Integrovaný vyhledávač spojení v regionální dopravě

Osnova bakalářské práce

# Analýza problému

## Veřejná doprava v ČR

První podkapitola popisuje fungování a specifika veřejné dopravy a informačních technologií, na kterých je v současné době postavena.

### Organizace veřejné dopravy

Tato sekce se bude zabývat stručným popisem našeho systému veřejné dopravy, zejména tedy regulovanými i komerčními tarify, integrovanými dopravními systémy, liberalizací železnice a dalších skutečností relevantních pro zbytek práce.

### „Otevřená“ data o dopravě

Zde se poprvé ponoříme do strastiplného tématu otevřených dat, Celostátního informačního systému JŘ a strojové zpracovatelnosti (zatím jen obecně).

## Hledače spojení/plánovače trasy

### Status quo

Tato sekce krátce připomene, co si kladou za cíl běžné hledače spojení (beze změny už posledních 20 let) a nastíní požadavky na softwarovou implementaci (rychlost, využití prostředků, škálovatelnost).

### Inspirace ke zlepšení

Minulé sekci se v této nastaví pomyslné zrcadlo – můžeme například uvést paralelu se Skyscannerem apod. a navrhnout přístup ke hledání, který používá podobný přístup k optimalizaci na základě rozličných tarifů v ČR (popsaných v minulé kapitole).

## Aplikace pro veřejnou dopravu

Zde podrobněji popíšeme, jaké funkce (vedle hledání spojení) používají aplikace organizátorů veřejné dopravy (Lítačka, ODISka atd.) a co se od takových aplikací obvykle očekává.

## Námět práce

Čeho se vlastně snažíme dosáhnout, co k tomu je potřeba a co naopak prozatím odložíme.

### Serverová a výkonná část

Tady se vysvětlí, proč je potřeba celý „supply chain“ vyhledávače spojení postavit od začátku (jak bylo nastíněno v minulých kapitolách) a v jaké jsme pozici oproti Chapsu s datovým monopolem a Seznamu s právníky. Co chceme, aby náš algoritmus uměl oproti stávajícím. Také upřesníme hranici mezi statickými a real-time daty a zamýšlenými způsoby jejich zpracování a obecně, co vlastně chceme, aby náš server byl schopný navenek poskytovat. Mimoto se zde vymezí, že zpracováváme pouze české JŘ (nikoli slovenské či dokonce evropské), nebereme zřetel na soukromé dopravce neoperující alespoň částečně pod tarifem IDS apod.

### Mobilní aplikace

Zde je namístě obhájit, proč má smysl dělat mobilní aplikaci a nikoli jen webovou službu, jaké cíle si aplikace klade co do komunikace se třetími stranami a také jaké jsou požadavky na design a ovládání. Součástí toho bude také plán, co všechno by bylo fajn okopírovat od existujících aplikací (třeba to posouvání doleva a doprava v lítačce/IDOSu, hledání podle polohy atd.) a co by bylo fajn okopírovat, ale nemám na to kapacitu (hledání podle města/obce, přesné délky přestupů). Je docela na hraně, co z toho vlastně patří sem a co do minulé sekce.

# Analýza řešení – zdroje dat

## Existující řešení

### Komerční

Tato sekce se bude zabývat především IDOSem a jejich neochotou zákonu navzdory data publikovat v rozumné kvalitě, a jeho dopadem na konkurenční řešení (a v důsledku toho i tuto práci).

Zmínka by měla padnout i o službách ve stylu Lítačka, tarifní počítadla, TMapy apod., které jsou všechny ve výsledku závislé na IDOSu.

### Amatérská

Krátká kapitola o historických pokusech o práci s daty o jízdních řádech mimo komerční sektor a jak to s nimi dopadlo.

## Datové formáty

Mnoho prací má často sekci, ve které se ospravedlňuje výběr použitých technologií. Já na výběr příliš neměl, neboť ty technologie už jaksi existují na straně poskytovatelů dat. Zde se jim tedy, a nedomyšlenostem v nich, budu věnovat.

### Formát JDF CIS JŘ

Tenhle si zrovna určitě nezaslouží uniknout bez velmi kritické zmínky, jak ohledně jeho vadného návrhu, tak kompletního ignorování jeho specifikace CHAPSem.

### TAF-TSI a RailML (SŽDC)

O něm pro změnu není zas tolik co říct, protože je navržený docela dobře, jen by se opět hodilo poznamenat, které podivuhodnosti se v něm mnohdy vyskytují v případě SŽ/CISu.

