Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Vývoj mobilní aplikace s užitím automatizace logistiky

Bc. Kateřina Svobodová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Kateřina Svobodová

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj mobilní aplikace s užitím automatizace logistiky

Název anglicky

Development of mobile application with the utilizitation of logistics automation

---

Cíle práce

Hlavním cílem práce je ověřit možnosti vývoje klient-server mobilních aplikací B2C v jazyce Java. Dalším cílem je prokoumat možnosti automatizace logistiky s možností uplatnění mikrologistických matematických modelů a jejich aplikaci pro dopravní, manipulační a zásobovací logistiku v rámci mobilních aplikací B2C pro malé firmy. Součástí aplikace bude i jednoduché rozhraní pro plánování rozvozu nebo doručování s využitím geolokačních dat. Vytvořená mobilní aplikace bude určena především pro operační systém Android.

Metodika

1) Na základě studia odborných zdrojů zpracujte literární rešerši z oblasti vývoje klient-server mobilních aplikací a uplatnění principů logistiky v automatizaci rozvozu a doručování

2) Na základě prototypu vytvořte klientskou aplikaci pro systém Android a s pomocí základnách postupů UML vytvořte serverovou část aplikace zahrnující podporu automatizace logistiky

3) Posuďte a zhodnoťte použité metody pro vývoj aplikace a postupy automatizace logistiky

---

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

Mobilní aplikace, Java, Android, logistika, B2C, GIS

Doporučené zdroje informací

ECKEL, B. *Myslíme v jazyku Java : knihovna programátora.*.

HEROUT, P. *Učebnice jazyka Java.* České Budějovice: Kopp, 2015. ISBN 978-80-7232-398-2.

HORTON, J. Android Programming for Beginners. Birmingham: Packt Publishing, 2018. ISBN 978-1789538502.

LAMBERT, D M. – STOCK, J R. – ELLRAM, L M. *Logistika.* Brno: CP Books, 2005. ISBN 80-251-0504-0.

PECINOVSKÝ, R. *Myslíme objektově v jazyku Java : kompletní učebnice pro začátečníky.* Praha: Grada, 2009.

SIXTA, J. – MAČÁT, V. *Logistika : teorie a praxe.* Brno: CP Books, 2005. ISBN 80-251-0573-3.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. David Buchtela, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 3. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 25. 3. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Vývoj mobilní aplikace s užitím automatizace logistiky“ jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 31. března 2023

Kateřina Svobodová

Poděkování

Děkuji vedoucímu bakalářské práce, panu Ing. Davidu Buchtelovi, Ph.D. za možnost výběru vlastního tématu, za odborné vedení a poskytnutí cenných rad během psaní práce.

Vývoj mobilní aplikace s užitím automatizace logistiky

Abstrakt

Tato diplomová práce se zabývá ověřením možnosti vývoje klient – server mobilních aplikací B2C v jazyce Java a výzkumem možnosti automatizace logistiky společně s uplatněním mikrologistických matematických modelů a jejich aplikací v dopravní, manipulační a zásobovací logistice v rámci mobilních aplikací B2C pro malé firmy. Výstupem je ověření postupů návrhu aplikace určené primárně pro obsluhu doručování poslední míle pro koncové zákazníky. Aplikace používá klient – server architekturu s klientskými mobilními zařízeními pro operační systém Android a serverovou aplikací v jazyce Java. Jako modelový příklad autorce slouží aplikace pro firmu, která se zabývá půjčováním šatů. Aplikace zahrnuje nejen klientské aplikace, ale zahrnuje i plánování rozvozu, popř. poskytuje údaje o chování zákazníků, stejně jako osob, které zboží rozvážejí. V této práci je ověřeno použití v praxi využitelných softwarových řešení a online služeb, které lze použít k vybudování softwarového systému pro řízení doručování koncovým zákazníkům. Aplikace předpokládá použití mobilních klientských zařízení pro komunikaci v systému klient – server a současně jako samostatná zařízení využívající online navigační a mapové služby pro navigaci při rozvozu.

Klíčová slova: Mobilní aplikace, Java, Android, logistika, B2C, GIS, operační systém, problém nejkratší cesty

Development of mobile application with the utilizitation of logistics automation

Abstract

This master's thesis deals with the verification of the possibility of developing client – server mobile B2C applications in Java and the research of the possibility of automation of logistics together with the application of micro logistic mathematical models and their applications in transport, handling and supply logistics within mobile B2C applications for small companies. The output is a validation of application design procedures primarily designed to handle last mile delivery for retail customers. The application uses a client – server architecture with client mobile devices for the Android operating system and a server application in Java. The author uses a mobile application for a dress rental company as a model example. The application not only includes client applications, but also includes delivery scheduling, or provides data on the behaviour of customers as well as the people who deliver the goods. In this paper, the use of practical software solutions and online services that can be used to build a software system for managing delivery to end customers is verified. The application assumes the use of mobile client devices for communication in a client – server system and at the same time as stand-alone devices using online navigation and mapping services for delivery navigation.

Keywords: Java, Android, logistics, B2C, GIS, mobile application, operating system, shortest path problem

Obsah

1 Úvod

Mobilní zařízení a jejich aplikace už neodmyslitelně patří ke každodennímu fungování v moderní době. Již v roce 2016 byly mobilní telefony první volbou ve způsobu přístupu na internet pro více než polovinu lidí na světě (StatCounter, 2016). Mobilní telefony a aplikace jsou tedy významnou částí našich životů. Zejména z toho důvodu je předkládaná diplomová práce zaměřená na mobilní aplikace, na zkoumání postupů i možností vývoje aplikací a jejich jednotlivých částí.

Jedním z nejčastějších typů mobilních aplikací jsou aplikace zaměřené na B2C sektor ve formě internetových obchodů. Takové aplikace řeší především propojení firmy se zákazníkem. Prostřednictvím aplikace je řešena celá logistika – prodej, doprava a dodání zboží, stejně jako postupy, jak řešit vzniklé problémy, případně automatizace některých procesů.

Android OS je dnes nejpoužívanějším operačním systémem pro mobilní zařízení, s podílem přes 72 % (StatCounter, 2022) a v jím používaném obchodě s aplikacemi jsou miliony funkčních aplikací. Ačkoli je určen primárně pro mobilní zařízení, setkat se s ním můžeme mj. i v televizorech nebo automobilech. I z tohoto důvodu se tato diplomová práce bude zaobírat aplikacemi právě pro Android OS.

Diplomová práce je rozdělena do dvou hlavních částí, a sice teoretické a praktické. Do teoretické části spadají první dvě kapitoly. První kapitola je věnovaná cílům práce a metodice. Druhá kapitola si klade za cíl definovat a vymezit problematiku OS Android, zabývá se otázkou designu a pravidly, které se v této oblasti uplatňují, popsány jsou zde vlastnosti modelu klient – server. Je zde rovněž zahrnuto i využití map, které jsou u tohoto typu aplikací integrovány a dělí se na tzv. vektorové a rastrové. Nelze opomenout klíčové algoritmy, a sice Dijkstrův, A* a Floyd-Warshallův, kterými je řešena co možná nejkratší cesta k zákazníkovi. Poslední dvě podkapitoly jsou věnovány mikrologistice a základnímu vývoji aplikací.

Praktická část této diplomové práce je rozdělena na dvě kapitoly. Kapitola *Vlastní práce* se zabývá vývojem aplikace. Jedná se konkrétně o model klient – server a zaměřená je na koncepci serverové část a řešení logistiky. Jde o aplikaci, která zprostředkovává prodej šatů. Poslední kapitola *Výsledky a diskuse* si klade za cíl shrnout přínosy této aplikace.

2 Cíl práce a metodika

Cílem práce je ověřit možnosti vývoje mobilních B2C aplikací s automatizací logistiky. S použitím modelové implementace klient-server mobilní aplikace určené pro B2C segment ověří možnosti uplatnění mikrologistických matematických modelů a jejich aplikaci pro dopravní, manipulační a zásobovací logistiku.

Jako modelový případ používá návrh mobilní aplikace, která rozšíří dosavadní služby nabízené firmou provozující půjčovnu šatů. Tyto služby budou zahrnovat dovezení šatů na akci, popřípadě domů a zpět. Uživatel bude mít možnost si vybrat několik šatů, které mu budou následně v krátké době doručeny. Šaty, které si vybere, mu budou zapůjčeny, nebo si je po skončení akce může za sníženou kupní cenu ponechat. Tato aplikace bude klientskou částí systému a bude určena především pro operační systém Android. Součástí aplikace bude i jednoduché rozhraní pro plánování rozvozu nebo doručování. Aplikace bude poskytovat údaje o chování zákazníků a osob provádějících rozvoz nebo doručování pro účely automatizace logistiky. Klientská aplikace bude používat serverovou část. Obě části systému, klientská i serverová, budou vytvořeny v jazyce Java a budou využívat data mapových serverů pro plánování dodávek a pro automatizaci logistiky v serverové části.

Implementace ověří možnosti uplatnění mikrologistických matematických modelů a jejich aplikaci pro dopravní, manipulační a zásobovací logistiku malých klient-server mobilních aplikací určených pro B2C segment.

3 Teoretická východiska

3.1 Android OS

Operační systém Android je open source platforma určená především pro mobilní zařízení, avšak dnes nalézá své využití i v televizorech či automobilech. Začal vznikat již v roce 2003 v Kalifornii. Roku 2005 tuto malou značku koupil gigant Google Inc., který Android provozuje dodnes (Gargenta, 2011).

3.1.1 Open source platforma

Open source software (otevřený software) znamená, že daný software může kdokoli volně použít a do jisté míry upravovat (opensource.com). Android OS spadá do Open Handset Alliance, sdružení výrobců mobilních zařízení, které má za cíl rozvoj nových technologií (Gargenta, 2011). Kromě Androidu jsou součástí tohoto sdružení také firmy jako Intel, NVIDIA, Samsung, a další.

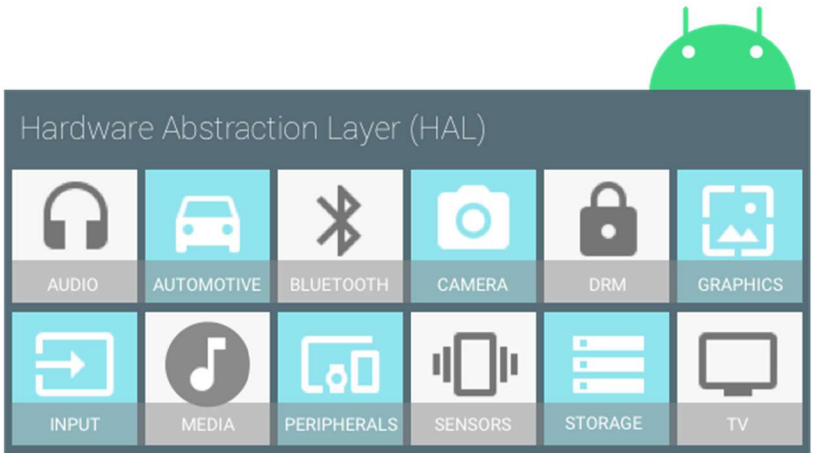
3.1.2 Architektura

Android OS se skládá z několika vrstev. Některé vrstvy nejsou jasně vymezené a často se prolínají.

Linuxové jádro je první a nejspodnější vrstvou architektury Android OS. Ta nabízí skvělou přenositelnost, bezpečnost a další funkce vyplývající z jeho širokého využití a popularity (Gargenta, 2011). Jádro zajišťuje především komunikaci mezi hardwarem a softwarem pomocí ovladačů, nebo třeba správu napájení, paměti a síťového připojení.

Další vrstvou je Hardware Abstraction Layer (HAL). Tato vrstva vytváří jednotné rozhraní, které je schopné ovládat širokou škálu hardwarových zařízení. Díky této vrstvě je Android schopen fungovat na velkém množství různých zařízení, a tak je snadno přenositelný.

Obrázek 1 - Komponenty vrstvy HAL (Android Inc.)

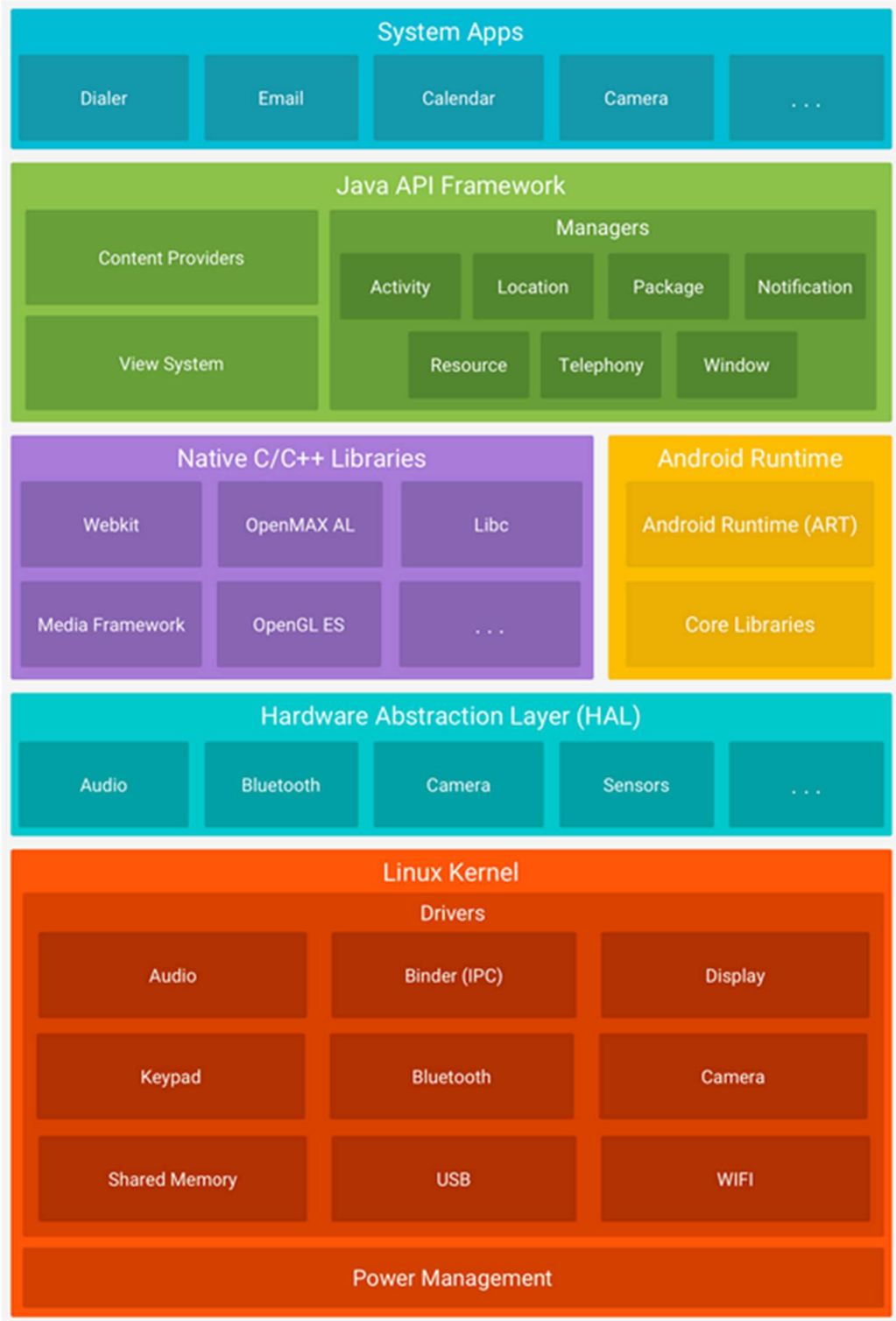


Další součástí Android architektury jsou nativní C/C++ knihovny. Ty obsahují základní funkce systému a poskytují přístup aplikacím k různým komponentám systému (Lacko, 2017). S touto vrstvou je často spojována vrstva Android Runtime (ART), která obsahuje virtuální stroj zajišťující komunikaci aplikací napsaných v jazyce Java s knihovnami v jazycích C a C++.

Nad knihovnami a ART se nachází aplikační framework. Tato vrstva také obsahuje aplikační knihovny, tentokrát ale napsané již v jazyce Java. Uživateli poskytuje možnosti práce s prvky systému a zařízení, jako jsou data o lokaci, data ze senzorů, či práce s Wi-Fi čipem (Gargenta, 2011). Dále tyto knihovny obsahují kousky kódu, znovupoužitelný software pro různé aplikace, jako jsou jednotné ovládací prvky a ikony. Aplikační framework obsahuje několik služeb, jako Package Manager (modul správy balíčků aplikací), View systém (správa prvků grafického UI), Window Manager (správa oken tvořených aplikacemi) a další (Lacko, 2017).

Poslední vrstvou architektury operačního systému Android je vrstva se systémovými aplikacemi. Zde se nachází všechny aplikace, se kterými uživatel přímo pracuje. Může se jednat o aplikace nativní, jako e-mailový klient, webový prohlížeč či kalendář, tak i aplikace stažené z internetu či obchodu s aplikacemi.

Obrázek 2 - Architektura Android OS (Android Developers)



3.1.3 Verze OS Android

Za dvacet let vývoje operačního systému Android tento systém prošel třinácti hlavními verzemi. Ačkoli pilně pracují na verzi Android 14, celosvětově nejrozšířenější verzí je stále Android 12 (25,29% zařízení) (Taylor, 2023). Při vývoji nových aplikací v programu Android Studio je navíc doporučováno aplikace vyvíjet se zpětnou kompatibilitou alespoň pro Android 7 (API 24), aby vyvíjená aplikace fungovala až na 94.4 % zařízení.

3.2 Material Design

Material Design obsahuje filozofii a způsoby vizuálního návrhu uživatelských rozhraní využívaných v systému Android. Jedná se o set předepsaných pravidel a doporučení designu, s jejichž použitím lze docílit vizuální jednotnost aplikací učených pro tento systém. Obsahuje návrhy nejčastěji používaných komponentů v operačním systému, jako jsou tlačítka, systémová oznámení a ikony, ale i doporučení ke tvaru bublin a hloubce stínů elementů, a také doporučení k celkovému rozvržení aplikace a jejích ovládacích prvků.

3.3 Škálovatelný design

Aby naše aplikace dosáhla velkého počtu uživatelů, je důležité, aby šla spustit na velkém množství zařízení. Vzhledem k tomu, že většina mobilních zařízení má různé rozměry, je důležité, aby šla aplikace zobrazit na různých rozměrech displejů a rozlišeních. Proto je zapotřebí používat škálovatelný design.

Při vývoji aplikace je důležité tvořit návrh uživatelského rozhraní v několika měřítkách, aby byla aplikace spustitelná a především použitelná na velké škále zařízení. Proto je vhodné při návrhu rozložení prvků využívat jejich rozmístění a vzdálenosti vůči ostatním systémovým prvkům a neurčovat jejich polohu pomocí absolutních rozměrů (Lacko, 2017).

I s tímto přístupem jsou ale absolutní rozměry důležité. Nejčastěji se používají jednotky palec či milimetr. Ale existují i jednotky, které jsou závislé na rozlišení displeje zařízení, jako px (rozměr uvedený v pixelech zařízení), nebo dp (abstraktní jednotka, 1 dp = 160 px/dpi).

Dalším důležitým bodem u škálovatelného designu je taky myšlenka na jeho přístupnost. Vzhledem k široké možnosti využití OS Android je potřeba myslet na to,

že aplikace by měla mít schopnost být obsluhována nejenom dotykem na chytrých telefonech, ale také například ovladačem u chytrých televizorů.

3.4 Základní pravidla designu

Aby byla aplikace použitelná, snadno přehledná a uživateli se s ní dobře pracovalo, je důležité dodržet několik základních pravidel jejího návrhu. Mezi ně lze zařadit též výše zmíněný Material Design. Tento design je většině uživatelů důvěrně blízký, již se s ním dříve setkali a umí s ním pracovat. I přesto ale musí být kladen důraz na účelnost, a ne pouze estetiku – jedinečný vzhled aplikace je důležitý pro tvorbu povědomí o značce a rozlišení aplikace od záplavy dalších, neměl by však být na úkor použitelnosti aplikace.

