   (d) Pro každou z těchto vybraných sousedních stanic se vloží na konec seznamu možných cest kopie současné cesty s touto stanicí vloženou na její konec.
4. Výsledkem je lineární seznam jednotlivých zastávek nalezené cesty včetně vzdáleností všech stanic.

Pro optimalizaci a omezení opakovaného hledání cesty pro stejné úseky jsem implementoval cache, která v případě volání se stejnými parametry vrátí tento již uložený lineární seznam.

9.4.2 Stanice na různých tratích

V případě, že se stanice nenachází na stejné trati, naleznu nejprve posloupnost tratí mezi počáteční a koncovou. Tato část využívá algoritmus hledání do šířky.

1. Vytvoří se seznam možných tras a na začátek se vloží tratě náležící počáteční stanici.
2. Poté probíhá následující cyklus dokud se nenalezne dostatečný počet možných tras.
   (a) Ze seznamu možných tras se vybere 1. trasa a z této poslední trať, která se označí jako navštěvovaná.
   (b) Pomocí tabulky `StationRouteJoin` se naleznou sousední tratě, které mají nějakou společnou stanici se zadanou tratí.
   (c) Z těchto tratí se odstraní počáteční tratě a ty, které se v navštěvované trase již vyskytují.
   (d) Pokud je některá ze sousedních tratí obsažena v seznamu koncových tratí, uloží se tato trasa do seznamu nalezených tras.
   (e) Pro každou ze zbylých tras se vloží na konec seznamu možných tras kopie současné trasy s touto tratí vloženou na její konec.
   (f) Tento cyklus se ukončí pokud počet tratí v aktuálně nalezené trase přesahuje počet tratí v nejkratší trase o 2 tratě.

3. Nalezené trasy se seřadí a odstraní se možné duplikované cesty a vrátí se seznam lineárních seznamů jednotlivých tratí.

Pro takto nalezené trasy se najdou zastávky, na kterých se mění trať. Pokud je takových zastávek více, vybere se ta s nejvyšším počtem zastavujících vlaků. Názvy těchto zastávek se poté složí do názvu cesty, který bude využit v aplikaci pro výběr zobrazované cesty v grafikonu. Následně se jednotlivé úseky mezi zastávkami na stejné trati poskládají do posloupnosti zastávek s využitím algoritmu z předchozí kapitoly.

Vybírání nejlepších cest

Různé nalezené cesty poté filtruji, aby se vybraly nanejvýš 3, které jsou vybrány jako nejlepší. Filtrace vypadá následovně:

1. Výběr podle vzdálenosti:
   - U poskládaných cest se vede vzdálenost pro každou stanici – celková vzdálenost je tedy u poslední stanice.
   - Vyberu 3 nejkratší cesty bez ohledu na jejich vzdálenost a poté je možnost výběru až dalších 2 cest, pokud jejich vzdálenost nepřesahuje nejkratší cestu o více jak 20 %.
2. Výběr podle počtu vlakových spojů:
   - V tomto bodě vyhledám vlakové spoje a vyberu až 3 cesty s nejvyšším počtem nalezených spojů.

9.4.3 Zhodnocení složitosti algoritmů

Algoritmy pro nalezení kompletní cesty zastávek na stejné trati a posloupnosti tratí mezi zastávkami jsou časově nejnáročnější, protože provádí velké množství dotazů do databáze. Navíc v případě hledání tratí, kdy probíhá hledání do šířky bez jakýchkoliv hodnot tratí, se jich prohledává velké množství. Při nalezení cesty se algoritmus ani neukončuje, dokud nenalezne větší množství cest, aby mohlo dojít k porovnání jejich vzdáleností a počtu vlakových spojů. Algoritmus tak nalezne například více než 60 cest, než dojde k ukončení. Pro všechny tyto cesty probíhá následně vytvoření kompletní cesty dalším časově náročným algoritmem.