### GTFS

Tenhle „standardizovaný“ formát od Googlu používá Praha a Brno. Má ale spoustu nedostatků, které sice mnohdy řeší rozšíření, ale oproti JDF se často některé věci pak musejí dost obcházet.

### Abirun TIM, TMapy, MPVNet a další řešení na míru IDS

Tohle jsou pozoruhodné služby, které poskytují firmy, o kterých dříve málokdo slyšel, dokud záhy nezačaly sosat z rozpočtu veřejné dopravy. Za zmínku stojí především jejich dynamicita (mění se bez oznámení každou chvíli a hodně toho rozbíjí).

## Sjednocení dat

### Identifikace zastávek

Problematika identifikace zastávek v CIS JŘ, která bezpochyby existuje, ale není zveřejněná. Vysvětlení, že zastávky potřebujeme nutně identifikovat, protože máme spoustu zdrojů dat, které potřebujeme navzájem provázat a zabránit duplikátům, a návrhy, jak to udělat.

### Data specifická pro IDS

Kvůli kompatibilitě s real-time daty potřebujeme zachovat některé informace, které jsou specifické pro konkrétní zdroj dat. Ne vždy stačí unikátní ID a je kolem toho potřeba tu a tam něco vymyslet.

## Technologie

Diskuze nad možnými formáty ukládání dat – relační vs grafové databáze (neo4j), existující řešení nad nimi (dp-nss od někoho na ČVUT). Proč nakonec používám SQLite místo serverově založených databází.

# Analýza řešení – vyhledávací jádro

## Algoritmy plánování trasy

### Obecné požadavky

Zde ilustrujeme, jaké omezení a zadání jsou obvykle předhozena vyhledávačům spojení a chceme je podporovat v algoritmu – tj. počet přestupů, jízda „přes“ apod.

### Dijkstra a jeho deriváty

Jak Dijkstra funguje, asi všichni víme, proto by se tato sekce měla zabývat spíše jeho úpravou pro hledání nejrychlejší trasy. Dále také, proč způsobuje problémy při přidání omezujících kritérií (počet/délka přestupů) a proč se ukázal jako nevhodný. (pozn.: nemá tohle být až v řešení, nikoli analýze?)

### RAPTOR

Rozdíly mezi Dijkstrou a RAPTORem, škálovatelnost, paralelizovatelnost. Je vhodné se také zaměřit na jednoduchou podporu pro multikriteriální vyhledávání a jak ji implementuji v práci (není nutně do puntíku stejná).

## Algoritmy výpočtu ceny

### Obecné požadavky

Vzhledem k tomu, že vyhledávací jádro nutně musí být schopno počítat cenu i „za běhu“, a nikoli jen po nalezení celé trasy, bude potřeba některé algoritmy nějakým zjednodušením donutit, aby běžely velice rychle. Můžeme diskutovat o tom, do jaké míry je zde žádoucí odchýlit se od tarifních podmínek v zájmu rychlosti výpočtu, který musí být rychlý, ale ne nutně korektní.

### Kilometrické tarify

Na první pohled jsou jednoduché, ale zkomplikují se, jakmile povolíme možnost mít předplacené tarifní zóny.

### Zónově relační tarify

Nejobvyklejší tarify, kde je cena určena počtem projetých zón. Pojednání o nabízejících se optimalizacích (bitové masky, tarifní matice) a podivuhodných rozšířeních, které si vymýšlí větší města pro svou MHD (Brno).

### Pásmový tarif PID

Nejpodivnější a nejproblematičtější tarif, který si bezpochyby zaslouží hlubší diskuzi, zejména proto, jaké kompromisy u něj je nutné uskutečnit pro rychlý výpočet, kvůli nekonzistentnímu zvýhodněnému jízdnému a různému přiřazení zastávek do tarifních pásem od spoje ke spoji.

## Technologie

Nutnost použití high-performance prostředí a optimalizovaných datových formátů pro některé operace, JNI pro komunikaci s backendem služby, rozbor možných bezpečnostních rizik. Proč C++, a ne třeba Rust a srovnání dostupných vývojových nástrojů pro tyto jazyky (pozn.: Visual Studio pro C++ je výrazně napřed, pro velká data debug printy nestačí atd.).