Material design doporučuje zarovnání prvků do mřížky podle delší strany displeje pro vytvoření konzistentního a strukturovaného rozvržení plochy aplikace (Lacko, 2017).

Dále je možné zapojit tzv. vizuální hierarchii. Jedná se o princip uspořádání prvků pro zvýraznění jejich hierarchie důležitosti (Interaction Design Foundation). Vhodným výběrem prvků, jejich vzhledu a umístění, lze navést uživatele, aby provedl vyžadovanou akci. Mezi základní principy vizuální hierarchie spadá například velikost písma, jeho barva a kontrast – větší, výraznější nápisy uživateli jasně signalizují vyšší důležitost. Důležité prvky v designu je vhodné umísťovat na viditelná a snadno dosažitelná místa, ale je také vhodné je nezahltit ostatními prvky designu – lidské oko tíhne k prvkům obklopeným dostatečně velkým prázdným prostorem (Interaction Design Foundation).

3.4.1 Design pro mobilní zařízení

Na vhodný design je potřeba klást důraz především při navrhování aplikací pro mobilní zařízení, zejména pro chytré telefony. Tato zařízení mají poměrně malý displej a je důležité, aby všechny ovládací prvky byly dostatečné velké a použitelné při ovládání dotykem prstů, ale také dostatečně malé, aby vizuálně nezahltily obrazovku zařízení. Také je třeba myslet na jejich vhodné rozmístění, vzhledem k tomu, že většina uživatelů používá svůj mobilní telefon ve svislé poloze a jednou rukou, tedy palcem.

Obrázek 3 - Vhodná místa pro ovládací prvky mobilní aplikace (Fleck, 2020)

Dalším důležitým aspektem je, že pokud uživatel používá mobilní zařízení, chce se dostat k informacím rychle a bez zbytečných informací navíc. Proto je důležité minimalizovat počet prvků na displeji a důležité prvky dostatečně zvýraznit, aby byly snadno viditelné a dosažitelné (Interaction Design Foundation).

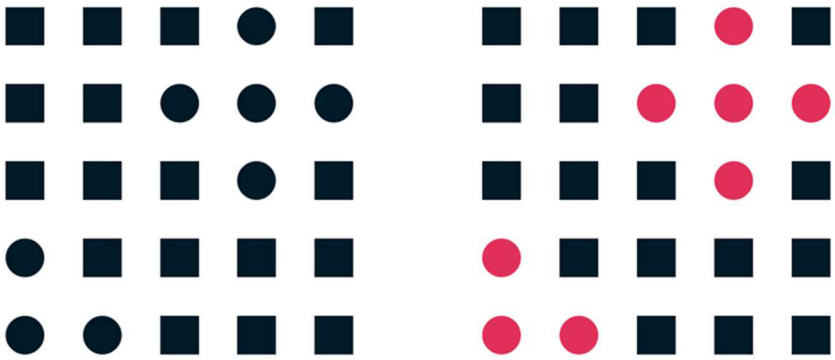
Dále je také důležité vzít v úvahu uživatelský vstup. Pokud vyžadujeme písemný vstup od uživatele, je třeba myslet na to, že onen vstup uživatel zadá pravděpodobně pomocí virtuální klávesnice. Ta často zabírá až půl obrazovky, proto informace důležité pro zadání vstupu musí být čitelné, i když máme k dispozici pouze část displeje.

3.4.2 Gestalt principy

Gestalt psychologie a její principy byly vytvořeny několika rakouskými a německými psychology začátkem 20. století. Základem této filozofie je poznání, že člověk nevnímá jako první jednotlivé prvky, které skládá do celku, ale naopak vnímá celek automaticky bez většího úsilí (Design do kapsy). Základních principů existuje osm. Jsou to následující:

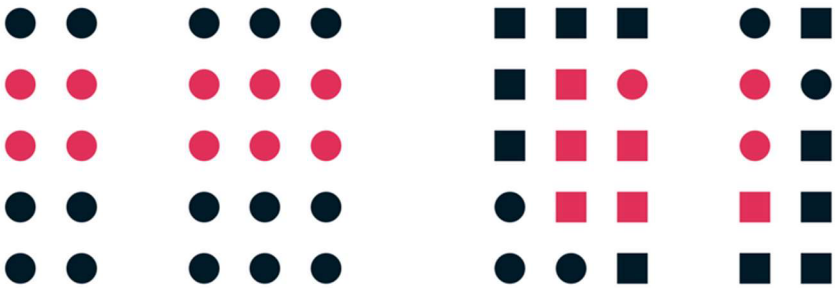
Princip podobnosti uvádí, že prvky podobné svojí velikostí, tvarem a barvou spolu budou v jisté kapacitě souviset. Tedy například primární tlačítka pro potvrzení v aplikaci by měla být stejně velká, tvarovaná a barevná.

Obrázek 4 - Princip podobnosti – prvky stejné velikosti, tvaru a barvy tvoří skupinu (Design do kapsy)



Princip blízkosti – související prvky se nachází ve vzájemné blízkosti, nesouvisející prvky by měly být oddělené.

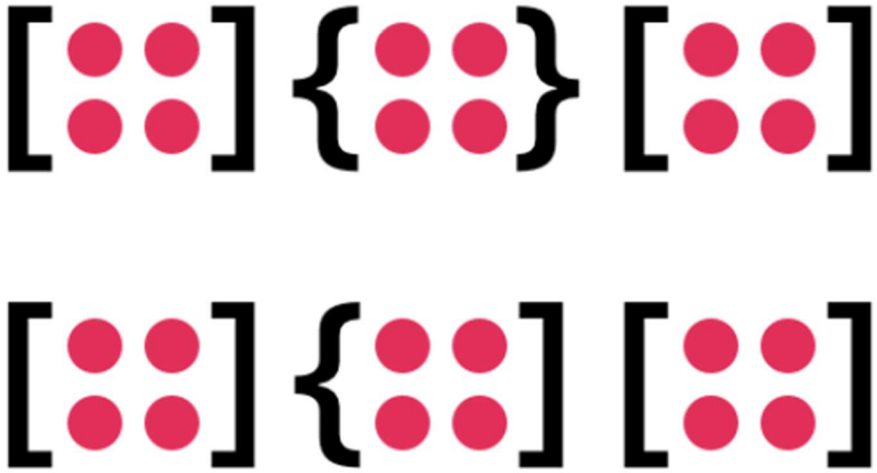
Obrázek 5 - Princip blízkosti – vzájemně blízké objekty tvoří skupinu (Design do kapsy)



Princip společného směru napovídá opět souvislost prvků, které se pohybují společným směrem a rychlostí.

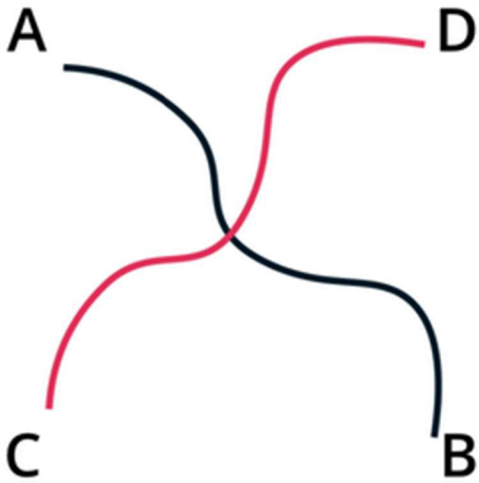
Princip symetrie rozmisťuje související prvky do symetrického uspořádání. Nejen, že uživatel bude vnímat takto rozmístěné prvky jako související, navíc je také bude vnímat jako estetické.

Obrázek 6 - Princip symetrie – symetrické prvky tvoří souvislou skupinu (Design do kapsy)



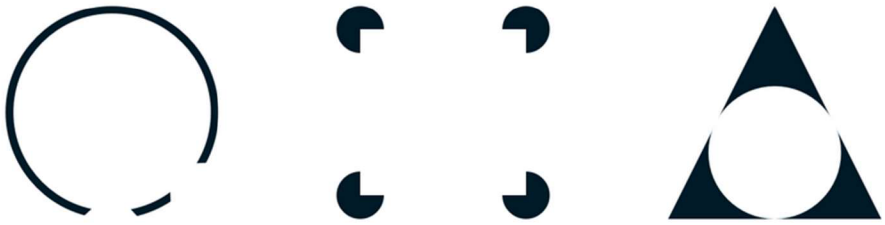
Princip návaznosti. Pokud uživatele navedeme od jednoho prvku k dalšímu, bude je vnímat jako související.

Obrázek 7 . princip návaznosti – propojené prvky jsou vnímané jako související (Design do kapsy)



Princip uzavření říká, že uživatel vnímá objekt jako celý i v případě jeho nekompletnosti, například nekompletní kružnici s vykousnutým bílým prostorem lidské oko stejně vnímá jako celistvou.

Obrázek 8 - Princip uzavření – vnímaný celek, i pokud není kompletní (Design do kapsy)



Princip popředí a pozadí – uživatel je poměrně snadno schopen rozeznat popředí od pozadí na základě vysokého kontrastu mezi objekty.

Obrázek 9 - Princip popředí a pozadí – vysoký kontrast a vnímání různých úrovní (Design do kapsy)

Princip známosti/smysluplnosti vyjadřuje schopnost vnímat jednotlivé prvky jako související a tvořící nějaký smysluplný objekt. Tento princip u člověka však vyžaduje jistou zkušenost. Člověk, který již dříve viděl dítě nakreslit obrázek slunce okamžitě rozezná, co kroužek s čárkami po obvodu znamená.

Obrázek 10 - Princip známosti/smysluplnosti – složení několika objektů do nám známého tvaru vytváří vnímání jejich souvislosti (Design do kapsy)

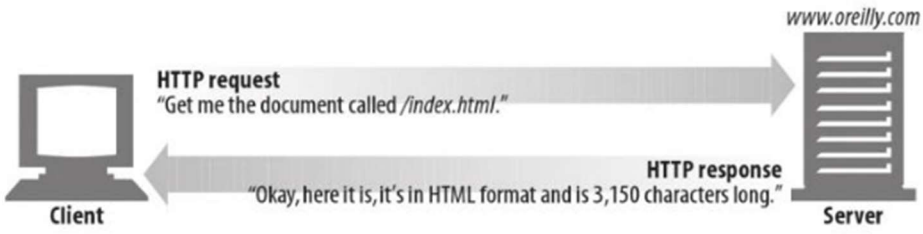
Na tyto principy je vhodné při návrhu uživatelského rozhraní aplikace pamatovat, aby vývojář lépe porozuměl lidskému chování a jak nejlépe uživateli dopomoci k vhodnému užívání aplikace.

3.5 Klient – server

Vzhledem k funkčnosti aplikace bude její implementace využívat model klient server. Jedná se o velmi rozšířenou architekturu implementace webových a mobilních aplikací a stránek. Tento model vlastně rozděluje funkce systému na dvě části – klientskou a serverovou, kde klientská část na sebe bere roli žadatele a server roli poskytovatele služby. Velmi častou implementací je, že klientem je frontendová aplikace poskytující rozhraní komunikace s uživatelem, do kterého uživatel může zadávat data a dotazy, a naopak také získávat informace a odpovědi na dotazy. Serverem je pak backendová aplikace komunikující s databází, která zadané dotazy zpracovává a vrací odpovědi. Tato výpočetní architektura má velikou výhodu v rychlosti zpracování uživatelských úkolů, protože výpočetní výkon je vlastně rozdělen mezi dva stroje. Klientská část modelu nemá velké požadavky na výkon, protože je často pouze zobrazovacím prostředkem a nevykonává žádné velké výpočetní úlohy (Oluwatosin, 2014).

Mezi klientem a serverem tedy musí docházet ke komunikaci. Nejčastějším způsobem komunikace je pomocí standardizovaných internetových komunikačních protokolů, jako jsou FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) a HTTP (Hypertext Transfer Protocol). Internetové protokoly jsou velmi důležité pro zjednodušení komunikace. Protože jeden server mlže obsluhovat několik klientů, internetový protokol zajišťuje, aby se odpověď na žádost vrátila správnému tazateli (klientu). Dále také může zajišťovat bezpečnost přenosu dat (HTTPS) (Gourley, et al., 2002).

Obrázek 11 - Komunikace klient – server (Gourley, et al., 2002)



3.6 Mapy

Jedním z hlavních stavebních kamenů aplikace určené pro doručování zboží je integrace mapových a navigačních služeb.

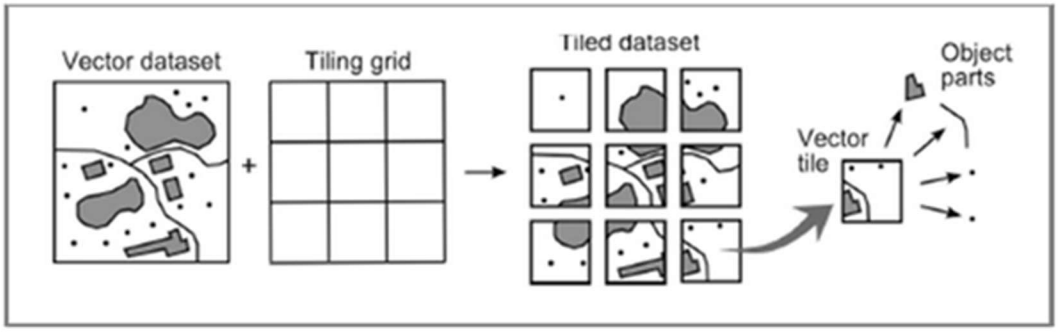
3.6.1 Vektorové vs. rastrové mapy

Pro zobrazení mapových dat lze použít buď vektorové, nebo rastrové dlaždice. Ty se liší v několika zásadních vlastnostech. Tyto dlaždice, neboli části map, Goodchild (Goodchild, 1990) přirovnává ke stránkám papírového atlasu – rozdělují celek mapy na menší čtverce.

Rastrové dlaždice zobrazují mapy pomocí rastrových obrazů. Dlaždice map jsou tvořeny rozdělením požadované oblasti na čtyři části, označené číslem (MapTiler). Každá z těchto čtyř dlaždic se při jejím přiblížení chová jako nový celek a je rozdělena na další čtyři dlaždice. Každá nová dlaždice má svoje unikátní koordinace. Zavadil (2013) uvádí, že většina webových služeb užívá velikost dlaždic 256 × 256 pixelů. Peterson (2012) odhaduje velikost potřebné paměti pro uložení takové dlaždice na 15 kB. S každým stupněm přiblížení se počet dlaždic potřebných pro zobrazení celé mapy čtyřikrát násobí. Při dvacáté úrovni přiblížení vzniká zhruba jeden bilion dlaždic. Takové množství by vyžadovalo zhruba 20 480 TB paměti. Pro snížení požadavků na paměť užívají poskytovatelé mapových služeb cache, tedy ukládání některých mapových dat a jejich znovupoužití, pokud je vyžaduje nový uživatel. Taková data ale nejsou příliš užitečná u mapových částí, které mají potenciál se často měnit. Vzhledem k tomu, že mapy jsou uložené v rastrovém formátu, při změně jejich malé části se musí dlaždice generovat znovu, což příliš nepřispívá rychlosti jejich použití. Rastrové mapy navíc nedovolují uživateli interagovat s jednotlivými objekty při jejich webovém zobrazení (Noskov, 2018).

Vektorové mapy pro zobrazování dat také používají systém dlaždic jako rastrové mapy, ovšem geografické údaje jsou uložené ve formě vektorů. To znamená, že každý objekt se skládá z různých bodů, čar a polygonů, a ne z pevného počtu pixelů, jako v rastrových formátech. Pokud dojde ke změně některého objektu, v mapách dochází ke změně pouze dotyčného vektorového objektu, a tedy není nutné generovat nový set dlaždic.

Obrázek 12 - Proces generování vektorových dlaždic (Gaffuri, 2012)



Vektorové mapy nejsou ale pokaždé rychlejší a vhodnější k použití než rastrové mapy. Mají ale nespornou výhodu v menším zatížení sítě, protože stahují menší množství dat (Netek, et al., 2020)

3.7 Nejkratší cesta

Aby bylo možné zákazníkovi dovézt zásilku v co nejkratším čase, je třeba nalézt nejkratší či nejrychlejší cestu ke zvolenému cíli. Pokud si představíme místa na mapě, kam musí řidič během své cesty zajet, jako vrcholy grafu a cesty, které musí projet, jako jeho ohodnocené hrany, lze využít některý z algoritmů pro vyhledání nejkratší cesty v grafu. Ohodnocování hran ovšem záleží na tom, co si pod pojmem „nejkratší cesta“ představíme. Samozřejmě se může jednat o cestu opravdu nejkratší, tedy máme za cíl ujet co nejméně kilometrů. Taková cesta nemusí být ale vždy ta nejlepší, například pokud projíždíme centrem města, kde často bývají dopravní zácpy. V takovém případě lze vybrat cestu nejrychlejší. Některé služby dokonce nabízejí upozornění na zácpy v reálném čase, jako třeba Google Maps, která sbírá data o pohybu jejich uživatelů.

Pokud ale cestujeme například na kole a ne automobilem, budeme možná také uvažovat cestu jinou. Pro použití se nám otevřou například pro vozidla jednosměrné komunikace či některé pěší komunikace, ale musíme brát v potaz například výškový profil trasy.

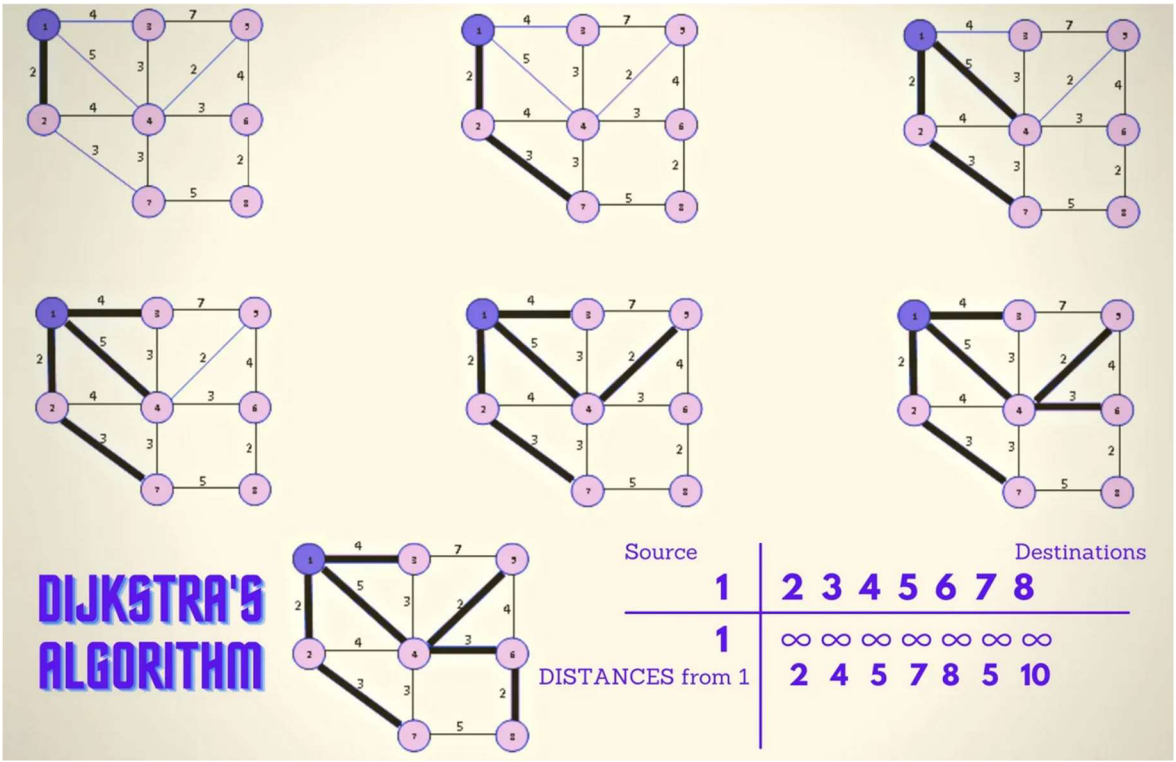
Pro vyhledání nejkratší cesty v grafu existuje několik technik a algoritmů. Nejčastěji použitými algoritmy jsou Dijkstrův algoritmus, A* algoritmus a Floyd-Warshallův algoritmus (Cormen, et al., 2009).