9.5 Hledání vlakových spojů

Vlakové spoje vyhledávám pro všechny stanice na nalezených cestách. Hledám se ty vlaky, které zastavují v alespoň 2 zastávkách na trase, aby se daly v grafikonu vykreslit. Zároveň ke každému spoji vyhledám názvy zastávek, protože v databázi jsou uloženy pouze identifikátory a relací tyto názvy nešlo získat v jednom kroku. Názvy těchto zastávek zobrazuji v podrobnostech o spoji.

9.6 Kreslící plátno

Veškeré osy, úsečky i textové údaje se vykreslují jako jednoduché objekty na kreslící plátno. Pro všechny tyto objekty musím vypočítat souřadnice, na kterých se mají vykreslit.

Android poskytuje ke kreslení 2 třídy - **Canvas** a **Paint**. Canvas ztvárňuje kreslící plátno a poskytuje metody pro vykreslení základních objektů jako například obdelník, kruh, úsečku, nebo jen text či barvu. Definuje tak tvary, které je možné vykreslit. Paint na druhou stranu definuje jak bude daný objekt vypadat. Určuje jeho barvu, styl písma a další parametry objektů. Tyto objekty se vykreslují v metodě `onDraw()` daného pohledu.

Obrazovka s vlakovým grafikonem jsem rozdělil do 2 kreslících pláten. Jedno úzké plátno na levé straně obrazovky obsahuje vertikální osu se stanicemi na trase. Rozdělení do 2 pláten bylo nutné z důvodu pevné pozice této vertikální osy pro zachování viditelnosti i v případě, kdy se s hlavním grafem pohybuje do stran. Druhé plátno zabírá většinu obrazovky a obsahuje vlakový grafikon. Toto plátno je široké jako trojnásobek šířky obrazovky pro přehledné zobrazení vlakových spojů. Grafikon se dá přibližovat a posouvat po obou osách.

9.6.1 Vykreslování grafikonu

Při kreslení grafikonu nejprve zjistím základní šířky pro časové jednotky dne a hodiny a poté vykresluji hlavní horizontální osu s popisky časových jednotek a pomocnými vertikálními přímkami skrz graf.

Z vykreslené vertikální osy jsem získal mapu stanic a jejich vertikálních souřadnic, které použiji pro kreslení zastávek vlaku ve stanicích. Horizontální souřadnice se dají vypočítat násobením času a základní šířky pro den a hodinu. Následně se v cyklu prochází vlakové spoje a tímto způsobem se úsečky vykreslují:

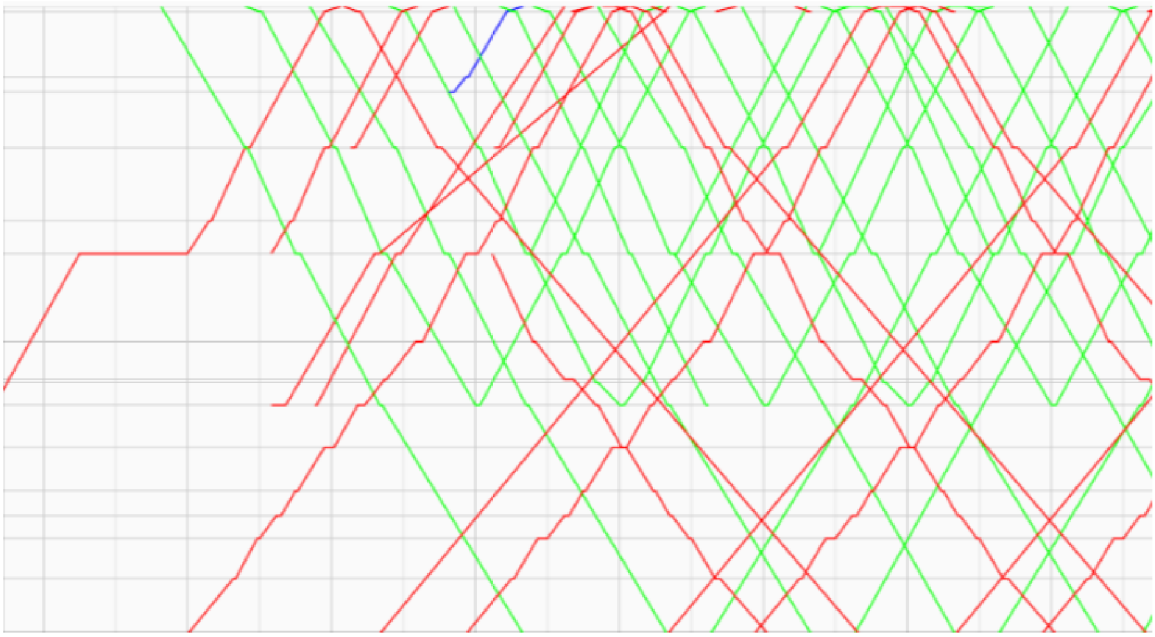
```
\\ výpočet souřadnice x
\\ parametry - čas příjezdu vlaku do stanice,
\\ základní šířka hodiny a případné posunutí o~den
float x = getTimeCoordinate(connection.getArrival(), hourWidth, addedDayWidth);

\\ souřadnice y se získá z~mapy vykreslených stanic
float y = selectedStationToYMap.get(connection.getStationId());

\\ vytvoření objektu a posunutí do počátečního bodu úsečky
Path path = new Path();
path.moveTo(x, y);

\\ dalšími posuny se již tvoří úsečka
path.lineTo(x, y);

\\ vykreslení vytvořené úsečky na plátno barvou odpovídající typu vlaku
canvas.drawPath(path, getPaintForTrain(train.getTrain().getType()));
```



Obrázek 9.4: Vykreslený grafikon

9.6.2 Osa stanic

Vertikální osu se stanicemi vykresluji na omezené místo na levé straně obrazovky. Vzhledem k malé šířce je potřeba delší názvy stanic vykreslit na více řádků. K tomu slouží třída `StaticLayout`, která text automaticky zalomí. Pro lepší orientaci v grafikonu vykresluji na souřadnicích stanic tenké horizontální úsečky.

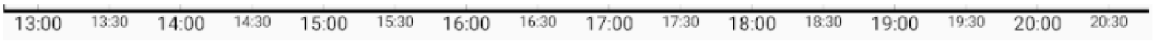
V případě, kdy stanice leží blízko sebe a došlo by k překrytí textů, vykresluji pouze text první a následující se přeskočí. Na obrázku 9.5 je možné vidět vykreslenou osu pro cestu z Prahy do Brna.

Obrázek 9.5: Vykreslená vertikální osa se stanicemi na trase

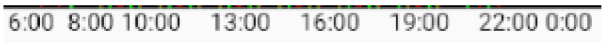
9.6.3 Časová osa

Stejně jako u osy se stanicemi musím problém s překrýváním textu řešit i u horizontální časové osy. V případě dostatečné šířky vykresluji celé hodiny i půl hodiny a pokud by se text překrýval, zobrazí se pouze celé hodiny s takovým odstupem, aby k překrývání nedošlo. Pro lepší orientaci v grafikonu se vykreslují na souřadnicích časů tenké vertikální úsečky.

Na obrázku 9.6 jde vidět optimální vykreslení všech časů a na obrázku 9.7 vykreslení textů tak, aby se nepřekrývaly.



