Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Indoor navigační systém

Bakalářská práce

Michal Bartoš

Vedoucí práce: Mgr. Jakub Geyer

České Budějovice 2019

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: Michal Bartoš
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika

Katedra/ústav PřF JU: Ústav aplikované informatiky

Školitel: Mgr. Jakub Geyer
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF JU:
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant:
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: Indoor navigační systém

Cíle práce:

Cílem práce je vytvořit navigační aplikaci pro mobilní zařízení, která bude umožňovat vyhledání optimální trasy a navigaci ve vnitřní prostorách budov. Aplikace bude umožňovat parametrizaci vyhledání trasy (např. trasa pro vozíčkáře). Součást práce není automatické určení polohy ve vnitřních prostorách budov. Práce bude obsahovat mapové navigační podklady a grafické znázornění minimálně 1 kompletní budovy fakulty. Součástí práce budou bezpodmínečně následující dokumenty:

- *Analýza funkčních a nefunkčních požadavků*
- *Analýza a výběr vhodných softwarových prostředků*
- *Dokumentace kódu*
- *Testovací dokumentace*
- *Uživatelská dokumentace*

Základní doporučená literatura:

- - - - -

Financování práce .....

Školitel práce ..... podpis: .....

U externích vedoucích fakultní garant práce ..... podpis: .....

Garant oboru bak. studia (nepožaduje se u oboru biologie) ..... podpis: .....

Vedoucí katedry/ústavu PřF JU, kde proběhne obhajoba ..... podpis: .....

Případný souhlas vedoucího ústavu AV ..... podpis: .....

V Českých Budějovicích dne

Podpis studenta: [redacted] .....

Bibliografické údaje

Bartoš M., 2019: Indoor navigační systém. [Indoor navigation system. Bc. Thesis, in Czech.] – 36 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tématem této bakalářské práce je vytvoření mobilní aplikace, která uživatelům umožní vyhledání optimální trasy ve vnitřních prostorách budov Jihočeské univerzity s ohledem na parametry trasy. Teoretická část se zabývá analýzou požadavků, metodikou a na základě získaných informací vybírá ideální technologie pro vyvíjenou aplikaci. Praktická část popisuje implementaci zvolených technologií a následné testování této aplikace. Aplikace implementuje uživatelské scénáře popisované v teoretické části.

Abstract

The topic of this bachelor thesis is to create a mobile application that allows optimal route search inside the buildings of the University of South Bohemia with respect to route parameters. The theoretical part deals with the analysis of requirements, methodology and on the basis of obtained information it selects ideal technologies for the application being developed. The practical part of the thesis describes implementation of selected technologies and subsequent testing of this application. The application implements user scenarios described in the theoretical part of the thesis.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 10.12.2019 Michal Bartoš

Poděkování

Rád bych poděkoval svému školiteli, Mgr. Jakubu Geyerovi, za odborné vedení, cenné rady a ochotu. Dále bych rád poděkoval své rodině a přítelkyni za jejich trpělivost a podporu.

Obsah

1. Úvod

Bakalářská práce se zabývá tvorbou mobilní aplikace pro navigaci ve vnitřních prostorách budov Jihočeské univerzity. Cílovou skupinou této aplikace jsou studenti, zejména studenti prvních ročníků, nebo veřejnost při různých veřejných událostech. Toto téma bylo zvoleno z důvodu obtížnější orientace v areálu Jihočeské univerzity. Z vlastní zkušenosti vím, že univerzitní navigaci bych nejen já, ale mí kolegové mnohdy ocenili. Dalším důvodem je dlouhodobý požadavek na navigační aplikaci ze strany univerzity.

1.1 Cíle práce

Cílem práce je vytvořit intuitivní mobilní aplikaci, která bude umožňovat vyhledání místností v areálu Jihočeské univerzity, zadání a vykreslení optimální trasy uvnitř budovy. Výběr trasy bude umožňovat parametrické vyhledávání, např. trasa pro vozíčkáře nebo bez použití výtahu. Dále bude možné vyhledávat informace o zaměstnancích Jihočeské univerzity. Výsledkem bude demo aplikace zahrnující budovu C Přírodovědecké fakulty Jihočeské univerzity s možností budoucího rozšíření o další budovy.