3.7.1 Dijkstrův algoritmus

Dijkstrův algoritmus byl vytvořen v 50. letech 20. století jako způsob pro vyhledání nejkratší cesty (cesty s nejnižší cenou) v grafu. Dijkstrův algoritmus lze využít pro vyhledání nejkratší cesty mezi libovolnými dvěma body v hranově ohodnoceném grafu. Algoritmus probíhá následovně (Lumen Learning):

1. Současný vrchol označíme jako navštívený,
2. Nalezneme všechny hrany vycházející ze současného vrcholu, zaznamenáme dosavadní cenu cesty, pokud je menší než předchozí hodnota,
3. Vybereme hranu s nejlepším ohodnocením a přesuneme se do souvisejícího vrcholu,
4. Algoritmus opakujeme, dokud nedosáhneme koncového vrcholu.

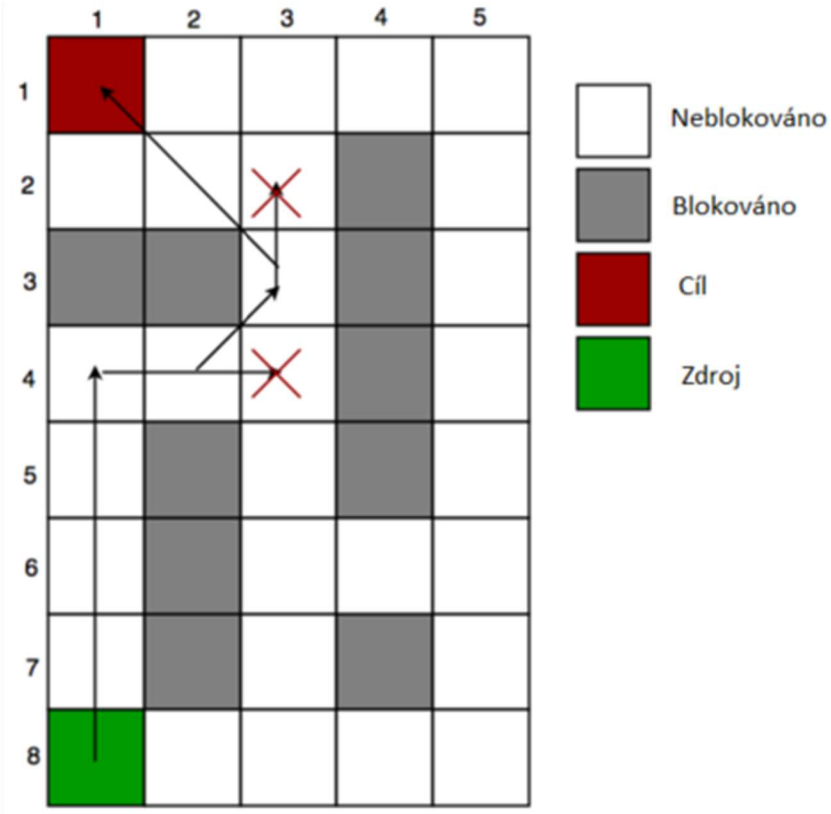
Obrázek 13 - Průběh Dijkstrova algoritmu (STechies)



A* algoritmus

Na A* algoritmus lze nahlížet jako na rozšíření Dijkstrova algoritmu o heuristické principy. Na rozdíl od Dijkstrova algoritmu ale vyhledává nejkratší cestu pouze k jednomu předem určenému vrcholu, a ne ke všem možným. V tomto algoritmu je vytvořena a udržována prioritní fronta ještě nenavštívených uzlů. Každému uzlu je přiřazena priorita v závislosti na minimalizačním kritériu (nejkratší cesta, nejmenší čas apod.). Uzly z prioritní fronty jsou následně postupně odebírány a fronta je zároveň doplňována o další sousedící uzly současného vrcholu (Hart, et al., 1968).

Obrázek 14 - Průběh A algoritmu (GeeksforGeeks, 2023)*



3.7.2 Floyd-Warshallův algoritmus

Tento algoritmus nalézá nejkratší cestu mezi všemi páry vrcholů v grafu za využití matice vzdáleností. Algoritmus postupně prochází hrany grafu a vylepšuje odhad nejkratší cesty (Cormen, et al., 2009). Je mírně náročnější na čas výpočtu, ale je vhodný pro aplikace, kde je třeba nalézt nejkratší cesty mezi všemi páry vrcholů.

3.8 Mikrologistika

V současné době je řízení dodavatelského řetězce stále obtížnější. Aby si společnosti udržely konkurenceschopnost, musí optimalizovat své logistické operace a zajistit rychlé a efektivní dodání svých výrobků zákazníkům. Jeden ze způsobů, jak tohoto docílit, jsou mikrologistické postupy. Tyto postupy jsou důležité zejména při obsluze první a poslední míle, tedy hlavně při obsluze koncových zákazníků.

Cílem mikrologistiky je optimalizovat logistické operace na místní úrovni, což může pomoci snížit náklady na dopravu, zvýšit rychlost dodávek a zlepšit celkovou spokojenost zákazníků.

Často je jediným úkolem řidiče vyzvednout zásilku v bodě A a dovézt ji do bodu B. V takovém případě je otázka mikrologistiky snadno vyřešená. Obvykle ale dojde k tomu, že řidič na své trase musí objet více míst a obsloužit více zákazníků. Pak je třeba prvotní algoritmy hledání nejkratší cesty rozšířit o další kroky pro nalezení nejoptimálnější celkové trasy.

Když máme vypočtené všechny nejlepší cesty mezi vrcholy uvažovaného grafu, lze pokračovat jedním z několika následujících postupů.

3.8.1 Problém obchodního cestujícího

Problém obchodního cestující je optimalizační problém hledající nejkratší možný způsob, jak obejít všechna požadovaná místa na mapě a vrátit se do výchozího bodu. Jedná se vlastně o vyhledání nejkratší hamiltonovské kružnice v hranově ohodnoceném grafu, kde hamiltonovská kružnice je taková posloupnost vrcholů grafu, která obsahuje každý jeho vrchol právě jednou (Šubrt, et al., 2022). Jedná se o NP obtížnou úlohu, tudíž je poměrně složité nalézt obecný algoritmus pro její řešení, zahrneme-li do zadání větší množství míst k navštívení – složitost řešení narůstá s jejich množstvím – pro deset míst je počet možných tras $3{,}6 \times 10^5 = 362\ 880$, ale pro dvacet míst je jich již $2{,}4 \times 10^{18}$ = téměř dva a půl tisíce biliard tras (Cormen, et al., 2009). Existují však různá řešení a nástroje pro řešení tohoto problému, která sice nejsou optimální, ale lze je vyřešit v poměrně krátkém čase.

3.8.2 Vehicle routing problem

Další optimalizační úlohou řešenou v mikrologistice může být Vehicle routing problem (VRP) – Problém směrování vozidel. Zabývá se nalezením vhodné množiny cest pro flotilu vozidel, které obsluhují množinu zákazníků z jednoho či více skladů. Vhodná cesta je cesta s nejmenší cenou, jako například nejmenší vzdálenost či čas. Jedná se vlastně o generalizaci Problému obchodního cestujícího (Dantzig, et al., 1959). Používá se například v případě společnosti přiřazující trasy rozvorů svým řidičům. Ne vždy je ale možné obsloužit všechny zákazníky. Pro tyto případy lze zákazníkům přiřadit prioritu. Navíc je důležité myslet i na další omezení – například vozidla mohou mít kapacitu pro maximální hmotnost nebo objem položek, které mohou převážet, nebo se od řidičů může vyžadovat, aby navštívili místa v určitých časových oknech požadovaných zákazníky.

VRP může mít několik variací:

VRP s kapacitními omezeními, ve kterém mají vozidla maximální kapacity pro položky, které mohou přepravovat.

VRP s časovými okny, kdy vozidla musí navštívit místa v určených časových intervalech.

VRP s omezeními zdrojů, jako je prostor nebo personál pro nakládku a vykládku vozidel ve skladu

VRP s vynechanými návštěvami zákazníků, kdy vozidla nemusí navštívit všechna místa pod podmínkou penalizace.

Podobně jako u Problému obchodního cestujícího, i VRP jsou velmi špatně obecně řešitelné. Doba potřebná pro jejich řešení roste exponenciálně s velikostí problému (Google Developers, 2023). U dostatečně velkých problému může nalezení optimálního řešení trvat i roky. Pro jejich řešení tak vznikají různé nástroje (knihovny), které do jisté míry VRP řešit umí, ale ne vždy vrací optimální řešení.

3.8.3 Job shop scheduling

Dalším nástrojem používaným v mikrologistice je Job Shop Scheduling – rozvrhování úloh. Jedná se o rozdělení *n* úloh s různou dobou zpracování mezi *m* strojů. Každá úloha se skládá z několika po sobě jdoucích úkolech zpracovávaných na různých strojích. Job shop scheduling se zabývá řešením vhodné posloupnosti řešení úkolu, aby došlo k minimalizaci

celkové délky průběhu úlohy (dokončení všech nutných úkolů) (Google Developers, 2023). Průběh zpracování má ale následující omezení:

- Žádný úkol nesmí započít před dokončením předchozího úkolu,
- Stroj může zpracovávat pouze jeden úkol v jeden čas,
- Započatý úkol musí být zpracován až k dokončení.

3.9 Základy vývoje aplikací

Proces vývoje mobilní aplikace probíhá v několika krocích (Panjuta, 2022):

- Plánování a návrh – nejprve je třeba navrhnout funkce a vzhled plánované aplikace. Často je vhodné začít prostě s papírem a tužkou a nakreslit si různé obrazovky s poznámkami o jejich funkčnostech. Již v této počáteční fázi je ale důležité mít jasno o cílové skupině uživatelů a jejich požadavcích, například pomocí využití person.
- Vývoj – zde dochází již k vlastnímu programování aplikace. Před samotným programováním aplikace ale lze také zahrnout fázi prototypování, buď pomocí online prototypovacích nástrojů, nebo jako součást vývoje ve vybraném programovacím jazyce.
- Testování – Na tuto fázi lze také nahlížet jako na součást fáze přechozí, nebo alespoň částečně. Testování v průběhu vývoje je důležitou součástí tvorby aplikací. Často je vhodné otestovat každý jednotlivou verzi vznikajících prototypů (Svobodová, 2020). Na závěr je samozřejmě vhodné provést finální důkladné testování hotové aplikace před její publikací.
- Nasazení – finální fází je pak nasazení aplikace. Její umístění záleží na jejím typu – v případě aplikace pro Android OS se jedná o Google Play Store, odkud si aplikaci mohou uživatelé stáhnout.

Pro zjednodušení vývoje aplikace existují různé široce využívané nástroje. Nejdůležitějším z nich je Android Studio, oficiální nástroj pro vývoj aplikací pro Android OS. Poskytuje určité množství nástrojů a vlastností, včetně editoru kódu a editoru uživatelského rozhraní.

Dalším nástrojem je Android SDK – Android Software Development Kit. Ten poskytuje uživatelům sadu nástrojů a knihoven pro vývoj aplikací (Android Developers, 2023).

Google Firebase je také jeden z nástrojů vhodný pro využití při vývoji aplikací. Na rozdíl od předchozích dvou uvedených je možné ho využít pro vývoj aplikací nejen pro Android OS, ale i pro iOS a web. Nabízí různé funkce, jako například cloudovou databázi podporující offline přístup a automatickou synchronizaci, cloudové úložiště pro data uživatelů aplikace a přidání ověřovacích služeb do aplikace, jako třeba ověření pomocí účtu na Google, Facebook nebo Twitter (Google Firebase, 2023).

4 Vlastní práce

Tato diplomová práce z části navazuje na autorčinu bakalářskou práci Návrh a implementace modelu a prototypu mobilní aplikace (Svobodová, 2020). V této bakalářské práci byly prozkoumány současné možnosti prototypování a byl vytvořen prototyp mobilní aplikace zaměřen na e-commerce, rozšiřující dosavadní služby nabízené firmou provozující půjčovnu šatů. Klíčovou součástí této služby je dovezení objednaných šatů zákazníkovi v co nejkratším čase.

Tato práce rozšiřuje předchozí prototyp o business logiku a backendovou aplikaci pro zaměstnance.

Jak již bylo dříve uvedeno, tato diplomová práce se zabývá vývojem mobilní aplikace. Jedná se o klasický model klient-server aplikace. Vzhledem k tomu, že již existuje velké množství služeb, které nabízejí řešení pro vývoj e-commerce mobilních aplikací, bude tato práce zaměřená spíše na backendovou část a řešení logistiky.

I logistika rozvozu objednaného zboží může být řešena hned několika způsoby. Jedním z často využívaných způsobů je outsourcing logistických služeb, tedy že firma pro řešení logistiky využívá služeb přímo logisticky zaměřených poskytovatelů služeb. Tyto služby mají buď svou vlastní flotilu kurýrů, nebo může využívat kurýry ze široké veřejnosti, kteří se přihlásili jako zprostředkovatelé kurýrních služeb. V tomto případě má každý z kurýrů verzi aplikace, která nabízí seznam zásilek čekajících na doručení. Kurýr si pak vybere zásilku z jeho okolí, kterou vyzvedne a zároveň obdrží informace o místě jejího doručení. Zásilku pak doručuje pomocí aplikací třetích stran, jako běžné navigační aplikace v jeho mobilním zařízení (Google Maps, HERE Maps a další), navigace v automobilech a podobně. Toto řešení ke vhodné pro firmy, které vyžadují velké množství kurýrů, kteří jsou schopni rychle obsloužit velkou oblast, jako jsou celá velkoměsta nebo i státy. Problém však může nastat, pokud žádný z kurýrů není k dispozici nebo v blízkém okolí vyzvednutí zásilky.

E-commerce platformy však mohou vyvíjet i vlastní aplikaci pro kurýrní služby či vlastní flotilu kurýrů, která samostatně vyhledá nejlepší trasu pro kurýra na základě implementované výpočetní logiky pro řešení problémů nejkratší cesty.

Další část mobilní aplikace může být vyvinutá například pro mezisklady, v tomto konkrétním případě čistírny oděvů, aby mohly firmu informovat o stavu skladu nebo i stavu položek (vyčištěno/nevyčištěno/připraveno k odvozu).

4.1 Business logika

Tato práce zkoumá některé aspekty vývoje mobilní aplikace Dressaster, která bude rozšiřovat služby módního salonu. Tento salon se zaměřuje na půjčování a prodej společenských šatů. Šaty, které si zákazník objedná, mu jsou v krátkém čase dovezeny, často přímo na akci, jako náhrada za současné šaty zákazníka. Dovezené šaty si pak zákazník buď může za sníženou cenu ponechat, nebo je vrátit. Pokud si zákazník přeje vrácení, šaty opět vyzvedne kurýr a odveze je do čistírny, kde jsou pak vyčištěné šaty týden uschované, a tak čistírna může sloužit jako mezisklad. Po týdnu kurýr odváží veškeré vyčištěné šaty zpět do salonu, kde čekají na další zápůjčku.

Cílová skupina, jejíž vymezení je velmi důležitým krokem ve vývoji aplikací, byla již dříve definována v autorčině bakalářské práci. Vzhledem k funkci a nabídce vytvářené aplikace (prodej a zápůjčka společenských šatů), byla cílová skupina stanovena jako ženy a dívky mladšího a středního věku, ideálně uživatelé seznámené s používáním chytrých telefonů s operačním systémem Android (Svobodová, 2020).

Pro objasnění business logiky systému byl vytvořen UML model s následujícími diagramy. Tyto diagramy jsou vytvořeny pomocí objektově orientovaných přístupů.

4.1.1 Objektově orientované modelování

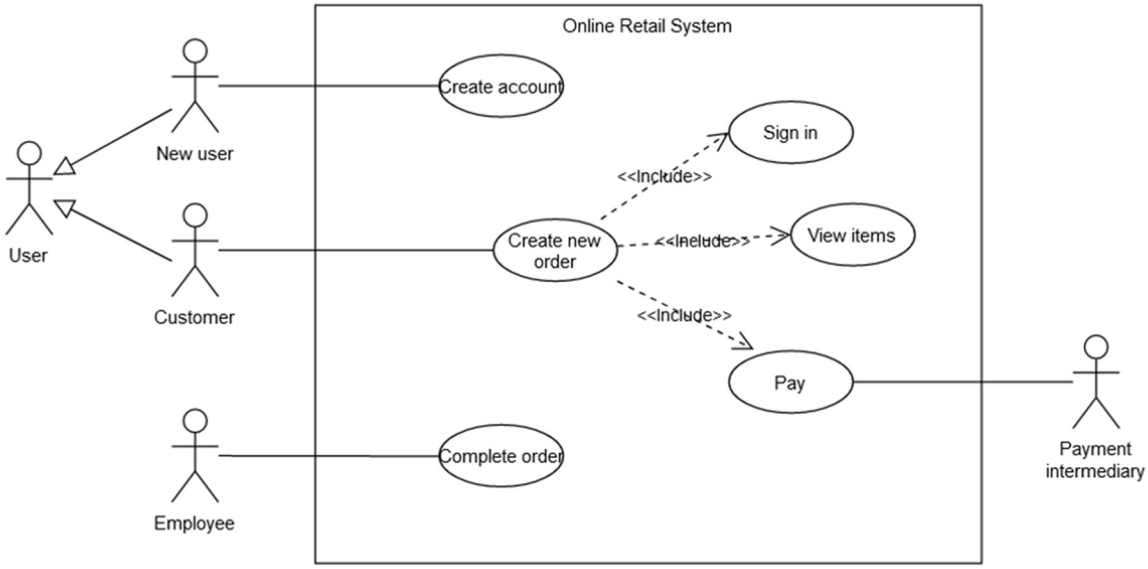
Tento modelovací přístup využívá diskrétní objekty pro popis a analýzu modelů. Tyto objekty jsou definovány pomocí UML (Unified Modellig Language), který popisuje jak datovou strukturu, tak chování objektů. Je založen na sadě metamodelů. Nicméně, sémantické integrační principy v UML nejsou zcela jasně vymezené (Gustas, 2011).

Objekt je entita s definovanou rolí, a může být seskupen do tříd na základě jeho struktury a chování.

4.1.2 Use case diagram

Use case je souborem interakcí a funkcí vykonávanými systémem, zatímco interaguje se svými uživateli (Gustas, 2011). V následujícím use case diagramu jsou znázorněni tři aktéři – Uživatel, Zaměstnanec a Zprostředkovatel platby. Aktér Uživatel má navíc dva potomky – Nový uživatel a Zákazník.