# Analýza řešení – aplikace

## Klient

### Požadavky na UI

Mobilní aplikace jsou specifické velice netrpělivými uživateli, je tedy důležité se zaměřit na co nejlepší flow celého rozhraní, dostupnost funkcí v co nejméně krocích a rychlé načítání. Zejména aplikace, které jsou již na trhu po léta, už toto mají zmáknuté a je docela oříšek přijít se zlepšením (i když to jde). Bylo by tedy dobré rozebrat, čím se inspirovat, a co naopak vylepšit.

### Spolehlivost

U jízdních řádů je zpravidla největší katastrofou, pokud se čas od času zmýlí. Je tedy vhodné prozkoumat možnosti, jak tomuto co nejlépe zabránit, a zároveň, proč to mnohdy nejde.

### Aktuálnost

Nejenže by data prezentovaná aplikací měla být platná k datu hledání, ale navíc by se měla aktualizovat průběžně v reálném čase (zejména co se týče odřeknutých spojů a zpoždění). To ale má svá úskalí, neboť služby poskytující tato data nejsou v naší moci.

### Integrovanost a interoperabilita

Aplikace by také měla spolupracovat s existujícími službami a aplikacemi, neboť sama o sobě spoustu věcí (zejména nákup jízdenek) nezmůže. Ne vždy to ale je tak jednoduché.

### Technické požadavky

U Androidu sice není moc, o čem se rozhodovat, ale bylo by dobré zdůvodnit, proč se hodí cílit na Android 8 a obecně o sdílení kódu mezi serverovou a klientskou částí, které to umožňuje.

## Server

### Zabezpečení

Vzhledem k tomu, že vyhledávací jádro běží v nativním kódu, je dobré se u serveru alespoň v krátkosti zmínit o důležitosti validace vstupu a zpracovávání výjimek.

### Konzistence dat

Data jízdních řádů vystoupivší z původních zdrojů nejsou inkrementální a mnohdy neobsahují unikátní identifikátory, je tedy nutné zajistit, aby byla data odeslaná klientovi dohledatelná při následujících požadavcích i v případě aktualizace datové sady.

### Sdílení kódové základny

Vysvětlení, proč je pro nás vhodné sdílet komponentu datového modelu mezi serverem a klientem, jaké to má dopady na implementaci (API, která můžeme použít).

### Technologie

Něco o Spring Bootu a Jakarta EE, výhody jejich použití oproti jiným, méně backend-oriented frameworkům.

# Řešení – datový model a zdroje

## Číselníky a kódovníky

Popis snahy o unifikaci výčtových parametrů z jízdních řádů, jako služby ve vlaku nebo dopravní prostředky. Naopak vysvětlení, kde to není zcela možné (komerční druhy vlaků).

## Centrální registr zastávek

### Účel a struktura

Popisuje, jaké atributy CRZ používá pro identifikaci zastávek, jak je izolovaný od zbytku serveru, jaké operace podporuje, že funguje jak vzdáleně, tak lokálně apod.

### Algoritmy

Rozbor algoritmu „písmenkového stromu se zástupnými znaky“, který CRZ používá pro shody jmen zastávek (třeba „Nové Město n.M.,,žel.st.“ s „Nové Město n. Metují., ž. st.“ Také způsob normalizace jmen pro hledání.

## Framework pro sběr dat

Ilustrace vrstevnatého systému sběrače dat, jak je paralelizován a kde naopak paralelizován být nemůže.

## Úskalí sběru dat

Tohle bude taková dlouhá megakapitola o všech možných problémech, které mohou nastat (a nastaly) při sběru dat ze zdrojů třetích stran, a jak se jim dá částečně či úplně zamezit. Hádal bych, že tu dost možná bude tak deset až dvacet podkapitol, protože jich bylo opravdu nezměrné množství.

## Unifikace a procesy nad daty

Co se s daty děje, když je konečně všechna postahujeme – tj. předložení do CRZ, výpočet pěších přestupů a tak dále. Také, jak se řeší slévání JŘ dohromady zejména v případě PID, jehož JŘ musíme neprve opatrně vymazat ze zdrojů z JDF, než je nahradíme novými z jejich GTFS.

# Řešení – REST API

## Datový model

Ten je víceméně stejný jako z minulé kapitoly, zde by se tedy jen popsaly odlišnosti, jako že se používají někde stringy místo enumů a podobně.

## Bezestavové požadavky

Tohle je super sekce, ve které se můžeme vysmát IDOSu, který používá na každé hledání jakousi „handle“, která může ovšem vypršet, a když tak otevřeme vyhledávač s výsledky z večera ráno, řekne to, že ho musíme obnovit. Implementovat jeho funkcionalitu bezestavově ale není zcela přímočaré, takže by se o tom mělo něco napsat.