Obrázek 9.6: Vykreslená horizontální osa s každou časovou jednotkou



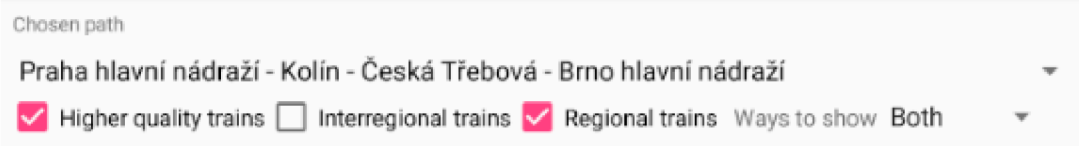
Obrázek 9.7: Vykreslená horizontální osa s nepravidelnými časovými jednotkami podle šířky textu

9.6.4 Filtrování spojů a výběr zobrazované cesty

Nad grafikonem jsou elementy pro výběr zobrazované cesty. V nabídce jsou nejvýše 3 cesty, které se z algoritmů vyhodnotili jako nejlepší.

Vzhledem k možnému vysokému počtu vykreslených spojů by byl grafikon nepřehledný, a proto si uživatel může pomocí filtru vybrat jaké typy a směry vlaků zobrazovat. Filtr zobrazuji v dialogu po stisku tlačítka v horním menu. Po změně nastavení filtrů celý grafikon překreslím a posunu a přiblížím na stejnou polohu, aby nedošlo k nechtěnému skoku v pohledu.

Na obrázku 9.8 je možné vidět některé dlouhé názvy zobrazené na dvou řádcích a také nevykreslené názvy stanic, které jsou reprezentovány pouze horizontální pomocnou úsečkou v grafikonu.



Obrázek 9.8: Filtrační část grafikonu

9.6.5 Zobrazení detailu spoje

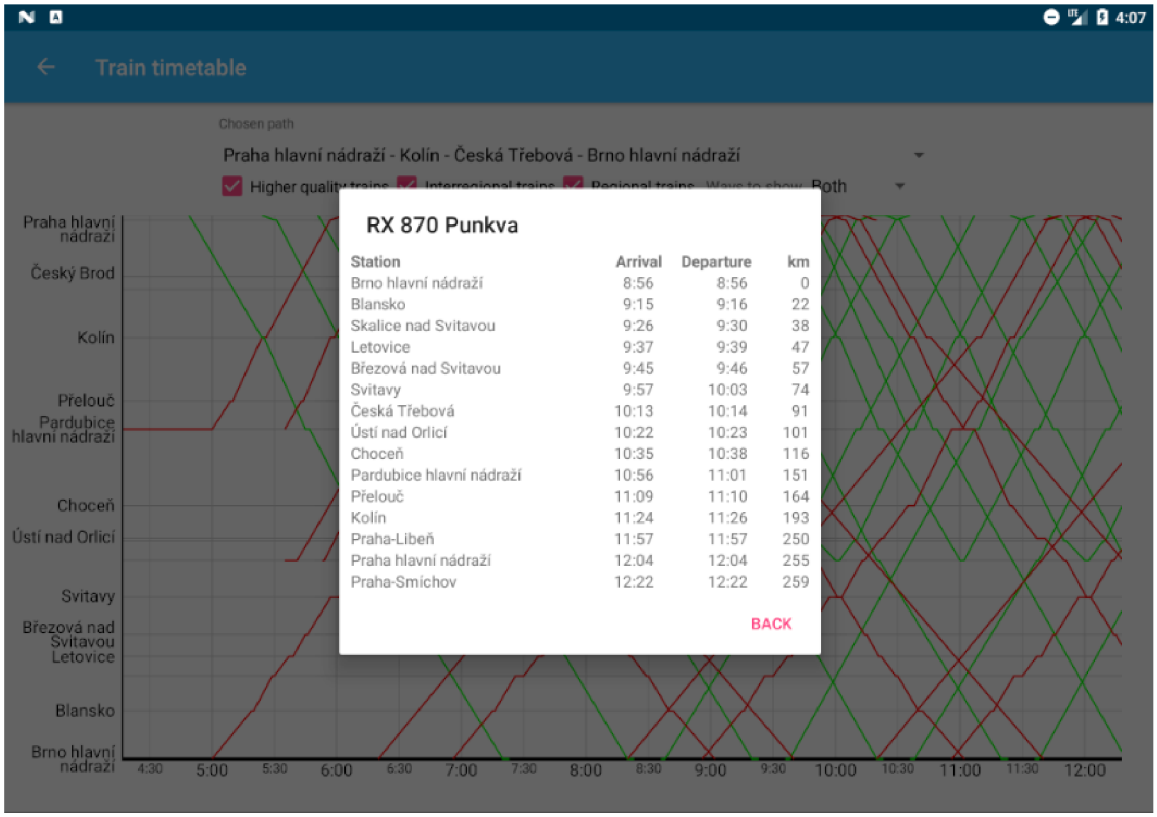
Uživatel má možnost zobrazit si detailní informace o vlakovém spoji, který zahrnuje veškeré jeho zastávky s časy příjezdů, odjezdů a vzdálenosti na trase. Vykreslované objekty, které znázorňují vlakové spoje ale bohužel neumožňují zjistit, zda bylo na objekt kliknuto. Zároveň událost kliku obsahuje souřadnice, které se počítají pouze v rámci obrazovky, takže nepočítá s posunutím nebo přiblížením grafu.