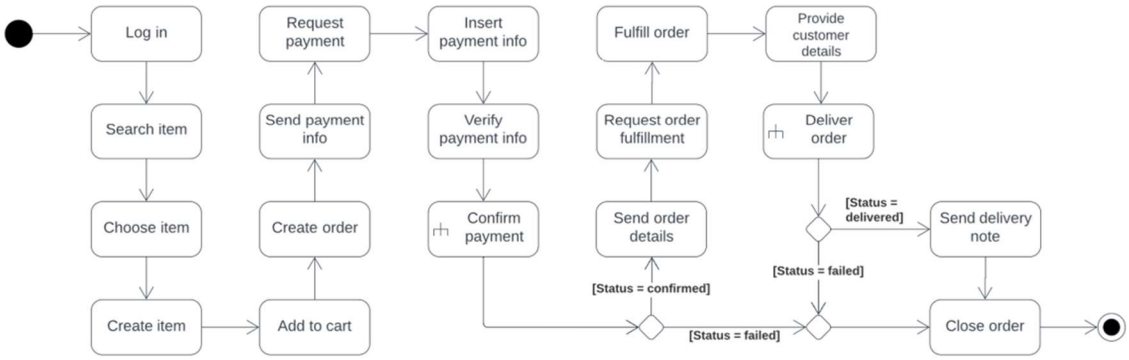
Obrázek 15 - Use case diagram



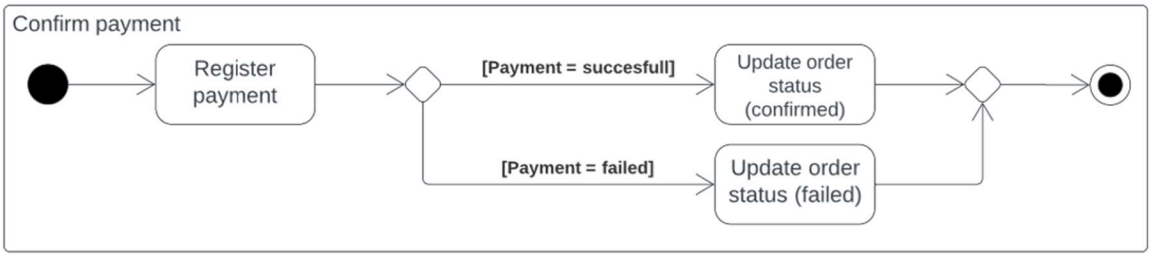
4.1.3 Digramy aktivit

Na rozdíl od use case diagramu, diagram aktivit zobrazuje dynamické aspekty systému. Zobrazuje sekvenci aktivit prováděných objekty v systému (Jena, et al., 2014). Níže jsou uvedené tři diagramy – hlavní diagram aktivit a dva subdiagramy zobrazující aktivity Potvrdit platbu a Doručit objednávku v bližším detailu.

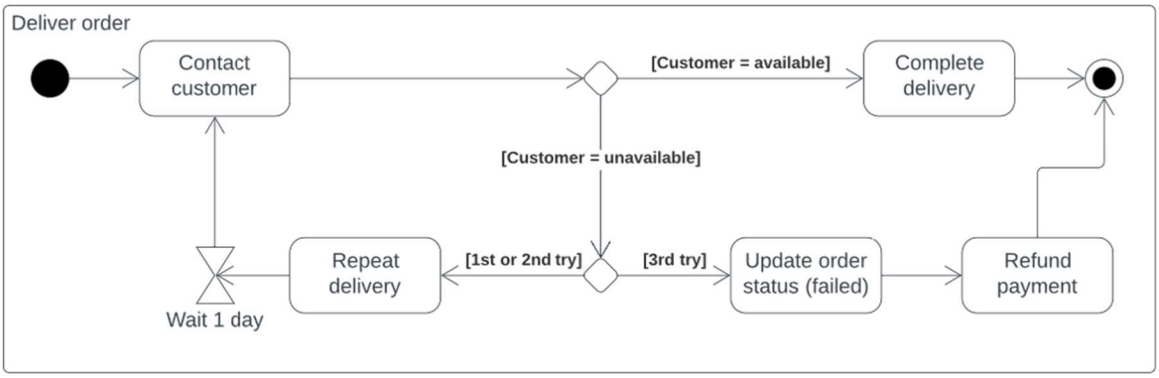
Obrázek 16 - Hlavní diagram aktivit



Obrázek 17 - Subdiagram pro aktivitu Potvrdit platbu



Obrázek 18 - Subdiagram pro aktivitu Doručit objednávku

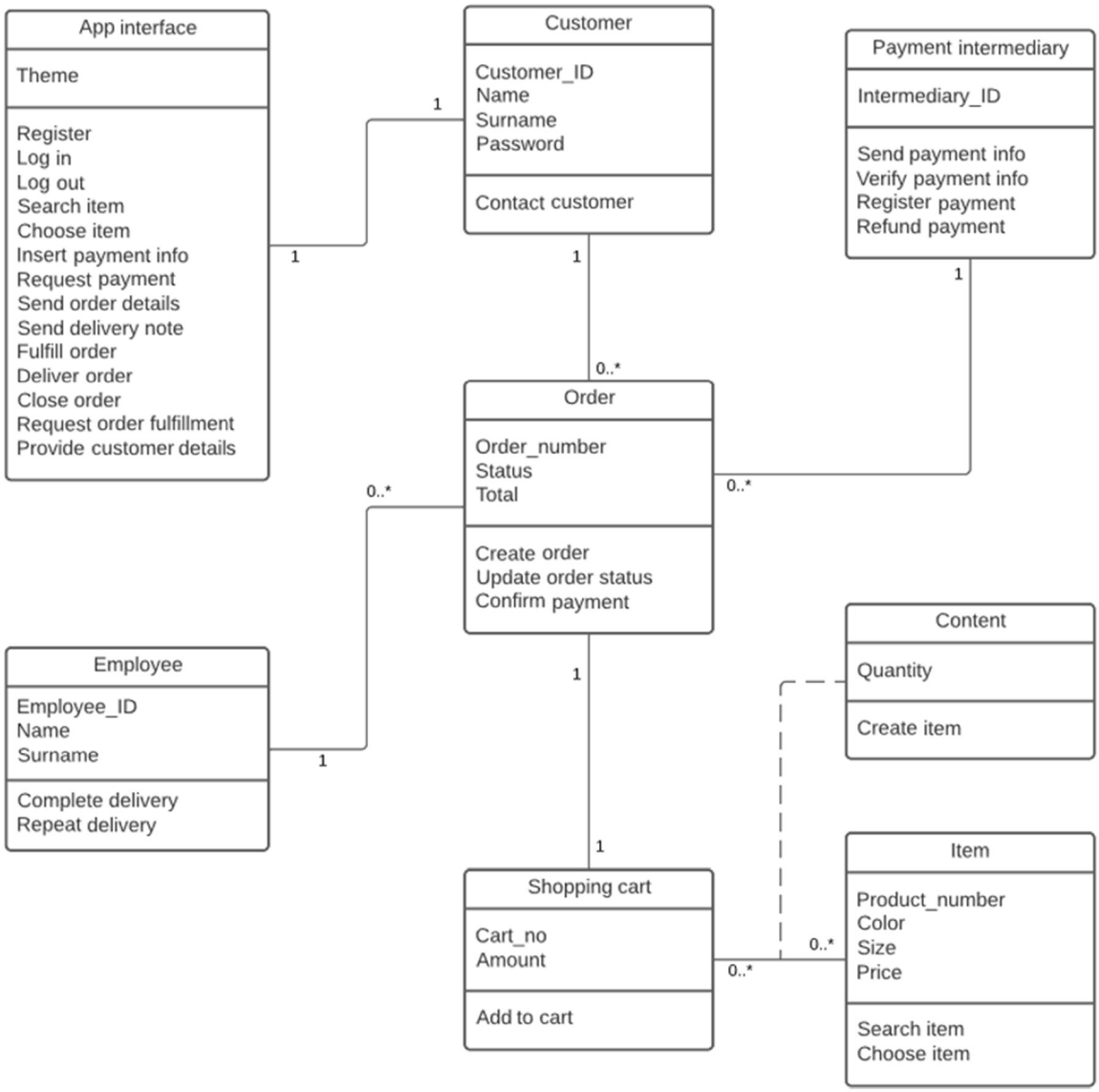


4.1.4 Diagram tříd

Třídy jsou skupiny objektů, které mají stejné vlastnosti. Každá třída má atributy s hodnotami a operace, které s těmito hodnotami pracují (Szlenk, 2006). Diagram tříd zobrazuje statický pohled na systém (Gustas, 2011).

Následující diagram zobrazuje návrh tříd systému s jejich možnými hodnotami a operacemi.

Obrázek 19 - Diagram tříd



4.2 Vývoj aplikace

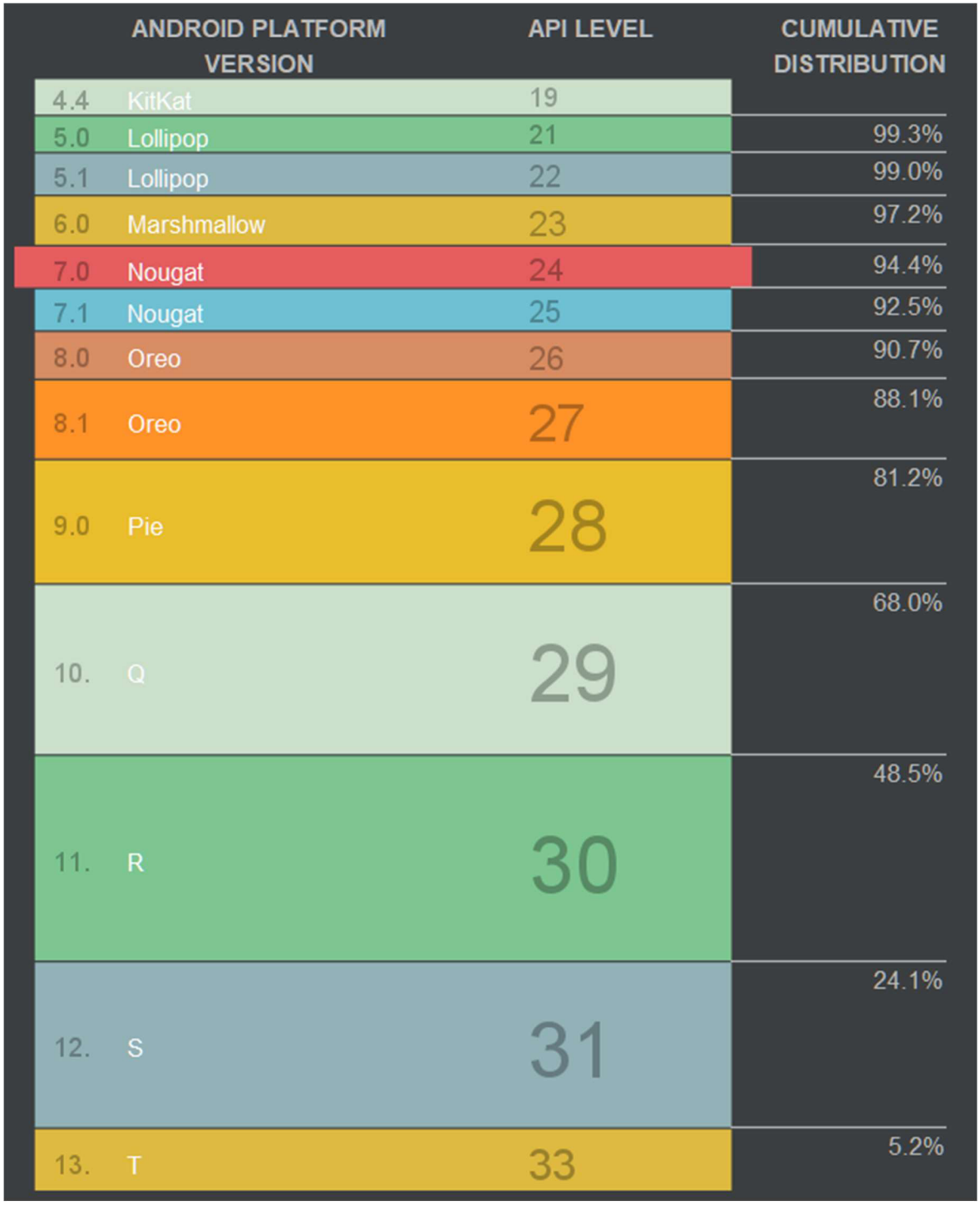
V této části diplomové práce se budeme zabývat možnostmi vývoje mobilní aplikace. Vzhledem k zadané business logice by bylo možné vyvíjet vlastně aplikace dvě – jednu pro koncového zákazníka a jednu pro zaměstnance a kurýry.

Vzhledem k tomu, že aplikace bude určena pro operační systém Android, bude vyvíjena v softwaru Android Studio. Tento software nabízí dva jazyky vývoje – Kotlin a Java. Kotlin je poměrně nový programovací jazyk (vznikl v roce 2011) běžící nad Java Virtual Machine.

4.2.1 Android Studio

Android Studio je oficiální IDE (Integrated Development Environment) pro vývoj aplikací pro Android OS. V současné době je nabízeno v poslední verzi, Electric Eel (Android Developers, 2023). Při započetí vývoje nové aplikace Android Studio nabízí možnosti volby zpětné kompatibility. Doporučuje použití zpětné kompatibility až k Android 7.0 Nougat, aby bylo možné spustit aplikaci až na 94,4 % zařízení (Obrázek 20).

Obrázek 20 - Distribuce verzí Android OS k 6.1.2023 (Android Studio, 2023)



ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
4.4	KitKat	19	
5.0	Lollipop	21	99.3%
5.1	Lollipop	22	99.0%
6.0	Marshmallow	23	97.2%
7.0	Nougat	24	94.4%
7.1	Nougat	25	92.5%
8.0	Oreo	26	90.7%
8.1	Oreo	27	88.1%
9.0	Pie	28	81.2%
10.	Q	29	68.0%
11.	R	30	48.5%
12.	S	31	24.1%
13.	T	33	5.2%

4.2.2 Aktivity

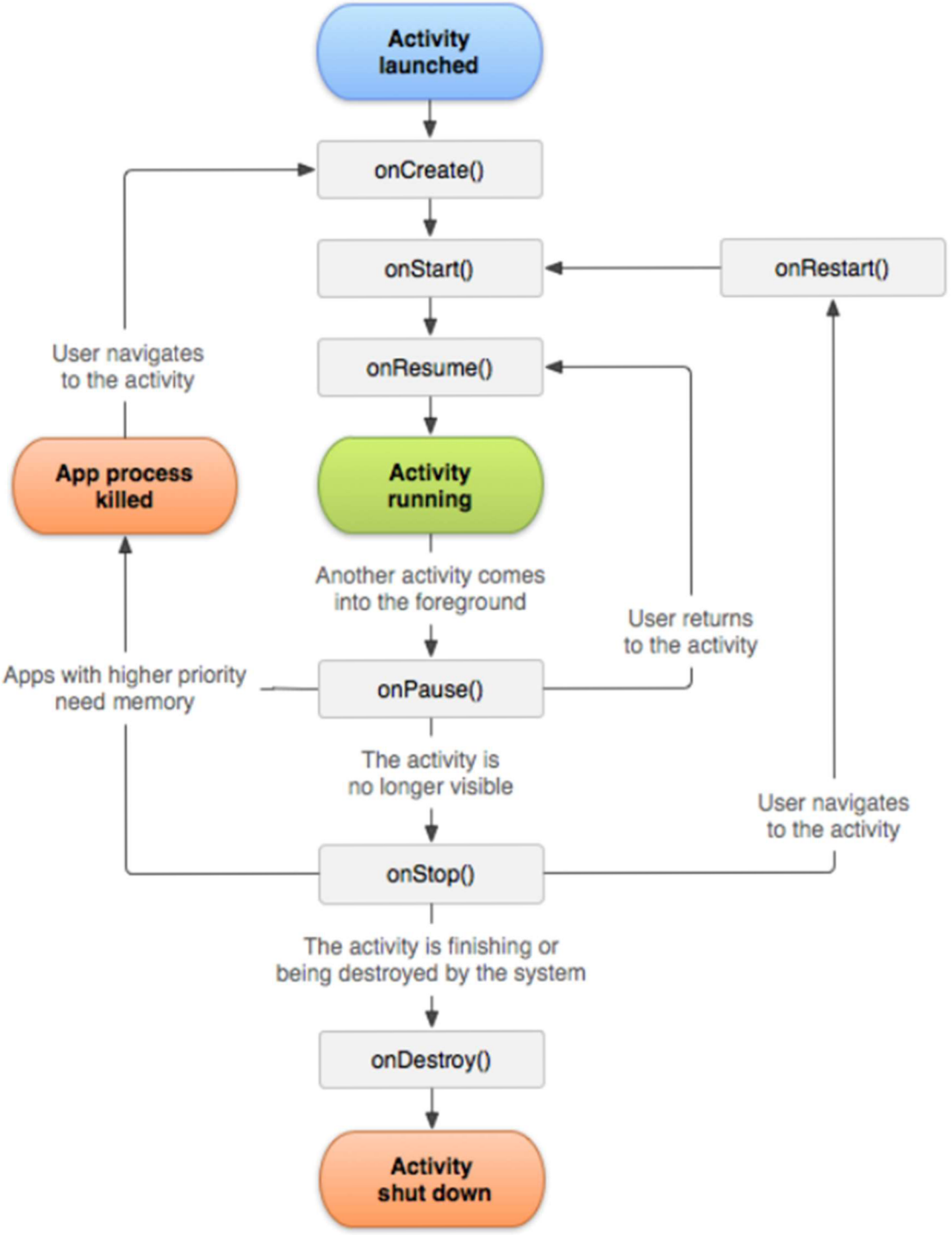
Jeden z hlavních pilířů vývoje aplikace pro Android je třída Aktivit. Některé jednoduché aplikace mohou obsahovat aktivitu pouze jednu, je ale běžné, že se jich v aplikaci vyskytuje více a že mezi sebou komunikují a předávají si informace. A nekomunikují pouze mezi sebou, ale také s uživatelem přes uživatelské rozhraní aplikace (Lacko, 2017). Android Studio při tvorbě nového projektu nabízí několik různých přednastavených aktivit, jako jsou obrazovka přihlašování nebo mapa. Rozhodně by každá aktivita měla být zaměřena pouze na jednu věc, kterou chce uživatel vykonat, jako vyplnění formuláře či odeslání zprávy. Na aktivitu lze tedy nahlížet jako na jedno okno aplikace. Toto pravidlo je povoleno porušit pouze u zařízení s velkými displeji, jako jsou tablety, které mohou zobrazovat nejen okno psaní zpráv, ale třeba i seznam konverzací, tedy seznam typu master-detail.

Aktivita může procházet několika stavy v průběhu jejího životního cyklu. Tři hlavní stavy jsou:

- Na popředí – aktivita má fokus, interaguje s uživatelem,
- Pozastavená – s uživatelem neinteraguje, ale je stále v paměti zařízení,
- Zastavená – kompletně překrytá jinou aktivitou, ale je stále v paměti zařízení.

Mezi těmito stavy může aktivita přecházet pomocí metod, neb může také úplně skončit.

Obrázek 21 - Životní cyklus aktivity a přechody mezi stavy (Google Developers, 2023)



4.2.3 Služby

Dalším důležitým stavebním kamenem při vývoji aplikace pro Android OS jsou služby. Na rozdíl od aktivit služby probíhají na pozadí a mohou trvat i déle. Nevyžadují uživatelské rozhraní. Typickým příkladem služeb může být přehrávání hudby na pozadí (Lacko, 2017).

4.2.4 Zdroje

Zdroje obsahují důležité části definice objektů. Jedná se vlastně o takovou knihovnu textových řetězců, použití barev, tvarů a dalších. Při vývoji je důležité tyto zdroje využívat, aby se zjednodušila práce s rozhraním a došlo k jeho sjednocení. Pokud chceme měnit barvu potvrzovacího tlačítka, neměníme ji přímo u konkrétní instance objektu, ale v souboru zdrojů, aby ke stejné změně došlo u všech potvrzovacích tlačítek v aplikaci.