## Serializace a přenos

Krátká sekce o použití Retrofit2 a Jacksonu k de/serializaci dat. Problémy, které způsoboval Jackson na Androidu s javovskými Recordy, a jak byl opraven.

# Řešení – vyhledávací jádro

## Formát dat

Vysvětlení, proč není úplně moudré používat SQLite v high-performance grafových algoritmech. Popis formátu (jen obecný, detailní bude až v dokumentaci) a struktur použitých pro akceleraci vyhledávání. Uvedení do konceptů použitých v designu formátu a modelu pro jeho rozšíření, kompatibilita napříč platformami.

## Reprezentace času

Při hledání téměř neustále potřebujeme nějaký údaj o reálném čase, ze kterého je možné velmi rychle zjistit datum (kvůli platnosti JŘ a časových kuponů) a hodiny/minuty ve dni (kvůli odjezdům spojů). Zároveň bychom chtěli, aby nezabíral moc místa (ne víc než Unixové razítko, ze kterého se ale datum zjišťuje komplikovaně) a bylo jej možné porovnávat jako celé číslo. Popis řešení, kterým jsme toho dosáhli.

## Šablony

Proč se ukázalo, že je nezbytně nutné hledat podle příjezdu i odjezdu a jak použít šablony jazyka C++ pro řešení této situace bez zbytečné duplikace kódu. Jak bylo toto řešení dále rozšířeno pro podporu kritéria ceny v RAPTORu bez duplikace ani snížení výkonu.

## Implementace Dijkstry

Popis implementace Dijkstry, jeho nevýhody a výhody, včetně benchmarku (pro porovnání s RAPTORem).

## Implementace RAPTORa

Popis implementace RAPTORa pro normální i multikriteriální hledání a benchmark obou.

## Inkrementální výpočet tarifu

Vysvětlení motivace pro inkrementální výpočet, požadavky na jeho přesnost a „axiomy“, které by měl splňovat, ale mnohdy nesplňuje.

## Přesný výpočet tarifu

Jak to chodí, když chceme vypočítat cenu pro zafixovanou trasu a techniky použité pro akceleraci. Vzhledem k tomu, že se věnujeme jen několika tarifům (PID, IDPK, nějaké snad ještě přibudou), je můžeme rozebrat každý zvlášť, nejsou-li víceméně totožné.

# Řešení – služby třetích stran

## Abstrakce

Real-time služby si každé hrají svou, takže je potřeba je nějak sjednotit, abychom je mohli prezentovat v UI mobilní aplikace.

## Rozhraní pro podsystémy

Jak vypadá rozhraní použité pro samostatné podsystémy aplikace, sloužící pro obsluhování požadavků třetích stran.

## Podsystém jízdenek IDPK

Na rozdíl od PID neposkytuje aplikace Plzeňského kraje možnost zakoupit jízdenku přes hluboký odkaz, moje aplikace tedy implementuje její funkcionalitu (MAP Phone) sama od sebe. Tato sekce by měla popsat, jak se to dělá a jaké to má potenciální nedostatky.

## Zabezpečení dat

Nebylo by úplně cool, kdyby díky mé aplikace někomu ukradli heslo k Lítačce. Proto by zde bylo dobré prozkoumat, jak se vlastně přístupové údaje k těmto službám reprezentují, jak je můžeme ukládat a jaká z toho plynou bezpečnostní rizika.

# Řešení – mobilní aplikace

## Uživatelské rozhraní

### Architektura

Takové ty obecné průpovídky o tom, že to má dole navigační lištu, jak to zachovává stav, kde jsou oddělené aktivity od fragmentů apod.

### Komponenty – přepínač data/času

Jak funguje můj úžasný horizontální výběr data a času, v čem je lepší než to, co je v Mém Vlaku (fuj).

### Navigace gesty

Jak aplikace používá dvojité kliknutí a dlouhé podržení jako zkratky k akcím, které by normálně byly složitější.

### Karty

Aplikace v návrhu hojně využívá „karet“, např. v detailu spoje karta s trasou, s výlukami a se službami. Vysvětlit, jak to pomáhá s lazy loadingem.