Pro výpočet skutečných souřadnic kliknutí musím s těmito údaji počítat. K dispozici jsou údaje o souřadnicích středu grafu `view.getCenter()`, což určuje posunutí v rámci grafu, a samotnou velikost přiblížení `scale`. Výpočet vypadá takto:

```
float realX = view.getCenter().x -
((graphView.getWidth() / 2) / scale) + (clickedX / scale);

float realY = view.getCenter().y -
((graphView.getHeight() / 2) / scale) + (clickedY / scale);
```

Po zjištění skutečné polohy kliku vytvořím pomocné kolečko o průměru 20 pixelů, které slouží pro zanedbání odchylky při nepřesném kliknutí prstem. Poté procházím vykreslenými úsečkami vlaků a testuji, zda se tyto 2 objekty protínají. Cyklus končí při prvním úspěšném protnutí, kdy vytvořím a zobrazím detail tohoto spoje. Na obrázku 9.9 je možné vidět vykreslené podrobnosti o vlakovém spoji, který se zobrazí po kliknutí na úsečku vlaku.



Obrázek 9.9: Zobrazení detailních informací o vlaku a jeho spoji

Kapitola 10

Testování

V této kapitole budou uvedeny zařízení, na kterých testování probíhalo a problémy na jaké se během toho přišlo. Dále budou popsány implementované řešení těchto problémů a možné rozšíření aplikace do budoucna.

10.1 Testovací zařízení

Testování v průběhu vývoje probíhalo na emulátoru, který je přítomen přímo ve vývojovém prostředí. Emulátor obsahuje velké množství zařízení s možností výběru verze operačního systému, takže se dá jednoduše testovat skrze několik zařízení. Vyvíjelo se primárně na emulovaném tabletu Nexus 9 s operačním systémem Android 7.1.1. Důkladné testování probíhalo celkově na těchto emulovaných zařízeních:

- Nexus 9 - úhlopříčka 8.9"- verze OS 4.4 - verze API 19
- Nexus 9 - úhlopříčka 8.9"- verze OS 5.1 - verze API 22
- Nexus 9 - úhlopříčka 8.9"- verze OS 7.1.1 - verze API 25
- Nexus 10 - úhlopříčka 10.1"- verze OS 8.1 - verze API 27
- Pixel 2 - úhlopříčka 5.0"- verze OS 8.1 - verze API 27

Emulovaná zařízení jsou sice odpovídají ve většině případech skutečnosti, ale i tak se nevyrovnají testu na reálných zařízeních. Testování na těchto zařízeních proběhlo až ke konci vývoje:

- Motorola Moto X - úhlopříčka 4.7"- verze OS 6.0.1 - verze API 23
- Honor 7 Lite - úhlopříčka 5.2"- verze OS 7.0 - verze API 24

10.2 Nalezené problémy

Při testování se projevila dlouhá doba hledání vlakových spojů. Například při hledání mezi Prahou hl. n. a Brnem hl. n. trvala celá operace přes 13 sekund. To je samozřejmě nepřijatelné a tak byl tento problém označen jako hlavní, který se musí vyřešit.