4.3 Řešení logistiky

Pro ověření možností užití mikrologistických metod pro plánování cest v reálné aplikaci byla vytvořena sada testovacích metod pro některé obvyklé situace řešení pomocí metod pro řešení problémů směrování vozidel. Tato sada byly vytvořena jako aplikace v jazyce Java. Volba jazyka Java je dána především univerzalitou jeho reálných použití. Lze v něm standardně vytvořit aplikaci běžící na serveru a poskytující rozhraní pro klientské aplikace běžící na mobilních zařízeních se systémy Android i iOS a současně vytvořit aplikaci, které spolupracuje s knihovnami pro řešení obecných optimalizačních nebo logistických dopravních problému. Takové knihovny jsou často vytvořeny v jazycích jako je C++. Důvodem pro volbu jazyka C++ pro optimalizační knihovny nebo knihovny pro routování a řešení přepravních problémů je potřebný výpočetní výkon. Jedná se o algoritmy, které jsou výpočetně velmi náročné i pro poměrně malý rozsah vstupních dat. Knihovny v jazyce C++ však lze z jazyka Java standardně volat a kombinace aplikační logiky napsané v jazyce Java a výpočetní knihovny napsané v jazyce C++ je velmi obvyklá.

Aplikace byla vytvořena jako projekt sestavovaný (build) pomocí nástroje Apache Maven, tedy jako takzvaný Maven projekt. Pro vývoj bylo použito vývojové prostředí (IDE) Apache Netbeans. Základní kostra aplikace byla automaticky vygenerována pomocí IDE s volbou Java Application.

Pro ověření metod směrování vozidel v této práci byla použita sada nástrojů Google Operation Research Tools (neboli OR-Tools). OR-Tools je rychlý a přenositelný softwarový balík s otevřeným zdrojovým (open-source) kódem pro řešení kombinatorických optimalizačních problémů.

Sada obsahuje:

- řešitele programování s omezeními;
- dva řešitele lineárního programování;
- wrappery pro komerční a jiné open source řešitele, včetně řešitelů smíšených celočíselných úloh;
- algoritmy pro problém „balení košů“ a problém „batohu“ nebo mnoha batohů (knapsack);
- algoritmy pro problém obchodního cestujícího a problém směrování vozidel;
- grafové algoritmy (nejkratší cesty, tok s minimálními náklady, maximální tok, přiřazení lineárního součtu).

Nástroje OR-Tools tvůrci napsali v jazyce C++, ale poskytují také wrappery v jazycích Python, C# a Java.

Vývoj sady OR-Tools vede společnost Google LLC jako open-source projekt. Zdrojové kódy projektu jsou dostupné z veřejného repository google/or-tools dostupném na webové službě Github. Sada OR-Tools je licencována za podmínek Apache License 2.0.

Sada OR-Tools je poskytována mj. prostřednictvím Maven repository Maven Central. Proto postačilo v Netbeans projektu do souboru s konfigurací pro Maven přidat definici se závislostí (dependency) na OR-tools.

4.3.1 Dopravní problémy v reálném světě

Problém obchodního cestujícího v reálném nasazení znamená vyřešit problém obchodního cestujícího (TSP) pro místa zobrazená na mapě. K tomu je třeba matice vzdáleností z místa *i* do místa *j* pro všechna místa, která má cestující navštívit. Problém obchodního cestujícího je obvykle v praxi řešen jako podproblém VRP pro počet vozidel rovný 1 (u problému směrování vozidel (VRP) může být počet vozidel větší, než 1). Knihovny obvykle obsahují pouze metody pro VRP, protože se jedná o nejčastější použití.

Matice vzdáleností může být před spuštění TSP (resp. VRP) algoritmu explicitně definována, a to v případě předem vypočtených „vzdáleností“ (ohodnocení oblouků grafu). Takové řešení je vhodné, pokud plánování cest zohledňuje například pouze skutečné

vzdálenosti, které se nemění, nebo časy cesty, které však nezohledňují dopravní zatížení apod. Je také možné použít funkci pro aproximaci vzdáleností mezi místy: například euklidovský vzorec pro výpočet vzdálenosti mezi body v rovině. Stále je však efektivnější předem vypočítat všechny vzdálenosti mezi místy a uložit je do matice, než je počítat za běhu.

Další alternativou je použití vhodného mapového serveru. V současné době všechny mapové servery nabízejí rozhraní (API) pro volání svých metod z ostatních aplikací. Samozřejmě se liší rozsahem služeb a také cenovou politikou, podle které své služby nabízejí. Příkladem je Google Maps Distance Matrix API k dynamickému vytvoření matice vzdáleností (nebo cestovního času). Taková matice je pak vstupní maticí pro problém směrování.

Je třeba si připomenout, že v reálném nasazení je získané řešení problému směrování vozidel použito pro přikázání skutečné cesty řidiči vozidla (nebo cyklistovi, chodci atd.) Ten se pak v současné době standardně mezi zadanými místy, které má navštívit, pohybuje pomocí navigace v autě nebo v mobilním telefonu. Výstupem řešení problému směrování vozidel je pak seznam míst a určení pořadí, ve kterém je třeba je navštívit, ale skutečná trasa mezi místy je generována navigací. Krokem, který předchází směrování vozidel je ohodnocení hran (oblouků) grafu, jehož vrcholy odpovídají místům, která má navštívit právě dostupná flotila vozidel a ve kterém se hledají nejkratší, resp. nejméně nákladné cesty mezi těmito vrcholy. Tímto krokem vznikají vstupní data pro problémy směrování. Vstupem pro problém směrování je matice nákladů na cestu mezi všemi dvojicemi míst, které je třeba navštívit. Tyto náklady jsou vypočítávány ze vzdáleností nebo časů a případně i dalších vstupních dat (výškový profil, intenzita dopravy). Pro efektivní plánování cest, zvlášť pokud záleží i na časech návštěv přikázaných míst je nutné, aby tato vstupní data, tedy vzdálenosti nebo časy a další pro cesty mezi zadanými místy odpovídala co nejlépe vzdálenostem, časům atd. skutečné cesty.

Alternativou pro reálné ohodnocení cest je použití vhodného mapového serveru. V současné době všechny mapové servery nabízejí rozhraní (API) pro volání svých metod z ostatních aplikací. Samozřejmě se liší rozsahem služeb a také cenovou politikou, podle které své služby nabízejí. Příkladem je Google Maps Distance Matrix API k dynamickému vytvoření matice vzdáleností (nebo cestovního času). Obdobnou službu (obvykle se jménem obsahujícím slovo matrix) nabízejí dnes prakticky všechny mapové služby. Příkladem je Bing API založené na mapách Microsoft nebo služby založené na OSM mapách jako je

Graphhopper nebo Mapbox. Matice vzdáleností nebo časů je pak vstupní maticí pro problém směrování.

4.3.2 Vytvoření modelu směrování

Metoda poskytující řešení TSP (VRP) musí jako vstupní data mimo matice vzdáleností dostat také uzel odpovídající výchozímu bodu (depo) a informaci o tom, zda se obchodní cestující musí do depa vrátit. Obvykle lze metodě také poskytnout uživatelskou funkci pro výpočet vzdálenosti, respektive nákladů na hranu grafu nebo oblouk grafu. Tato funkce se standardně předává jako callback funkce. Výchozí možností je pak právě použití předem generované matice vzdáleností. To znamená, že náklady na cestu mezi libovolnými dvěma místy jsou právě vzdálenost mezi nimi. Obecně však mohou náklady zahrnovat i další faktory. Obvykle je vhodné definovat více metod pro vyhodnocování nákladů na oblouk. Volba metod primárně závisí na tom, které vozidlo cestuje mezi místy. Například pokud mají vozidla různou rychlost, lze definovat náklady na cestu mezi lokalitami jako vzdálenost vydělenou rychlostí vozidla – tedy jako dobu cesty. Samozřejmě, zde se situace komplikuje v případech, kdy aplikace používá pro přepravu mopedy, cyklisty nebo pěší, a zejména, pokud se jedná o přepravu po městě. Zde pak jak vzdálenost, tak i čas, může záviset na možnostech používat některé cesty, které jsou pro automobily jednosměrné, cesty jen pro cyklisty, cesty jen pro pěší, stoupání a klesání cesty, možnost přestupu na prostředek hromadné dopravy a další. Pak také může graf zahrnovat i hrany (oblouky) získané z mapy hromadné dopravy, jako například metro apod. V takových případech mohou být rychlosti pro jednotlivé typy přepravců velmi různé pro jednotlivé hrany (oblouky) grafu na jedné cestě v grafu.

4.3.3 Nastavení parametrů vyhledávání

Knihovny používají pro vyhledávání nejkratších cest v grafu některý z algoritmů, které neprohledávají všechny cesty grafu a k tomu vyžadují heuristickou metodu pro strategii hledání řešení. Řešitel směrování nevrací vždy optimální řešení TSP (VRP), protože problémy směrování jsou výpočetně příliš náročné nebo neřešitelné. Obvykle pro větší počet uzlů grafu, pro které hledáme řešení, vrácené řešení není optimální trasou. Často dochází k potížím při výskytu lokálního minima, neboli řešení, které je kratší, než všechny okolní trasy uvažované ve výpočtu díky použité (heuristické) strategii vyhledávání. Nalezené lokální minimum ale nemusí nutně být globálním minimem. Zde lze použít jinou strategii

hledání, která umožňuje řešiteli uniknout lokálnímu minimu. Po vzdálení se od lokálního minima řešitel pokračuje v hledání. Jedná se o obecný problém, obdobný jako například hledání minima v grafu.

4.3.4 Problém směrování vozidel

V problému směrování vozidel (VRP) je cílem najít optimální trasy pro více vozidel, která navštěvují množinu míst. (Pokud je k dispozici pouze jedno vozidlo, redukuje se na problém obchodního cestujícího).

Otázkou však je, co je „optimální trasa“ pro VRP? U problému obchodního cestujícího hledáme trasu s nejmenší celkovou vzdáleností. Takové zadání však pro VRP není vhodné. Pokud neexistují žádná další omezení a budeme-li hledat trasu s nejmenší celkovou vzdáleností, pak optimálním řešením je přiřadit právě jednomu vozidlu návštěvu všech míst a najít pro toto vozidlo nejkratší trasu. Jedná se v podstatě o stejný problém jako v případě TSP. To jistě není cílem VRP. Lepší způsob, jak definovat optimální trasy, je minimalizovat délku nejdelší jednotlivé trasy mezi všemi vozidly. To je správná definice, pokud je cílem dokončit všechny dodávky co nejdříve. Níže uvedený příklad VRP najde takto definované optimální trasy. V dalších částech pak budou zmíněny další způsoby zobecnění TSP přidáním omezení na vozidla:

- Kapacitní omezení: vozidla musí vyzvednout položky v každém místě, které navštíví, ale mají maximální přepravní kapacitu.
- Časová okna: každé místo musí být navštíveno v určitém časovém okně.

Následující kapitoly se budou zabývat ukázkami řešení různých modifikací VRP. Většina z nich (pokud není uvedeno jinak), využívá jako jeden ze vstupů zadání matici vzdáleností pro 16 bodů k navštívení. Ta je pro jednoduchost uvedena zde:

```
    public final long[][] distanceMatrix = {
     {0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388,
354, 468, 776, 662},
     {548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594,
480, 674, 1016, 868, 1210},
     {776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278,
1164, 1130, 788, 1552, 754},
     {696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514,
628, 822, 1164, 560, 1358},
     {582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400,
514, 708, 1050, 674, 1244},
     {274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662,
628, 514, 1050, 708},
     {502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004,
890, 856, 514, 1278, 480},
     {194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354,
320, 662, 742, 856},
     {308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696,
662, 320, 1084, 514},
     {194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422,
388, 274, 810, 468},
     {536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878,
764, 730, 388, 1152, 354},
     {502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0,
114, 308, 650, 274, 844},
     {388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114,
0, 194, 536, 388, 730},
     {354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308,
194, 0, 342, 422, 536},
     {468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650,
536, 342, 0, 764, 194},
     {776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152,
274, 388, 422, 764, 0, 798},
     {662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844,
730, 536, 194, 798, 0},
        };
```

4.3.5 Příklad VRP

V této části je uveden příklad VRP, v němž je cílem minimalizovat nejdelší jednotlivou trasu.

Vstupní data mají tuto strukturu:

- Matice vzdáleností mezi jednotlivými místy v délkových jednotkách (např. v metrech),
- Počet míst,
- Počet vozidel ve vozovém parku,
- Index depa – místa, kde všechna vozidla začínají a končí svou trasu.

Souřadnice umístění

Pro výpočet matice vzdáleností vycházíme ze souřadnic míst, která potřebujeme navštívit. Souřadnice polohy však nejsou součástí vstupních dat VRP problému. K řešení VRP potřebujeme pouze matici vzdáleností, kterou jsme předem vypočítali. Údaje o poloze jsou nutné pouze k identifikaci poloh jednotlivých uzlů, která jsou v matici vzdáleností označena svými indexy (0, 1, 2 ...). Po získání řešení VRP pak zpětně jednotlivým uzlům přiřadíme polohy pro navigaci vozidel.

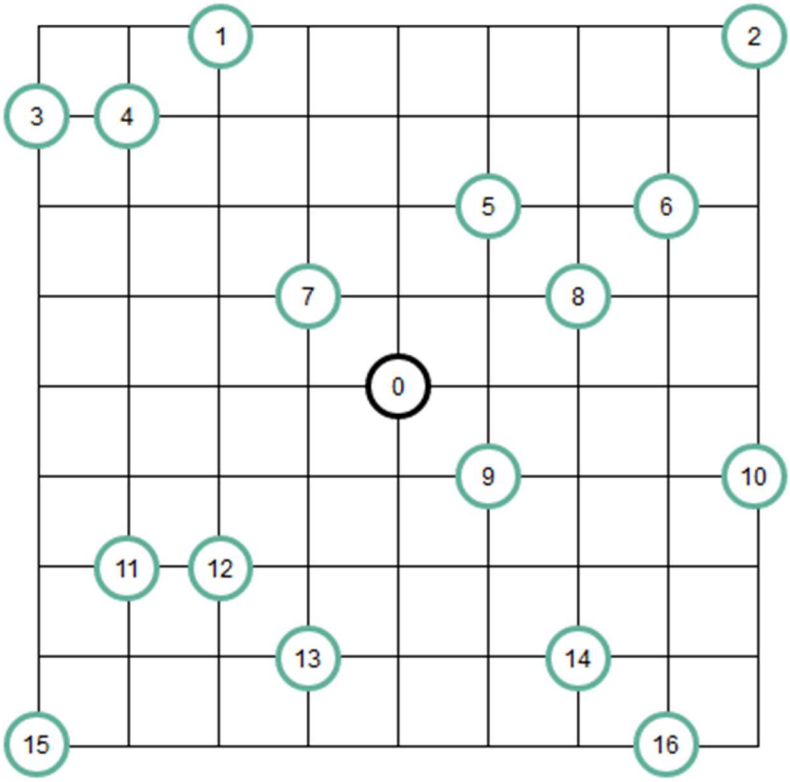
Vstupní data příkladu

Pro tento i následující příklady VRP byla zvolena následující vstupní data:

- Počet míst: 16
- Počet vozidel: 4
- Index depa: 0

Souřadnice poloh míst k navštívení a depa jsou uvedeny na následujícím diagramu (Obrázek 22):

Obrázek 22 - Souřadnice polohy pro problém VRP, černě je označeno depo



Pro zjednodušení sestavování problému se vzdálenosti mezi místy počítají pomocí Manhattanské vzdálenosti (Black, 2019), v níž je vzdálenost mezi dvěma body (x_1, y_1) a (x_2, y_2) definována jako $|x_1 - x_2| + |y_1 - y_2|$. Důvodem použití této definice vzdálenosti je jen její jednoduchost. Pro výpočet vzdáleností lze použít jakoukoli vhodnější metodu. Můžeme také získat matici vzdáleností pro libovolnou množinu míst na světě pomocí API rozhraní vhodné mapové služby. Mapové služby obvykle poskytují přímo službu, která vrací matici vzdálenosti pro celou sadu bodů najednou (například Google Distance Matrix API).

Přidání celkové vzdálenosti

Pro řešení VRP je třeba upravit uživatelskou funkci poskytnutou metodě knihovny tak, aby počítala kumulativní vzdálenost ujetou každým vozidlem na jeho trase (tedy kumulativní náklady na hrany nebo oblouky grafu). Obecně lze použít libovolné veličiny, včetně možnosti, kdy kumulativně sledujeme veličiny v průběhu výpočtu nákladů na celou cestu v grafu (dynamicky). Můžeme tak sledovat například únavu cyklisty nebo chodce, povinné přestávky v jízdě nebo přestávky na občerstvení apod.

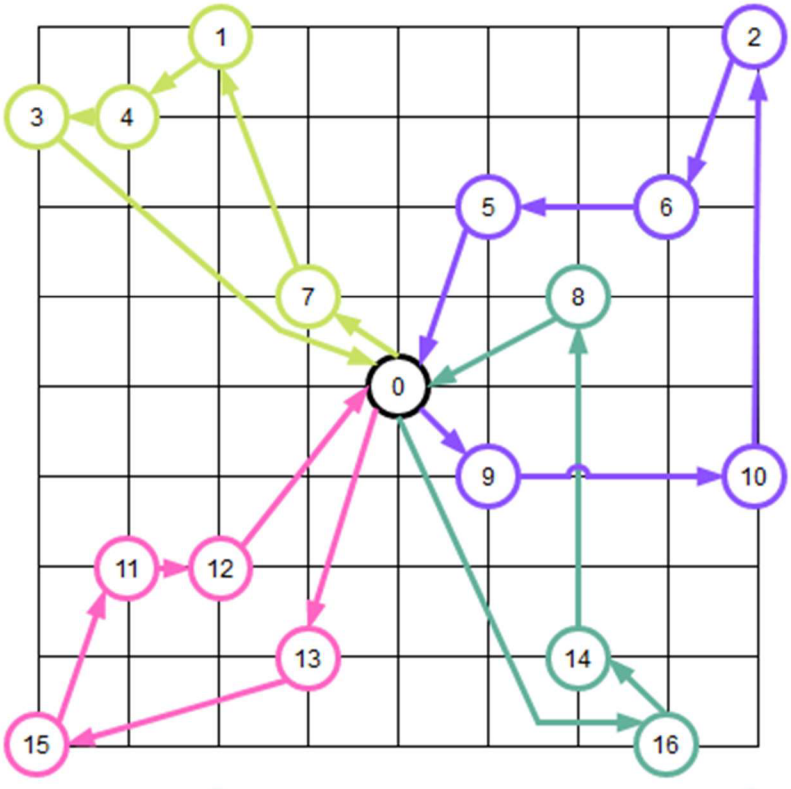
Řešení příkladu

Nyní je uveden výstup z IDE Apache NetBeans. První řádek (i pro ostatní příklady) obsahuje vždy hodnotu optimalizačního kritéria.

```
Optimalizovaný náklad řešení: 177500
Cesta pro vozidlo číslo: 0:
0 -> 9 -> 10 -> 2 -> 6 -> 5 -> 0
Délka cesty: 1712m
Cesta pro vozidlo číslo: 1:
0 -> 16 -> 14 -> 8 -> 0
Délka cesty: 1484m
Cesta pro vozidlo číslo: 2:
0 -> 7 -> 1 -> 4 -> 3 -> 0
Délka cesty: 1552m
Cesta pro vozidlo číslo: 3:
0 -> 13 -> 15 -> 11 -> 12 -> 0
Délka cesty: 1552m
Nejdelší cesta: 1712m
```

Grafické zobrazení cest pro jednotlivá vozidla (Obrázek 23):

Obrázek 23 - Kumulativní vzdálenost ujetá jednotlivými vozidly



Použití rozhraní mapového API pro výpočet matice vzdáleností

Služby mapového API lze použít k vytvoření matice vzdáleností pro libovolnou sadu míst definovaných adresami nebo zeměpisnými šířkami a délkami. Rozhraní API můžeme použít k výpočtu matice vzdáleností pro mnoho typů směrovacích problémů.