### Tematický design

Služby třetích stran, jako je Lítačka nebo Virtuální karta, mají obvykle svou vlastní identitu, tj. logo, barvy a podobně. Tu sice není z právního hlediska možné jen tak okopírovat, aplikace se jí ale ve vhodných situacích přibližuje zabarvováním UI do tematických barev. Přitom pro co nejvíce komponent používá sdílený kód i rozvržení (přihlašovací obrazovka).

### Nákup jízdenek

Sám sice nikomu nic neprodávám, ale je zvykem, že v situacích, kde se na obrazovce objeví nějaký podnět k platbě, se obvykle objevuje spousta příjemných obrázků (manipulace). Zde se představí, jak toho dosahuje moje aplikace.

### Online/offline informace

Obdobně jako v aplikaci Můj Vlak jsou real-time data oddělená od statických a aplikace umí většinu funkcionality přizpůsobit jejich dostupnosti. Příklad, jak to funguje u detailu spoje.

### Kontrola jízdenek

Jak funguje obrazovka pro zobrazení QR kódu jízdenky, včetně využití senzorů zařízení.

### Výsledky hledání

Jak se zde používá nekonečné scrollování nahoru a dolů, swajpování spojů zleva doprava pro přizpůsobení trasy. Proč to funguje tak blbě (a proč to tak blbě funguje i u Lítačky a proč za to může Google).

### Hlavní obrazovka

Jak probíhal návrh kriticky oceňované hlavní obrazovky, co na ní uživatelům připadá nejdůležitější a co zde naopak schází.

## Perzistence dat

Jak se data v aplikaci ukládají a synchronizují se serverem.

## Aktualizace

Aplikace bude (brzo, snad) mít možnost aktualizovat se sama, aby se nemusely manuálně stahovat balíčky. Mechanismus, jak to funguje, bude popsán zde.

### Kompatibilita s API

Řešení kompatibility mezi různými verzemi API a aplikace.

# Další vývoj

## Změny tarifů

Velká část informací, na kterých je aplikace založená, byla získána na základě zákona 106/1999 manuálně a nedisponuje možností automatizované aktualizace. Je tedy nutné systém v případě změn upravovat. Byla však vyvinuta snaha nutnost úprav co nejvíce omezit.

## Informace o linkách a spojích

Aplikace momentálně umožňuje pouze hledání spojení a zobrazování odjezdů/příjezdů ze stanic. Hodilo by se umět i vyčarovat linkové jízdní řády a sledování konkrétního spoje.

## Podpora dalších dopravních systémů

Někteří dopravci, např. Arriva, mají své vlastní tarify vedle těch integrovaných, nebo (v drážní dopravě) podporují Systém jednotného tarifu aneb OneTicket. Ačkoli jsou aplikací de facto podporovány i takové tarify, implementován je pouze tarif Českých drah TR10. Bohužel to mnohdy není možné, protože skutečné ceny jsou mnohdy nižší než ty nominální, v případě RegioJetu třeba „Uvedené ceny jsou nejvyšší možné při nákupu jízdenky ve vlaku. Klienti však mohou cestovat levněji, v závislosti na času spoje a dne (sic).“ Takovéto ceníky předpovídané jakýmsi orákulem kdesi na serveru bohužel použít nelze.

## Webová aplikace

V omezené šíři (tedy bez podpory API třetích stran) by hledání poskytované serverem mohlo být možné využít i v aplikaci pro webové prohlížeče.

## Historie zpoždění

Taková docela šikovná funkce, kterou má zatím myslím jen IDOS, a pro některé služby (PID, ČD, Brno) by ji asi i šlo implementovat.

## Proxy pro real-time data

Praha a Brno uveřeňují real-time data jako GTFSReal proudy, které není úplně praktické číst přímo z MA, ale mohlo by se hodit je agregovat na serveru a pak předávat dál (licence to prý umožňuje). Momentálně je totiž čtu z API Golemio, které má rate limit.

## Unit testy

Jelikož vyhledávací algoritmy běží na velmi nestálé databázi, neobsahují žádné mocking prostředí ani unit testy. To je překvapivě docela velký problém, protože data jsou extrémně různorodá a malé změny v kódu mnohdy rozbijí chování, které jsme dlouho brali za samozřejmost. Unit testy s dobrou coverage by toto měly přinejmenším zmírnit.

# Závěr

### Konkurence

Jak si aplikace vede oproti IDOSu, PubTranu apod. co se výsledků a cen týče, kde je lepší a kde naopak horší, co by se dalo upravit.

### A co dál

Shrnutí funkčnosti a spolehlivosti aplikace, úvaha o možnosti reálného použití apod.