Z důvodů náročnosti problematiky univerzitní navigace bylo přistoupeno k rozdělení projektu na 3 části. Finální aplikace se bude skládat ze tří částí, a to:

1. Indoor lokalizace zpracovaná Davidem Langerem.
2. Indoor navigace zpracovaná v rámci této bakalářské práce.
3. Outdoor navigace zpracovaná Pavlem Beranem.

Jelikož všechny projekty budou používat některé části kódu stejné (např. vykreslení trasy, skript pro aktualizaci dat ze serveru), budou vyvíjeny společně s kolegy. Také zdrojová data budou identická a budou získávána z portálů STAG[1] a OrgStructure[2] v rámci bakalářské práce kolegy Pavla Berana, jenž má tuto část obsaženou ve své práci.

[1]Portál *STAG* je dostupný na: https://wstag.jcu.cz/portal/
[2]Portál *OrgStructure* je dostupný na: https://orgstr.jcu.cz/

2. Analýza

Funkční a nefunkční požadavky byly získány na základě požadavků Ústavu aplikované informatiky. Pro získání funkčních požadavků bylo provedeno mimo jiné i doplňkové dotazníkové šetření na deseti respondentech. Cílem bylo získat předběžný přehled funkcí, které uživatelé od aplikace očekávají.

2.1 Funkční požadavky

1. Vyhledání optimální trasy ve vnitřních prostorách budov.
2. Vykreslení optimální trasy.
3. Fulltextové vyhledávání.
4. Zobrazování mapových podkladů.
5. Zobrazení informací o jednotlivých místnostech.
6. Zobrazení informací o jednotlivých zaměstnancích.
7. Uložení posledních pěti vyhledávaných výrazů.
8. Parametrické vyhledávání trasy:
    a) Uživatel bude moci zvolit bezbariérovou trasu.
    b) Uživatel bude moci zvolit použití výtahu.
9. Získání výchozí pozice uživatele:
    a) Pomocí fulltextového vyhledávače.
    b) Pomocí rozšířeného vyhledávání.
    c) Pomocí stisknutí bodu na mapě.

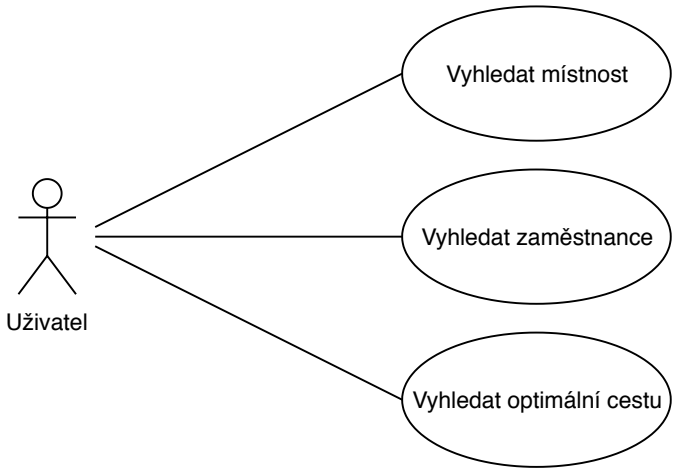
2.2 Nefunkční požadavky

1. Mobilní aplikace.
2. Data uchovávána v grafově orientované databázi.
3. Použití MVVM architektury.

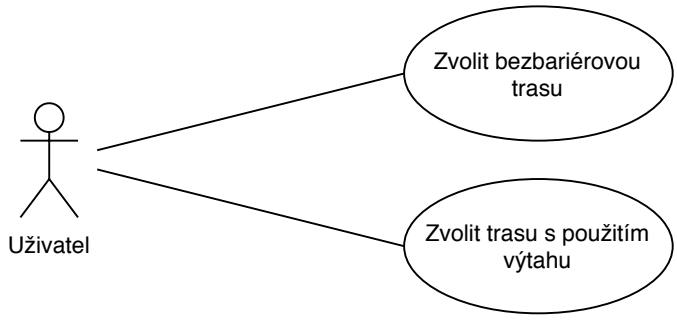
2.3 Případy užití

Na základě výše popsaných skutečností a konzultací s vedoucím práce byly sestaveny případy užití a následně pro přehlednost rozděleny do čtyř částí:

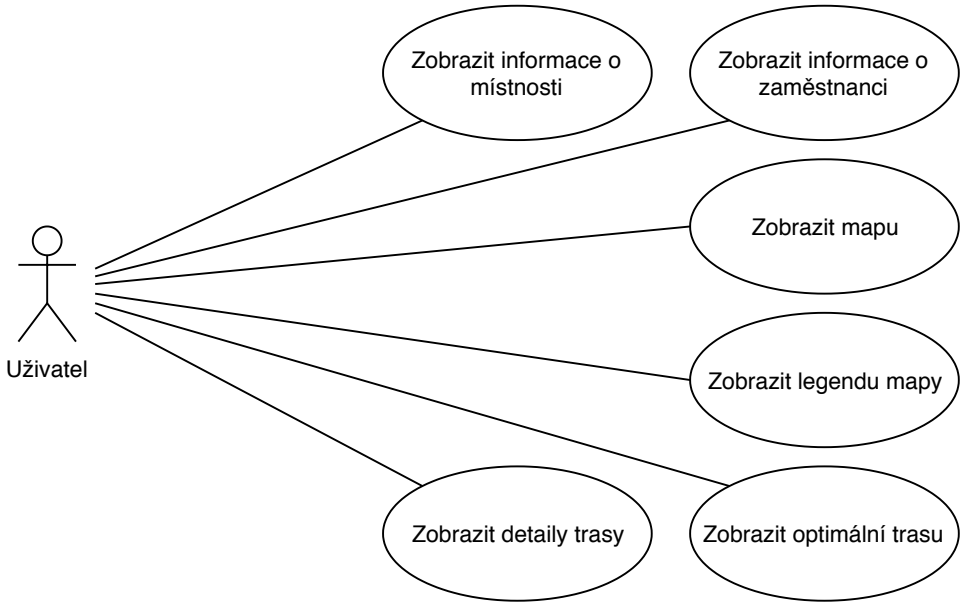
- Vyhledání objektů, viz Obrázek 2.1.
- Parametrizace trasy, viz Obrázek 2.2.
- Zobrazení objektů, viz Obrázek 2.3.
- Určení výchozí pozice, viz Obrázek 2.4.



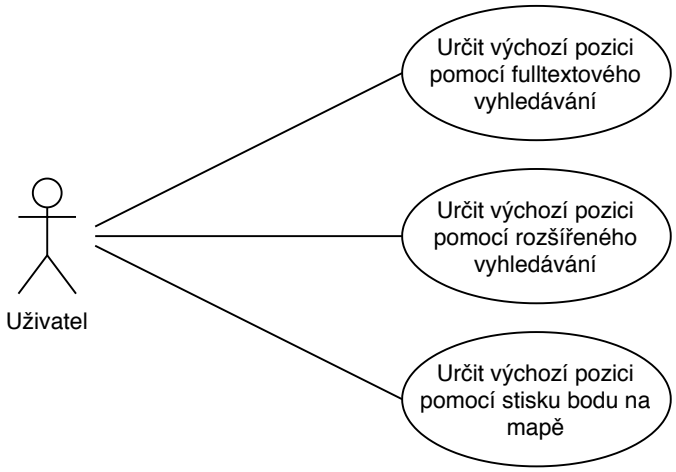
Obrázek 2.1: Diagram případů užití – část Vyhledání objektů



Obrázek 2.2: Diagram případů užití – část Parametrizace trasy



Obrázek 2.3: Diagram případů užití – část Zobrazení objektů



Obrázek 2.4: Diagram případů užití – část Určení výchozí pozice

2.4 Metodika vývoje

Vývoj mobilní aplikace je dynamický, na aplikaci jsou často kladeny požadavky na změny ze strany klienta, je tedy vhodné použít agilní metodiku vývoje, jedním z těchto přístupů je Scrum.

Scrum je framework pro vývoj a udržování komplexních produktů [1]. Rolemi ve Scrumu jsou product owner, scrum master a vývojový tým. Pro organizaci práce používá Scrum tzv. backlogy, základně se dělí na produktový a sprint backlog.