Služby API pro výpočet matice vzdáleností přijímají zeměpisné souřadnice nebo přímo seznam adres jednotlivých míst. Co použijeme záleží na účelu využití služby. Pokud po vyřešení problému VRP budeme zobrazovat pouze body na mapě a generovat cesty mezi body jen jako stopu na mapě, budeme předávat službě API pro výpočet matice vzdáleností souřadnice a po vyřešení VRP pak zpětně bodům řešení VRP přiřadíme souřadnice. Pokud řidič vozidla (chodec) bude používat své (klientské) mobilní zařízení pro navigaci, budeme předávat výstup jako data pro navigační software, který samostatně, nezávisle na serveru bude vypočítávat navigační pokyny, pak pravděpodobně budeme pro generování matice vzdáleností s výhodou předávat přímo seznam adres a pak zpětně bodům řešení VRP přiřadíme jednotlivé adresy v pořadí, ve kterém je má jednotlivé vozidlo (chodec) navštívit.

Odpověď API obsahuje vzdálenosti (v metrech) a dobu trvání cesty (v minutách a sekundách) mezi dvojicemi poloh nebo adres. Služby API jsou však z důvodu výpočetní náročnosti limitovány. Buď počtem zadaných poloh nebo počtem vracených párů start-cíl. Často také roste cena zavolání služby API s počtem dvojic. Například Google API Distance Matrix je omezeno na požadavky, pro které je vráceno maximálně 100 párů start-cíl.

Pokud bychom chtěli odeslat jeden požadavek obsahující více než 10 adres, a tedy požadovali více než 100 dvojic start-cíl, musíme volat více požadavků API.

4.3.6 Kapacitní omezení

Problém směrování vozidel s kapacitními omezeními (capacitated vehicle routing problem – CVRP) je VRP, ve kterém vozidla s omezenou přepravní kapacitou potřebují vyzvednout nebo doručit položky na různá místa. Položky mají určité množství, například hmotnost nebo objem, a vozidla mají maximální kapacitu, kterou mohou přepravit. Cílem je vyzvednout nebo doručit položky s co nejmenšími náklady, přičemž nikdy nesmí být překročena kapacita vozidel.

Funkce knihovny jsou volány tak, předpokládáme, že všechny položky jsou vyzvedávány nebo obdobně také tak, že jsou všechny položky doručovány. V tomto případě se kapacitní omezení uplatní, když vozidla opustí depo plně naložená. Kapacitní omezení je

však v obou případech implementováno stejným způsobem. Složitější situace nastává, pokud se na trase vozidel vyskytují jak vyzvednutí, tak dodávky.

Nové položky ve vstupních datech jsou:

- Poptávky: Každé místo má poptávku odpovídající množství – například hmotnosti nebo objemu – položky, která má být vyzvednuta.
- Kapacity: Každé vozidlo má kapacitu: maximální množství, které může vozidlo pojmout. Při jízdě vozidla po trase nesmí celkové množství přepravovaných položek nikdy překročit jeho kapacitu.

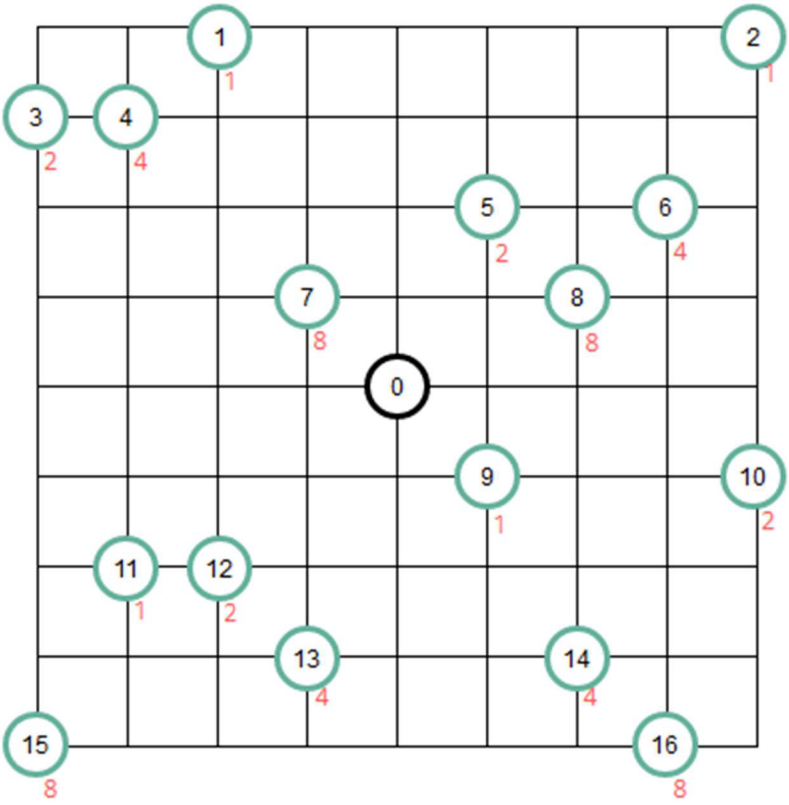
Pro výpočet vzdálenosti, respektive nákladů na hranu či oblouk grafu potřebujeme obdobnou funkci jako pro VRP bez kapacitních omezení.

Kromě toho potřebujeme uživatelskou funkci, která vrací poptávku v každém místě. Také tato funkce je obvykle předávána jako callback. Dále pak matici (vektor) kapacitních omezení pro jednotlivá vozidla.

Poptávky jednotlivých míst byly zadány následovně:

```
    public final long[] demands = {0, 1, 1, 2, 4, 2, 4, 8, 8, 1, 2,
1, 2, 4, 4, 8, 8};
```

Obrázek 24 - Souřadnice polohy včetně požadavků (červeně)



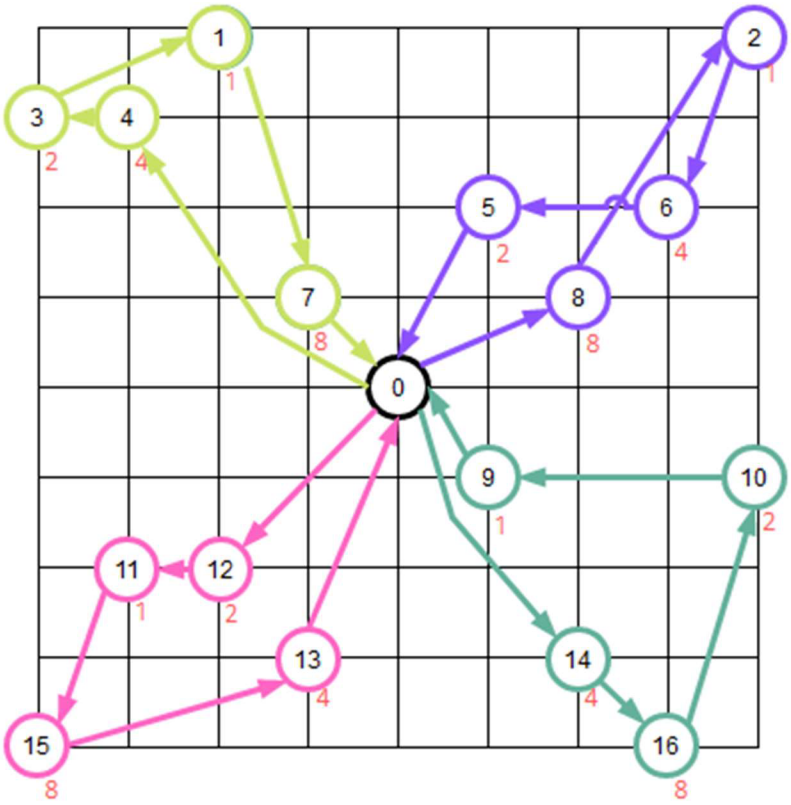
Předpokládejme, že naše 4 vozidla mají jednotlivou kapacitu 15 jednotek. Pak jejich trasy budou následující:

```
    Optimalizovaný náklad řešení: 6208
    Cesta pro vozidlo číslo: 0:
    0 Naloženo: (0) -> 4 Naloženo: (4) -> 3 Naloženo: (6) -> 1
Naloženo: (7) -> 7 Naloženo: (15) -> 0
    Délka cesty: 1552m
    Cesta pro vozidlo číslo: 1:
    0 Naloženo: (0) -> 14 Naloženo: (4) -> 16 Naloženo: (12) -> 10
Naloženo: (14) -> 9 Naloženo: (15) -> 0
    Délka cesty: 1552m
    Cesta pro vozidlo číslo: 2:
    0 Naloženo: (0) -> 12 Naloženo: (2) -> 11 Naloženo: (3) -> 15
Naloženo: (11) -> 13 Naloženo: (15) -> 0
    Délka cesty: 1552m
    Cesta pro vozidlo číslo: 3:
    0 Naloženo: (0) -> 8 Naloženo: (8) -> 2 Naloženo: (9) -> 6
Naloženo: (13) -> 5 Naloženo: (15) -> 0
    Délka cesty: 1552m
    Celková délka všech cest: 6208m
    Celkem naloženo za všechny cesty: 60
```

Výstup vypisuje index navštíveného místa a naložené množství jednotek v závorce.

Obrázek 25 zobrazuje grafické znázornění výstupu.

Obrázek 25 - Cesty pro jednotlivá vozidla při kapacitních omezeních



Problémy s více typy a kapacitami nákladu

Ve složitějších CVRP může každé vozidlo přepravovat několik různých typů nákladu, přičemž pro každý typ je stanovena maximální kapacita. Pro řešení takových problémů stačí naprogramovat příslušnou uživatelskou callback funkci, která pro každý typ nákladu vrací jinou velikost kapacity.

Co se stane, když problém nemá řešení?

Problém směrování s omezeními, jako je CVRP, nemusí mít proveditelné řešení – například pokud celkové množství přepravovaných položek přesahuje celkovou kapacitu vozidel. Můžeme samozřejmě jen nastavit limit času běhu programu, po jeho překročení prohlásit problém za neřešitelný a donutit uživatele přidat vozidla, navýšit jejich kapacitu a podobně. To však obvykle není operativní řešení a někdy je ekonomicky nepřijatelné. Častým řešením pak je nastavit sankce za upuštění od návštěvy míst. To umožňuje řešiteli VRP vrátit „řešení“, které nenavštíví všechna místa v případě, že je problém nesplnitelný.

Obecně může být těžké určit, zda daný problém má řešení. Dokonce i pro CVRP, ve kterém celková poptávka nepřevyšuje celkovou kapacitu, je určení, zda se všechny položky vejdou do vozidel, verzí vícenásobného problému batohu (Alves, et al.).

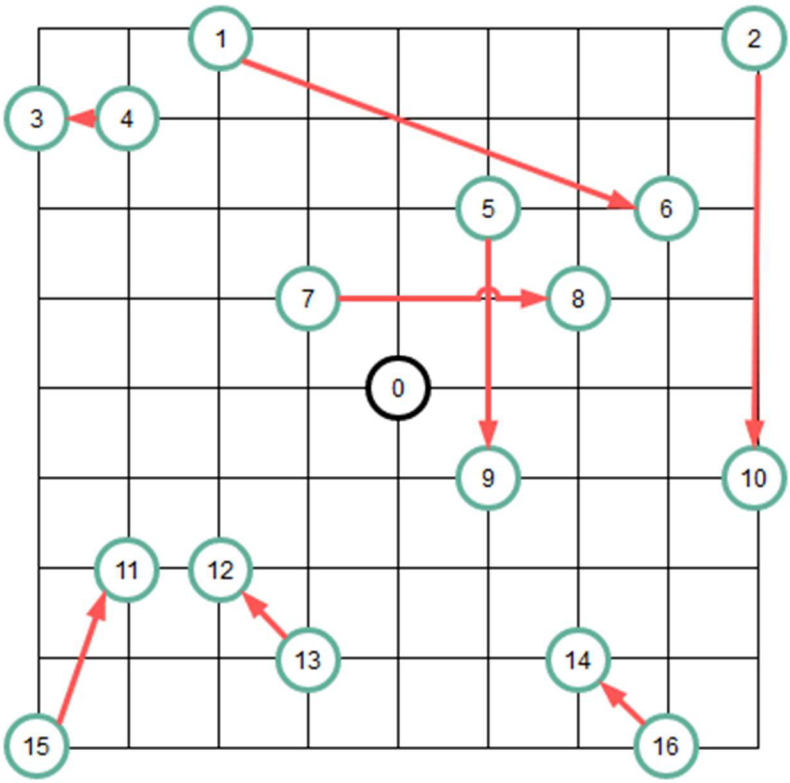
4.3.7 Směrování vozidel s vyzvednutím a doručením

V této části bude popsán VRP, ve kterém každé vozidlo vyzvedává položky na různých místech a předává je na jiných. Cílem je přiřadit vozidlům trasy pro vyzvednutí a doručení všech položek a zároveň minimalizovat délku nejdelší trasy.

Pro každou dvojici míst vyzvednutí-doručení v grafu existuje orientovaná hrana z místa vyzvednutí do místa doručení. Stejně jako v předchozích případech data zahrnují matici vzdáleností a seznam dvojic míst vyzvednutí a doručení, které odpovídají orientovaným hranám ve výše uvedeném diagramu. Knihovna musí podporovat heuristiku, která umožní přednostně dodržet nutné vyzvedávání a doručení. Používá se zobecněná heuristika přiřazování pro standardní problém směrování vozidel.

Jedná se o přístup „nejdříve shluk – potom trasa“, který poskytuje vynikající výsledky pro většinu standardních problémů. Tato heuristika nejprve přiřazuje skupinu zákazníků (tedy dvojic míst vyzvednutí-doručení) jednomu vozidlu a teprve poté řeší trasu vozidla (Fisher, et al., 1981).

Obrázek 26 - Dvojice míst vyzvednutí-doručení



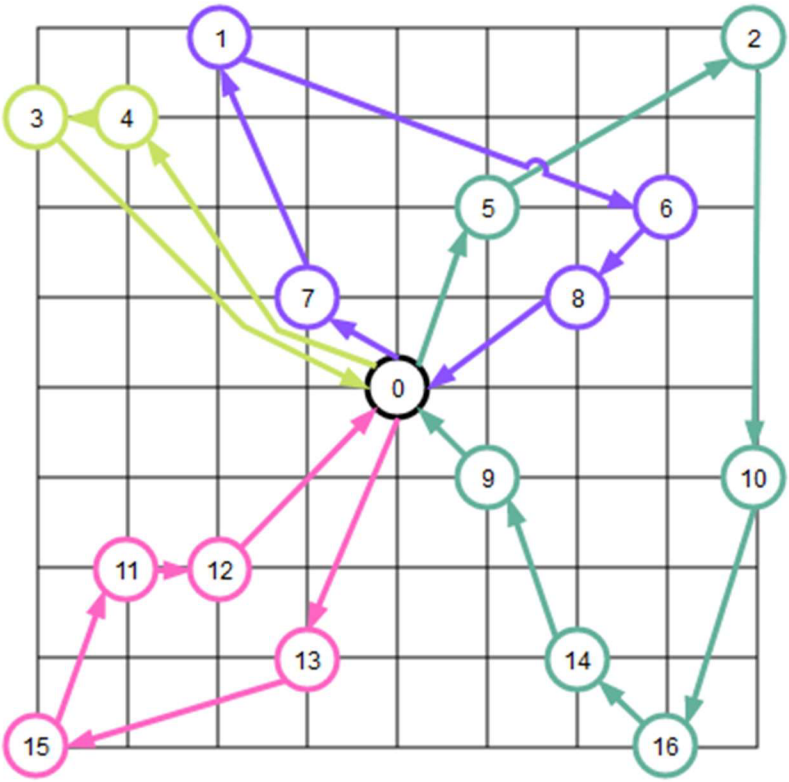
Tento příklad je opět obsluhován čtyřmi vozidly. Do vstupů příkladu byly přidány dvojice míst (znázorněné výše, Obrázek 26):

```
    public final int[][] pickupsDeliveries = {
            {1, 6},
            {2, 10},
            {4, 3},
            {5, 9},
            {7, 8},
            {15, 11},
            {13, 12},
            {16, 14},
        };
```

Výstup řešení problému je následující:

```
Optimalizovaný náklad řešení: 226116
Cesta pro vozidlo číslo: 0:
0 -> 13 -> 15 -> 11 -> 12 -> 0
Délka cesty: 1552m
Cesta pro vozidlo číslo: 1:
0 -> 5 -> 2 -> 10 -> 16 -> 14 -> 9 -> 0
Délka cesty: 2192m
Cesta pro vozidlo číslo: 2:
0 -> 4 -> 3 -> 0
Délka cesty: 1392m
Cesta pro vozidlo číslo: 3:
0 -> 7 -> 1 -> 6 -> 8 -> 0
Délka cesty: 1780m
Celková délka všech cest: 6916m
```

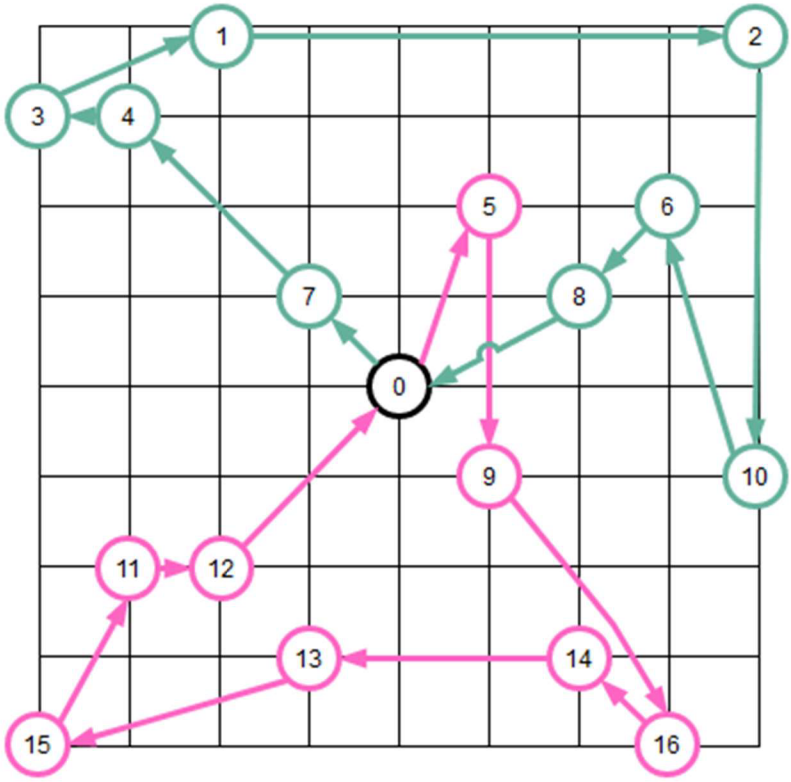
Obrázek 27 - Trasy vozidel pro dvojice míst vyzvednutí-doručení



Pro tento případ byl spočten ještě jeden příklad, kde došlo ke snížení počtu vozidel ze čtyř na dvě. Následuje výstup řešení problému a jeho grafické znázornění.

```
Optimalizovaný náklad řešení: 300060
Cesta pro vozidlo číslo: 0:
0 -> 5 -> 9 -> 16 -> 14 -> 13 -> 15 -> 11 -> 12 -> 0
Délka cesty: 2716m
Cesta pro vozidlo číslo: 1:
0 -> 7 -> 4 -> 3 -> 1 -> 2 -> 10 -> 6 -> 8 -> 0
Délka cesty: 2944m
Celková délka všech cest: 5660m
```

Obrázek 28 - Trasy v případě pouze dvou obslužných vozidel



První výpočet pro 4 vozidla nalezl 4 trasy s počty zastávek 5, 7, 3, 5. Druhý výpočet nalezl pro 2 vozidla 2 trasy s počty zastávek 9, 9. První výpočet trval 0,9 s, druhý výpočet trval 48,9 s (na stroji s procesorem Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz, 32GB RAM, výpočty používají standardně jedno vlákno). Tento výsledek potvrzuje extrémně stoupající výpočetní náročnost úlohy pro zvyšující se počet uzlů na trase jednoho vozidla.