Dalším problémem, který již nebyl tak vážný, bylo vrácení pohledu grafikonu při změně ve filtru do původní pozice. Grafikon se celý překresluje, takže provedený posun či přiblížení byl ztracen a obraz skočil na nové místo.

10.3 Implementované řešení problémů

10.3.1 Rychlost vyhledání spojů

Při řešení rychlosti aplikace byla použita metoda profilování vláken na procesoru. Díky tomu byl získán přehled o tom, jaké metody trvají nejdéle času. Z této analýzy vyšlo najevo, že nejdelší jsou volání do databáze. Ta totiž probíhala v cyklech a bylo jich příliš mnoho. Díky logování bylo poté zjištěno, že spousta těchto volání byla provedena se stejnými argumenty. Bez optimalizace proběhlo v testovaném hledání až 900 volání.

Po tomto zjištění byly implementovány v mezipaměti mapy, které fungovaly jako cache a do kterých se ukládaly podle parametrů volání výsledky z databáze. Před voláním do databáze se tedy zjistilo, zda už proběhlo, a pokud ano, vzal se z výsledek z cache a volání stálo minimum času.

Díky této optimalizaci se omezil počet volání do databázeo 160 a čas vyhledání se tak snížil z **13 sekund** na **4 sekundy**. V případě, kdy se provede opakované hledání a cache tak již obsahuje všechny výsledky volání, proběhne celá operace pouze 0.5 sekundy.

10.3.2 Práce s grafikonem

Pro vyřešení problému s vrácením pohledu grafikonu při změně filtru do původní polohy stačilo před překreslováním uložit tuto původní pozici a přiblížení, a následně po vykreslení pohled na tuto pozici posunout. Bohužel metody, které nastavovaly pozice a přiblížení v kódu nefungovaly. Tyto atributy se ale daly použít při nastavování kreslícího plátna do pohledu, čím byl dosažen požadovaný efekt a pohled při změně filtru pozici nezměnil.

10.4 Možné rozšíření

Z testování také vyplynuly některé věci, které by byly dobré v budoucnu implementovat. Těmito věcmi jsou:

- Skript pro získání vlakových spojů – rozpoznání, ve které dny vlaky jezdí a přenesení těchto informací do databáze
- Aplikace
  - Práce s implementovaným rozšířením skriptu.
  - Přidání pokročilých možností do vyhledávacího formuláře.
  - Další optimalizace hledání tras a spojů.

Kapitola 11

Závěr

Cílem této práce bylo navrhnout a implementovat mobilní aplikaci pro operační systém Android pro interaktivní porovnávání vlakových spojů. Nejprve jsem provedl analýzu existujícího webového informačního systému nabízející stejnou službu a nastudoval informace o grafikonu vlakové dopravy a vývoji mobilních aplikací cílených na tablety. Poté jsem navrhnul databázi, způsob získání dat o vlakových spojích, a samotnou mobilní aplikaci. Na závěr proběhlo testování a implementace nalezených problémů.

Výsledkem práce je mobilní aplikace umožňující vyhledání vlakových spojů mezi 2 zastávkami. Aplikace vyhledá a zvolí optimální trasu mezi těmito zastávkami včetně přestupů a zobrazí vlakové spoje v grafikonu. Grafikon umožňuje filtrovat různé druhy vlaků a vybrat jaké směry se v grafu zobrazují. Také lze zvolit až ze 3 tras, které byly vybrány jako optimální. Druhou částí práce byl skript pro získání dat o vlakových tratích, zastávkách, a vlakových spojích, které se v grafikonu zobrazují.