4.3.8 Problém směrování vozidel s časovými okny

Mnoho problémů trasování vozidel zahrnuje plánování návštěv zákazníků, kteří jsou k dispozici pouze v určitých časových oknech. Tyto problémy jsou známé jako problémy směrování vozidel s časovými okny (VRPTW).

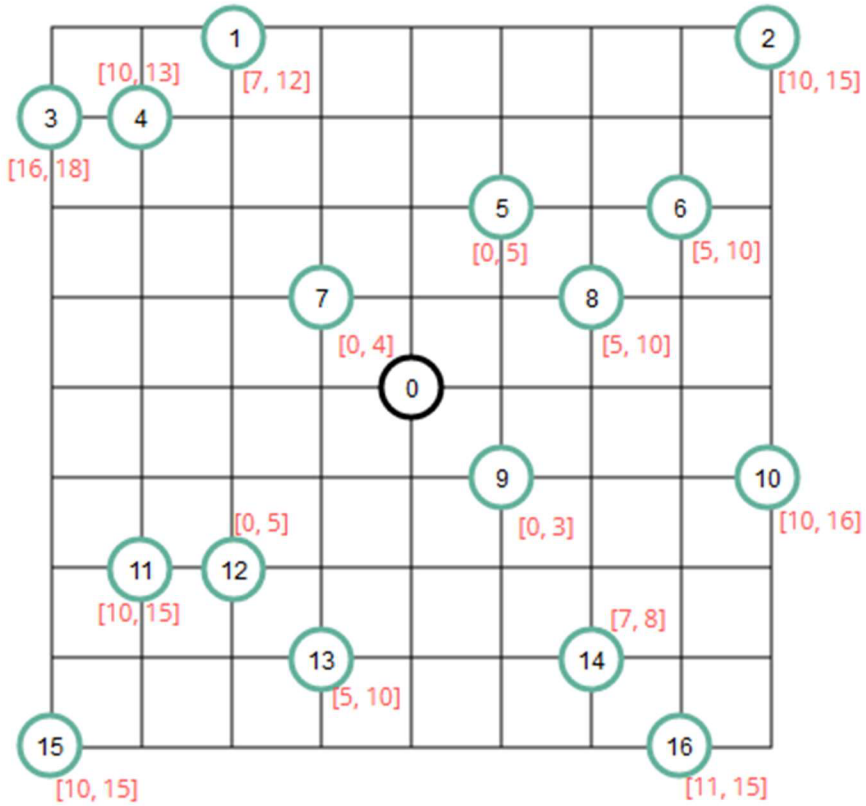
Protože problém zahrnuje časová okna, vstupními daty je matice času, která musí obsahovat cestovní časy mezi místy (nikoli matice vzdáleností jako v předchozích příkladech). Cílem je minimalizovat celkovou dobu jízdy vozidel.

Vstupní data se skládají z:

- Matice časů cest mezi jednotlivými místy. Zde je rozdíl proti předchozím problémům, které používají matici vzdáleností. Pokud všechna vozidla jedou stejnou rychlostí, získáme stejné řešení s použitím matice vzdáleností i matice časů, protože cestovní vzdálenosti jsou konstantním násobkem cestovních časů.
- Časová okna. Pole časových oken pro místa, která si lze představit jako požadované časy pro návštěvu. Vozidla musí navštívit lokalitu v rámci jejího časového okna.
- Počet vozidel ve vozovém parku.
- Index depa.

Pro volání funkce knihovny musíme vytvořit uživatelskou funkci pro výpočet nákladů na oblouk, která musí při definování nákladů na cestu vycházet z časů cest mezi lokalitami. Tato funkce se obvykle opět předává jako callback funkce knihovny pro VRP. Tato uživatelská callback funkce sleduje náklady, tedy časy, které se kumulují během trasy vozidla. Pokud callback funkce zohledňuje pouze čas, vrací kumulativní dobu jízdy vždy, když vozidlo přijede do místa s daným indexem. Součástí vstupních dat jsou časová okna, obvykle ve formě horní meze pro čekací doby v místech. Obvykle není potřebná minimální čekací doba.

Obrázek 29 - Souřadnice polohy včetně časových oken (červeně)



Na rozdíl od předchozích příkladů je zde vstupní matice časová, a ne vzdáleností. Je zde tedy využitá matice časů zadaná následovně:

```
public final long[][] timeMatrix = {
 {0, 6, 9, 8, 7, 3, 6, 2, 3, 2, 6, 6, 4, 4, 5, 9, 7},
 {6, 0, 8, 3, 2, 6, 8, 4, 8, 8, 13, 7, 5, 8, 12, 10, 14},
 {9, 8, 0, 11, 10, 6, 3, 9, 5, 8, 4, 15, 14, 13, 9, 18, 9},
 {8, 3, 11, 0, 1, 7, 10, 6, 10, 10, 14, 6, 7, 9, 14, 6, 16},
 {7, 2, 10, 1, 0, 6, 9, 4, 8, 9, 13, 4, 6, 8, 12, 8, 14},
 {3, 6, 6, 7, 6, 0, 2, 3, 2, 2, 7, 9, 7, 7, 6, 12, 8},
 {6, 8, 3, 10, 9, 2, 0, 6, 2, 5, 4, 12, 10, 10, 6, 15, 5},
 {2, 4, 9, 6, 4, 3, 6, 0, 4, 4, 8, 5, 4, 3, 7, 8, 10},
 {3, 8, 5, 10, 8, 2, 2, 4, 0, 3, 4, 9, 8, 7, 3, 13, 6},
 {2, 8, 8, 10, 9, 2, 5, 4, 3, 0, 4, 6, 5, 4, 3, 9, 5},
 {6, 13, 4, 14, 13, 7, 4, 8, 4, 4, 0, 10, 9, 8, 4, 13, 4},
 {6, 7, 15, 6, 4, 9, 12, 5, 9, 6, 10, 0, 1, 3, 7, 3, 10},
 {4, 5, 14, 7, 6, 7, 10, 4, 8, 5, 9, 1, 0, 2, 6, 4, 8},
 {4, 8, 13, 9, 8, 7, 10, 3, 7, 4, 8, 3, 2, 0, 4, 5, 6},
 {5, 12, 9, 14, 12, 6, 6, 7, 3, 3, 4, 7, 6, 4, 0, 9, 2},
 {9, 10, 18, 6, 8, 12, 15, 8, 13, 9, 13, 3, 4, 5, 9, 0, 9},
 {7, 14, 9, 16, 14, 8, 5, 10, 6, 5, 4, 10, 8, 6, 2, 9, 0},
    };
```

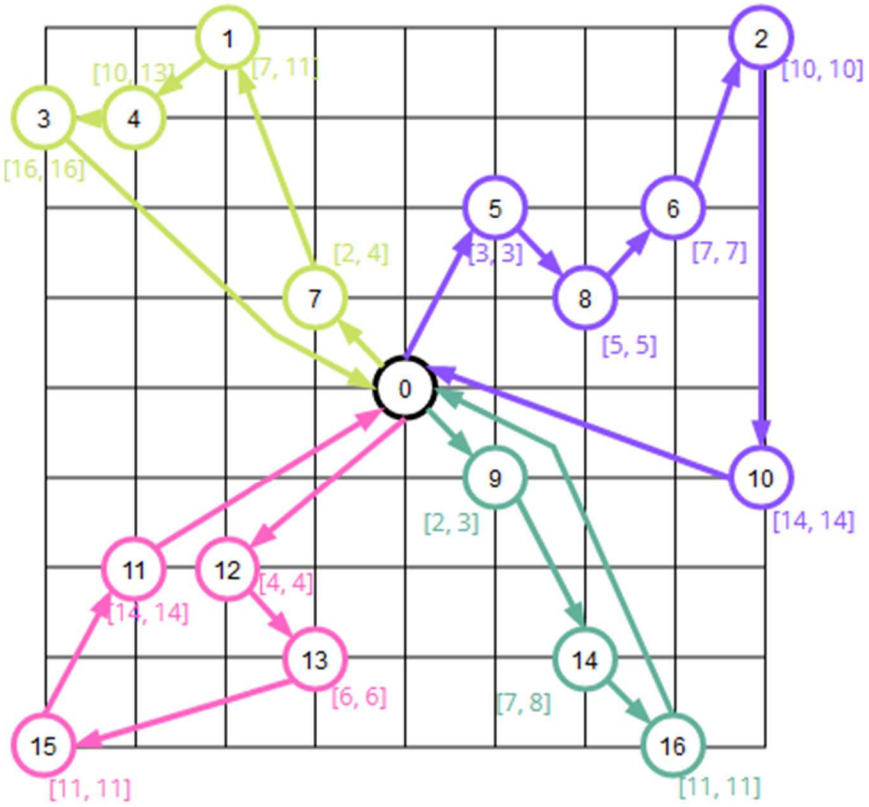
Časová okna byla do zadání přidána následovně:

```
public final long[][] timeWindows = {
        {0, 5}, // depot
        {7, 12}, // 1
        {10, 15}, // 2
        {16, 18}, // 3
        {10, 13}, // 4
        {0, 5}, // 5
        {5, 10}, // 6
        {0, 4}, // 7
        {5, 10}, // 8
        {0, 3}, // 9
        {10, 16}, // 10
        {10, 15}, // 11
        {0, 5}, // 12
        {5, 10}, // 13
        {7, 8}, // 14
        {10, 15}, // 15
        {11, 15}, // 16
    };
```

Výstup řešení je uveden níže, obsahuje index navštíveného místa a v závorkách časy příjezdu a odjezdu.

```
    Optimalizovaný náklad řešení: 71
    Cesta pro vozidlo číslo: 0:
    0 Čas: (0,0) -> 9 Čas: (2,3) -> 14 Čas: (7,8) -> 16 Čas: (11,11)
-> 0 Čas: (18,18)
    Čas cesty: 18min
    Cesta pro vozidlo číslo: 1:
    0 Čas: (0,0) -> 7 Čas: (2,4) -> 1 Čas: (7,11) -> 4 Čas: (10,13)
-> 3 Čas: (16,16) -> 0 Čas: (24,24)
    Čas cesty: 24min
    Cesta pro vozidlo číslo: 2:
    0 Čas: (0,0) -> 12 Čas: (4,4) -> 13 Čas: (6,6) -> 15 Čas: (11,11)
-> 11 Čas: (14,14) -> 0 Čas: (20,20)
    Čas cesty: 20min
    Cesta pro vozidlo číslo: 3:
    0 Čas: (0,0) -> 5 Čas: (3,3) -> 8 Čas: (5,5) -> 6 Čas: (7,7) -
> 2 Čas: (10,10) -> 10 Čas: (14,14) -> 0 Čas: (20,20)
    Čas cesty: 20min
    Celkový čas všech cest: 82min
```

Obrázek 30 - Trasy vozidel pro problém s časovými okny



Okna řešení

Řešení VRPTW obsahuje mimo posloupností míst pro jednotlivá vozidla také časové okno v každém místě, které má vozidlo navštívit a které je nutno dodržet – okno řešení. Okno řešení v místě je časový interval, během kterého musí vozidlo přijet, aby dodrželo plán. Okno řešení je podintervalem časového okna zadaného jako omezení v místě a obvykle je menší než toto okno.

4.3.9 Omezení zdrojů

Doposud jsme se zabývali problémy směrování s omezeními, která platí během jízdy vozidla. Dále představíme VRPTW, který má omezení také v depu: všechna vozidla musí být naložena před odjezdem z depa a vyložena při návratu. Protože jsou k dispozici například pouze dvě nakládací rampy, mohou být současně naložena nebo vyložena nejvýše dvě vozidla. V důsledku toho musí některá vozidla čekat na naložení jiných, což zdržuje jejich odjezd z depa. Problémem je najít optimální trasy vozidel pro VRPTW, které zároveň splňují omezení nakládky a vykládky v depu.

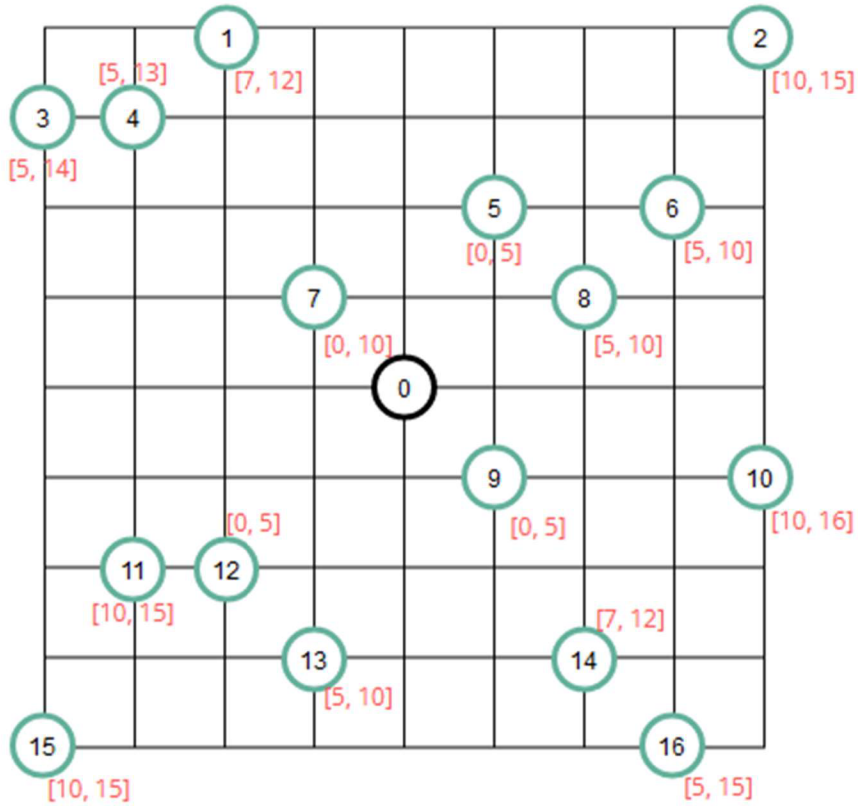
Vstupní data v tomto případě obsahují:

- Pole časů cest mezi jednotlivými místy.
- Pole časových oken pro požadované návštěvy míst.
- Čas potřebný k naložení vozidla.
- Čas potřebný k vyložení vozidla.
- Maximální počet vozidel, která mohou být naložena nebo vyložena současně.

Zde musíme vytvořit uživatelskou funkci, která generuje časová okna pro nakládku a vykládku vozidel v depu. Tato okna jsou proměnná časová okna, což znamená, že nemají pevný čas začátku a konce (na rozdíl od časových oken v lokacích). Šířky oken jsou určeny hodnotami časy potřebnými pro naložení vozidel.

Pro tento příklad bude využitá také matice časů a časová okna, stejně jako v předchozím případě.

Obrázek 31 - Souřadnice polohy včetně omezení zdrojů (červeně)



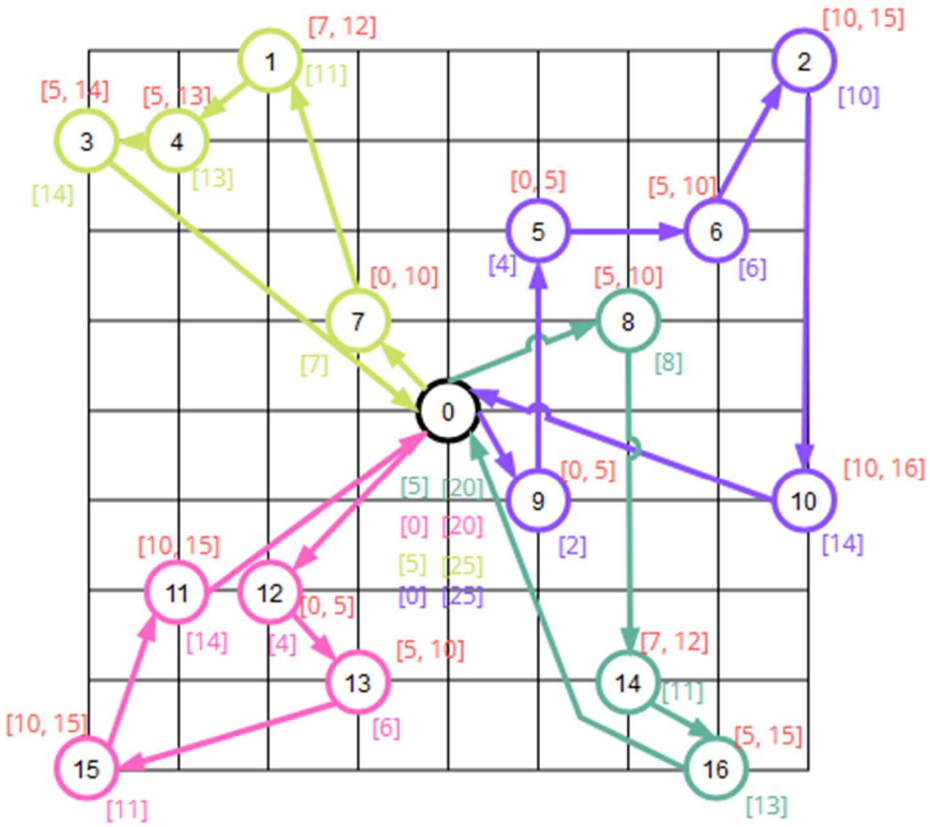
Dále jsou do zadání vloženy časy nakládky a vykládky a počet nakládacích ramp v depu:

```
    public final int vehicleLoadTime = 5;
        public final int vehicleUnloadTime = 5;
        public final int depotCapacity = 2;
```

Výsledek pak zákonitě musí obsahovat situace, kdy některá vozidla musí čekat na naložení ostatních. V tomto příkladě je to zjevné u vozidel 1 a 3 (ve výstupu označené jako vozidla 0 a 2), které čekají 5 časových jednotek, než se uvolní obě nakládací rampy, kde nakládka/vykládka trvá právě 5 časových jednotek.

```
    Optimalizovaný náklad řešení: 71
    Cesta pro vozidlo číslo: 0:
    0 Čas: (5,5) -> 8 Čas: (8,9) -> 14 Čas: (11,12) -> 16 Čas:
(13,15) -> 0 Čas: (25,25)
    Čas cesty: 25min
    Cesta pro vozidlo číslo: 1:
    0 Čas: (0,0) -> 12 Čas: (4,4) -> 13 Čas: (6,6) -> 15 Čas: (11,11)
-> 11 Čas: (14,14) -> 0 Čas: (20,20)
    Čas cesty: 20min
    Cesta pro vozidlo číslo: 2:
    0 Čas: (5,5) -> 7 Čas: (7,7) -> 1 Čas: (11,11) -> 4 Čas: (13,13)
-> 3 Čas: (14,14) -> 0 Čas: (24,24)
    Čas cesty: 24min
    Cesta pro vozidlo číslo: 3:
    0 Čas: (0,0) -> 9 Čas: (2,3) -> 5 Čas: (4,5) -> 6 Čas: (6,9) -
> 2 Čas: (10,12) -> 10 Čas: (14,16) -> 0 Čas: (29,29)
    Čas cesty: 29min
    Celkový čas všech cest: 98min
```

Obrázek 32 - Trasy vozidel s omezením zdrojů a časovými okny naložení-vyložení



4.3.10 Sankce a vypuštění návštěv

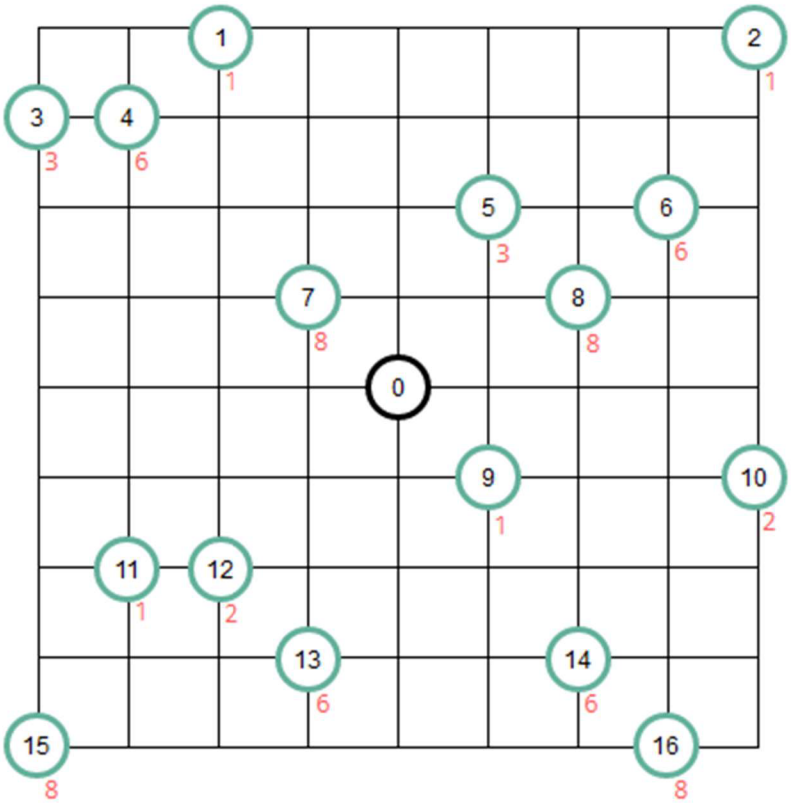
V důsledku předepsaných omezení může dojít k tomu, že problém směrování nemá reálné řešení. Pokud je například zadán VRP s kapacitními omezeními, v němž celková poptávka na všech místech přesahuje celkovou kapacitu vozidel, řešení neexistuje. Jedinou možností by bylo navýšit počet nebo kapacitu vozidel. To obvykle není realizovatelné operativně a obvykle ani z ekonomických důvodů. V takovém případě musí vozidla upustit od návštěvy některých míst. Problém spočívá v tom, jak rozhodnout, které návštěvy upustit.

K vyřešení tohoto problému zavedeme sankce za vynechání jednotlivých míst. Použít lze stejnou řešitelskou funkci knihovny, ale kdykoli se od návštěvy místa upustí, k celkovým kumulativním nákladům vozidla se přičte sankce. Řešitel pak najde trasu, která minimalizuje celkové náklady (například celkovou vzdálenost) plus součet pokut za všechna vyřazená místa.

Jako příklad uvažujme jednoduchý VRP s kapacitními omezeními, který je dán níže uvedeným kódem a diagramem, v němž červená čísla (kromě depa) představují požadavky. Tento příklad opět v zadání využívá výše uvedenou matici vzdáleností.

```
    public final long[] demands = {0, 1, 1, 3, 6, 3, 6, 8, 8, 1, 2,
1, 2, 6, 6, 8, 8};
```

Obrázek 33 - Příklad VRP s kapacitními omezeními (červeně)



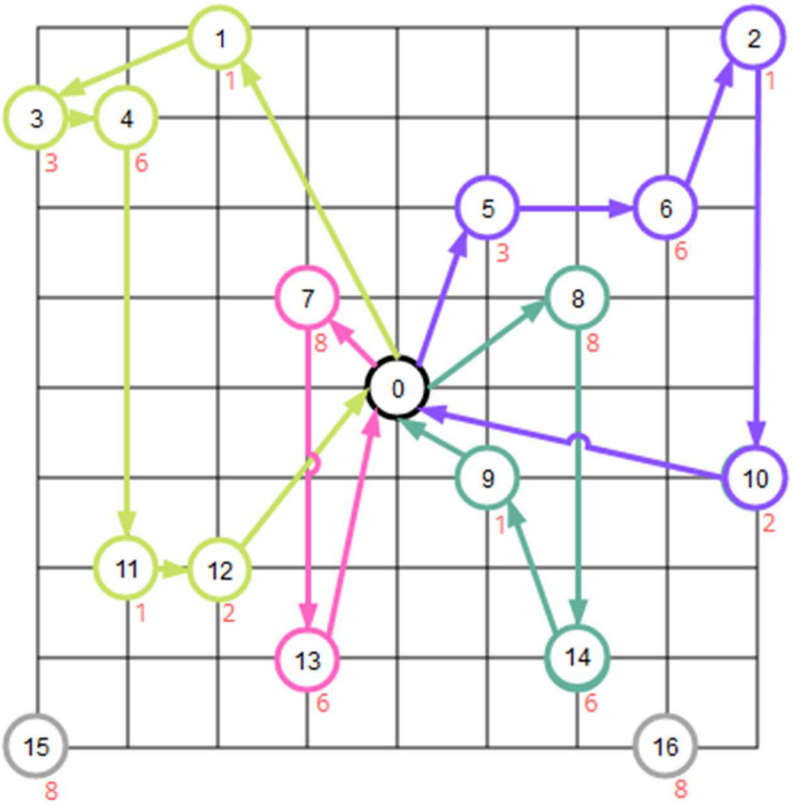
Předpokládejme, že existují čtyři vozidla, každé s kapacitou 15. Nelze naložit náklad na všech šestnácti místech, protože součet všech požadavků je 70 a celková kapacita všech vozidel je 60 (limit 4 × 15 = 60). Problém se řeší tak, že každému místu přiřadíme velkou pokutu – řekněme 100. Poté, co řešitel zjistí, že problém je nesplnitelný, vypustí místa 15 a 16. Nejkratší trasy jsou zjevné na následujícím diagramu (Obrázek 34) a ve výstupu řešení:

```
    Optimalizovaný náklad řešení: 7548
    Vynechaná místa: 15, 16
    Cesta pro vozidlo číslo: 0:
    0 Naloženo: (0) -> 8 Naloženo: (8) -> 14 Naloženo: (14) -> 9
Naloženo: (15) -> 0
    Délka cesty: 1096m
    Cesta pro vozidlo číslo: 1:
    0 Naloženo: (0) -> 1 Naloženo: (1) -> 3 Naloženo: (4) -> 4
Naloženo: (10) -> 11 Naloženo: (11) -> 12 Naloženo: (13) -> 0
    Délka cesty: 1872m
    Cesta pro vozidlo číslo: 2:
    0 Naloženo: (0) -> 7 Naloženo: (8) -> 13 Naloženo: (14) -> 0
    Délka cesty: 868m
    Cesta pro vozidlo číslo: 3:
    0 Naloženo: (0) -> 5 Naloženo: (3) -> 6 Naloženo: (9) -> 2
Naloženo: (10) -> 10 Naloženo: (12) -> 0
```

```
Délka cesty: 1712m
Celková délka všech cest: 5548m
Celkem naloženo za všechny cesty: 54
```

Výstup řešení opět uvádí index místa a v závorce celkem naložené jednotky.

Obrázek 34 - Trasy vozidel s vypuštěním návštěv míst 15 a 16



Velikost penalizace

Výše penalizace v poměru ke kumulaci nákladů na cestu vozidla (například celková vzdálenost) závisí na aplikaci. Například pro případ, kde chceme uskutečnit co nejvíce dodávek, volí uživatel velkou sankci v poměru k nákladům cesty. Menší penalizace vede k vypuštění více míst, než je nezbytné k tomu, aby byl problém řešitelný. To je také možnost, kterou lze volit v případě například v případě, kdy návštěva některých míst je spojena s dodatečnými náklady.

4.3.11 Další modifikace problému směrování vozidel

Nastavení počátečních tras pro vyhledávání

U některých problémů je také možné zadat sadu počátečních tras pro VRP, místo vyhledání počátečního řešení podle nějaké heuristiky. Dá se říci, že heuristikou v tomto případě je vyjít z již vyzkoušeného dobrého řešení.

Nastavení počátečních a koncových míst tras

Dosud jsme předpokládali, že všechna vozidla začínají a končí v jediném místě, a to v depu. Pro každé vozidlo v problému VRP můžeme také chtít nastavit různá počáteční a koncová místa.

Povolení libovolného místa začátku a konce

V některých verzích problému směrování vozidel je povoleno, aby vozidla začínala a končila v libovolných místech. Obvykle toho lze dosáhnout, stačí vždy uvažovat náklady na cestu z libovolného místa do depa a opačně za nulové.

5 Výsledky a diskuse

V rámci této diplomové práce byly prozkoumány některé z principů vývoje aplikací a logistiky. Byla zaměřena na sektor B2C e-commerce a rychlé doručování zboží.

V první části byly představeny principy vývoje aplikace pro Android OS. Velikou výhodou právě vybraného operačního systému je jeho široké využití a přenositelnost. Je také distribuován jako Open Source, takže k jeho vývoji a vývoji aplikací pro něj má snadnou přístupnost široká veřejnost.

V této části také byly představeny principy designu systému a aplikací. Byly jmenovány principy vývoje pro různě velké displeje, tedy škálovatelný design a byla zdůrazněna důležitost úvahy nad rozmístěním ovládacích prvků, když bereme v potaz ovládání zařízení jednou rukou. Také byl zmíněn Material Design, což je set doporučení a pravidel, na která by měl dbát každý vývojář aplikací pro Android OS, aby docílil jejího hladkého fungování, a hlavně snadného používání uživateli, kteří jsou již obeznámeni s dalšími aplikacemi navržené stejným způsobem.

V další části této diplomové práce byly probrány mapové služby a některé algoritmy pro vyhledání nejkratší cesty v grafu (Dijkstrův, A* a Floyd-Warshallův algoritmus). Po vyhledání nejkratších cest mezi jednotlivými body pak následuje jejich propojení v celkovou nejkratší cestu pomocí dalších uvedených algoritmů (Problém obchodního cestujícího, Problém směrování vozidel a Job shop scheduling). Právě problém obchodního cestujícího je v praktické části práce řešen detailněji.

V praktické části se pak práce zabývá řešením business logiky aplikace, jejíž prototyp byl vytvořen v bakalářské práci autorky. Je zde popsán způsob fungování aplikace pro zápůjčku a prodej společenských šatů, jejich rozvoz a případné využití čistíren jako meziskladů. Princip fungování aplikace je také detailněji rozveden pomocí objektově orientovaného modelování a UML diagramů.

V hlavní části praktické sekce pak byl vytvořena sada testovacích metod pro ověření logistických principů v jazyce Java. Zabývá se především řešení problému směrování vozidel (VRP), což je zobecněný případ problému obchodního cestujícího. Tento problém je řešen v několik variantách s různými omezeními. Omezení kapacitní, zdrojů a časových oken, problém s vyzvednutím a doručením a problém s vypuštěním návštěv.

6 Závěr

V dnešní době jsou problémy logistiky čím dál tím více běžnou součástí denního života. Více a více firem má jako svůj hlavní předmět činnosti založen na doručování a přepravě zásilek přímo koncovým zákazníkům do bytu či na pracoviště. Souvisí to s velkým rozvojem e-comerce pro retail zákazníky. Systémy pro řízení firemních činností stále častěji obsahují moduly pro logistiku, zejména problematiku obsluhy první a poslední míle. Nezbytnou součástí takovýchto systémů je zapojení mobilních zařízení, jako jsou tablety, mobilní telefony nebo automobilové palubní systémy, které interagují se serverovými řešeními i navzájem.

Hlavním cílem této diplomové práce bylo ověřit možnosti vývoje klient-server mobilních aplikací B2C v jazyce Java se zaměřením na možnosti integrace existujících řešení pro architekturu klient – server určenou pro mobilní klientská zařízení a na serverové straně využívající integraci s výpočetními nástroji pro logistiku, mapovými aplikacemi a službami pro routování a navigaci

Dalším cílem bylo ověřit praktickou použitelnost současných výpočetních nástrojů a modelů pro logistiku pro účely dopravních problémů v prostředí pozemních sítí silnic a cest.

V této práci je ověřena existence celé řady v praxi bezprostředně využitelných softwarových řešení a online služeb, které lze použít k vybudování softwarového systému pro řízení doručování koncovým zákazníkům. Konkrétně bylo ověřeno, že mobilní platforma Android je snadno použitelná pro využívání online navigačních a mapových služeb a pro komunikaci systému klient – server. Na serverové straně lze pak využít ověřená serverová řešení pro výstavbu systému, který integruje aplikaci pro e-commerce, řízení nákupu a skladových zásob a pro řešení logistických a transportních problémů souvisejících s doručováním a obsluhou koncových zákazníků. Pro konkrétní řešení serverové aplikace byla ověřena možnost použití jazyka Java, na jehož základě lze vystavět aplikaci, která využívá výpočetních knihoven pro logistiku a na druhou stranu umožňuje nasazení v aplikačním serveru pro komunikaci s mobilními klienty.

7 Seznam použitých zdrojů

Alves, Gabriel, Jain, Pranjal a Kile, Jeff. Backpack Problem. *Brilliant.* [Online] https://brilliant.org/wiki/backpack-problem/.

Android Developers. 2023. Android for Developers. [Online] 2023. https://developer.android.com/.

Android Developers. 2023. Android Studio Electric Eel | 2022.1.1. *Android Studio.* [Online] 2023. https://developer.android.com/studio/releases.

Android Developers. Platform Architecture. *Android Developers.* [Online] https://developer.android.com/guide/platform.

Android Inc. Legacy HALs. *Android Source.* [Online] [Citace:] https://source.android.com/docs/core/architecture/hal/archive.

Android Studio. 2023. 2023.

Black, Paul E. 2019. Manhattan distance. *Dictionary of Algorithms and Data Structures.* 2019.

Cormen, Thomas H., a další. 2009. *Introduction to Algorithms.* 3. Massachusetts : Massachusetts Institute of Technology, 2009. ISBN 978-0-262-53305-8.

Dantzig, G. B. a Ramser, J. H. 1959. The Truck Dispatching Problem. *Management Science.* místo neznámé : INFORMS, 1959. Sv. 6, 1, stránky 80-91.

Design do kapsy. Gestalt principy. *Design do kapsy.* [Online] https://designdokapsy.cz/zaklady/principy/gestalt-principy/.

Fisher, M. L. a Jaikumar, R. 1981. A Generalized Assignment Heuristic for Vehicle Routing. *Networks.* 1981. 11, stránky 109-124.

Fleck, Renee. 2020. 8 mobile UX best practices every designer should consider. *Dribble.* [Online] 2020. https://dribbble.com/stories/2020/05/12/8-mobile-design-tips.

Gaffuri, Julien. 2012. Toward Web Mapping with Vector Data. [editor] N. Xiao, a další. *Proceedings of the The Annual International Conference on Geographic Information Science.* 2012, Sv. 7478, stránky 87-101.

Gargenta, Marko. 2011. *Learning Android.* Sebastopol : O'Reilly Media, Inc., 2011. ISBN 1449307248.

GeeksforGeeks. 2023. A* Search Algorithm. *GeeksforGeeks.* [Online] 2023. https://www.geeksforgeeks.org/a-search-algorithm/.

Goodchild, Michael F. 1990. Tiling large geographical databases. [editor] A.P. Buchmann, a další. 1990, Sv. 409, stránky 135-146.

Google Developers. 2023. The activity lifecycle. *Google Developers.* [Online] 2023. https://developer.android.com/guide/components/activities/activity-lifecycle.

Google Developers. 2023. The Job Shop Problem. *Google OR-Tools.* [Online] 2023. https://developers.google.com/optimization/scheduling/job_shop.

Google Developers. 2023. Vehicle Routing. *Google OR-Tools.* [Online] 2023. https://developers.google.com/optimization/routing.

Google Firebase. 2023. Make your app the best it can be. *Google Firebase.* [Online] 2023. https://firebase.google.com/.

Gourley, David, a další. 2002. *HTTP: The Definitive Guide.* místo neznámé : O'Reilly Media, Inc., 2002. ISBN 1565925092.

Gustas, Remigijus. 2011. Modeling Approach for Integration and Evolution of Information System Conceptualization. *International Journal of Information System Modeling and Design.* 2011. Sv. 2, 1, stránky 45-73.

Hart, Peter E., Nilsson, Nils J. a Raphael, Bertram. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics.* 1968. Sv. 4, 2, stránky 100-107.

Interaction Design Foundation. Mobile User Experience (UX) Design. *Interaction Design Foundation.* [Online] https://www.interaction-design.org/literature/topics/mobile-UX-design.

Interaction Design Foundation. Visual Hierarchy. *Interaction Design Foundation.* [Online] https://www.interaction-design.org/literature/topics/visual-hierarchy.

Jena, Ajay Kumar, Swain, Santosh Kumar a Mohapatra, Durga Prasad. 2014. A novel approach for test case generation from UML activity diagram. *International Conference on Issues and Challenges in Intelligent Computing Techniques.* 2014. stránky 621-629.

Lacko, Ľuboslav. 2017. *Mistrovství - Android.* Brno : Albatros Media a. s., 2017. ISBN 978-80-251-4875-4.

Lumen Learning. Shortest Path. *Lumen Learning.* [Online] https://courses.lumenlearning.com/waymakermath4libarts/chapter/shortest-path/.

MapTiler. Tiles à la Google Maps. *MapTiler.* [Online] https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/.

Netek, Rostislav, a další. 2020. Performance Testing on Vector vs. Raster Map Tiles—Comparative Study on Load Metrics. *ISPRS International Journal of Geo-Information.* 2020, Sv. 9, 2.

Noskov, Alexey. 2018. Computer Vision Approaches for Big Geo-Spatial Data: Quality Assessment of Raster Tiled Web Maps for Smart City Solutions. [editor] Bandrova T. a Konečný M. 2018, stránky 296-305.

Oluwatosin, Haroon Shakirat. 2014. Client-Server Model. *IOSR Journal of Computer Engineering.* 2014. Sv. 16, 1, stránky 67-71. ISSN 2278-8727.

opensource.com. What is open source? *opensource.com.* [Online] https://opensource.com/resources/what-open-source.

Panjuta, Denis. 2022. The Complete Android 12 & Kotlin Development Masterclass. *Udemy.* Červenec 2022.

Pecinovský, Rudolf a Pavlíčková, Jarmila. 2021. *Začínáme programovat v jazyku Java.* Praha : Grada Publishing, a.s., 2021. ISBN 978-80-271-3062-7.

Peterson, Michael P. 2012. The Tile-Based Mapping Transition in Cartography. [autor knihy] L., Reyes Nunez, J. Zentai. *Maps for the Future. Lecture Notes in Geoinformation and Cartography.* Berlin : Springer, Berlin, Heidelberg, 2012, Sv. 5, stránky 151-163.

StatCounter. 2016. *Mobile and tablet internet usage exceeds desktop for first time worldwide.* 2016.

StatCounter. 2022. Mobile Operating System Market Share Worldwide. *StatCounter.* [Online] Duben 2022. https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202204-202204-bar.

STechies. Dijkstra's algorithm in Python. *STechies.* [Online] https://www.stechies.com/dijkstras-algorithm-python/.

Šubrt, Tomáš, Fejtar, Jiří a Mach, Jiří. 2022. Dopravní logistika II - Obslužnost úseků dopravní sítě. 2022.

Svobodová, Kateřina. 2020. Návrh a implementace modelu a prototypu mobilní aplikace. Praha : Česká zemědělská univerzita, 2020.

Szlenk, Marcin. 2006. Formal Semantics and Reasoning about UML Class Diagram. *International Conference on Dependability of Computer Systems.* 2006. stránky 51-59.

Taylor, Petroc. 2023. Mobile Android operating system market share by version worldwide from January 2018 to January 2023. *Statista.* [Online] 27.. únor 2023. https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/.

Zavadil, Filip. 2013. Software pro zobrazování a export mapových dlaždic. Praha : České vysoké učení technické, 2013.

8 Seznam obrázků

9 Přílohy

Příloha A – Obsah přiloženého CD

Příloha A

Obsah přiloženého CD

Kořenový soubor aplikace pro IDE Apache NetBeans řešící uvedené příklady dopravních problémů:

Příloha\dp-master\src\main\java\svok\dp\Dp.java