Do budoucna by se mohl rozšířit skript pro získání vlakových spojů o to, v jaké dny vlaky jezdí a s tímto omezením počítat i v aplikaci. Zároveň by se mohla do vyhledávacího formuláře přidat pokročilá možnost pro manuální zadání přestupního bodu. V aplikaci by se mohlo více optimalizovat hledání cesty v grafu – mohla by se například přidat cena tratí, která by značila kolik vlakových spojů se na ní nachází a poté upřednosňovat tratě s větším počtem spojů.

Literatura

[1] CasperJS: *CasperJS documentation*. [Online; navštíveno 09.05.2018].
URL http://docs.casperjs.org/en/latest/

[2] Eid, I.: *RoomAsset*. [Online; navštíveno 01.02.2018].
URL https://github.com/humazed/RoomAsset

[3] ŽelPage: *Řazení vlaků*. [Online; navštíveno 05.02.2018].
URL http://www.zelpage.cz/razeni/

[4] ŽelPage: *Seznam tratí ČR*. [Online; navštíveno 13.12.2017].
URL http://www.zelpage.cz/trate/ceska-republika

[5] ČESKO: *Vyhláška č. 122/2014 Sb., o jízdních řádech veřejné linkové dopravy*. [Online; Citováno 28. 02. 2018].
URL https://www.zakonyprolidi.cz/cs/2014-122

[6] ČESKO: *Vyhláška č. 7/2015 Sb., vyhláška, kterou se mění vyhláška č. 173/1995 Sb., kterou se vydává dopravní řád drah, ve znění pozdějších předpisů*. [Online; Citováno 28. 02. 2018].
URL https://www.zakonyprolidi.cz/cs/2015-7

[7] Gilleland, M.: *Levenshtein Distance, in Three Flavors*. [Online; navštíveno 16.03.2018].
URL https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm

[8] Google: *Distribution dashboard*. [Online; navštíveno 09.05.2018].
URL https://developer.android.com/about/dashboards/

[9] Google: *LiveData Overview*. [Online; navštíveno 16.12.2017].
URL https://developer.android.com/topic/libraries/architecture/livedata

[10] Google: *Material design guidelines*. [Online; navštíveno 12.10.2017].
URL https://material.io/guidelines/

[11] Google: *ViewModel Overview*. [Online; navštíveno 16.12.2017].
URL
https://developer.android.com/topic/libraries/architecture/viewmodel

[12] Howard Wainer, J. N., Polina Harik: *Visual Revelations: Stigler's Law of Eponymy and Marey's Train Schedule: Did Serjev Do It Before Ibry, and What About Jules Petiet? CHANCE*, ročník 26, 2013: s. 53–56.

[13] IDNES.CZ: *IDOS*. [Online; navštíveno 12. 10. 2017].
URL https://jizdnirady.idnes.cz/vlakyautobusy/spojeni/

[14] IDOS: *Spoje*. [Online; navštíveno 20.11.2017].
URL https://jizdnirady.idnes.cz/vlakyautobusy/spoje/

[15] Marey, E. J.: *La Méthode graphique dans les sciences expérimentales et particuliérement en physiologie et en médecine*. Paris, 1878.

[16] Seznam: *Jízdní řády*. 2018, [Online; navštíveno 28. 02. 2018].
URL https://www.seznam.cz/jizdnirady/muni%3A5740%3A9mMT8xTuNN-%3Emuni%3A3468%3A9hChxxXvtO

[17] Tufte, E. R.: *The Visual Display of Quantitative Information*. Cheshire, Connecticut: Graphics Press, 1983, ISBN 978-0961392147.

[18] Uhlíř, J.: *Interaktivní porovnávání vlakových spojů*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2017.

[19] Wikipedia: *Pojmenování vlaků ČD*. [Online; navštíveno 05.02.2018].
URL
https://cs.wikipedia.org/wiki/Pojmenov%C3%A1n%C3%AD_vlak%C5%AF_%C4%8CD