Produktový backlog je seznam úkolů, ve kterých se kumuluje budoucí práce,

sprint backlog je seznam úkolů, ve kterém se kumulují úkoly rozpracované v aktuálním sprintu. Sprint je čas, ve kterém se má realizovat daný sprint backlog, obvykle jeden nebo dva týdny. Nedílnou součástí scrumu jsou tzv. ceremonies. Daily scrum probíhá denně a slouží k synchronizaci aktivit a k vytvoření plánu na dalších 24 hodin [2]. Sprint planning slouží k určení priorit a naplánování úkolů do dalšího sprintu, probíhá vždy před začátkem nadcházejícího sprintu. Retrospektiva je událost, ve které mají členové týmu možnost zhodnotit předchozí sprint nebo co by se mělo v příštím sprintu zlepšit.

Pro tuto práci byl zvolen Scrum ve zjednodušené formě, jelikož autor této práce byl zároveň člen vývojového týmu i scrum master. Implementace jednotlivých funkcí byla rozdělena na části, složitější úkoly byly rozděleny do menších částí, kterým byla přidělena priorita dokončení, popis implementace a datum plánovaného dokončení sprintu. Pro znázornění tzv. scrum boardu byla zvolena aplikace Trello[1], kde byly vytvořeny následující sloupce:

- *Backlog*, obsahující všechny nevyřešené úkoly,
- *Sprint backlog*, obsahující úkoly k dokončení v aktuálním sprintu,
- *Probíhající*, obsahující momentálně vyvíjené funkce,
- *Testování*, obsahující funkce k otestování,
- *Dokončené*, obshaující všechny dokončené úkoly.

Sloupce *Backlog* a *Sprint backlog* byly seřazeny podle priority od nejvyšší po nejnižší a do sloupce *Probíhající* byly vždy vybírány úkoly dle nejvyšší priority. Jednotlivým položkám byl také přiřazen štítek označující, zda se jedná o vylepšení aplikace či chybu v aplikaci. V průběhu vývoje aplikace byl kladen důraz na to, aby nebyly současně vyvíjeny více než dvě funkce, to má za cíl omezit množství rozdělané práce. Sloupec *Testování* sloužil k důslednému otestování implementovaných funkcí. Dokončené úkoly byly následně verzovány pomocí nástroje Git do soukromého repozitáře [3]. Vývoj aplikace byl průběžně konzultován s vedoucím práce, ve Scrumu má tato pozice nejblíže definici product ownera.

[1]Dostupné na: https://trello.com/

3. Návrh

Tato kapitola se zabývá popisem a výběrem vhodných technologií, popisem drátěných modelů a architekturou mobilní aplikace.

3.1 Technologie

3.1.1 Platforma

Před začátkem vývoje aplikace je velmi důležité vybrat správné technologie, toto rozhodnutí může výrazně ovlivnit dobu vývoje. Jedním z nefunkčních požadavků byla mobilní aplikace, na dnešním trhu s mobilními telefony dominují především operační systémy Android a iOS. Dle dat[1] z prosince 2018 měl systém Android podíl 76,65 % na českém trhu, systém iOS pouze 21,23 %. Zbylých 2,12 % tvořily ostatní operační systémy.

K vývoji mobilní aplikace lze přistupovat třemi hlavními způsoby:

1) Nativní mobilní aplikace

2) Multiplatformní mobilní aplikace

3) Webová aplikace

Nativní aplikace

Nativní mobilní aplikace jsou vyvíjeny pro konkrétní platformu a v programovacím jazyce definovaným danou platformou. [5] Nejčastěji se jedná o systém Android společnosti Google, umožňující vývoj pomocí programovacích jazyků Java nebo Kotlin a o systém iOS společnosti Apple, umožňující vývoj v jazycích Swift nebo Objective-C.

Výhodou nativních aplikací je především rychlost a velmi dobrý uživatelský prožitek. Nativní aplikace poskytují vývojářům přímý přístup k celé sadě vlastností zařízení, například k GPS nebo fotoaparátu, a to vede k rychlejšímu spuštění dané vlastnosti.

Hlavní nevýhodou je především omezení aplikace pouze na jednu platformu. V případě, že vznikne požadavek na podporu více platforem a aby aplikace zůstala nativní, je nutné vyvíjet každou aplikaci zvlášť v podporovaném programovacím jazyce dané platformy. V tomto případě roste doba i cena vývoje, jelikož je nutné znát konkrétní programovací jazyky.

[1] http://gs.statcounter.com/os-market-share/mobile/czech-republic/2018

Multiplatformní aplikace

Multiplatformní mobilní aplikace je taková aplikace, kterou je možné spustit na více mobilních platformách. Největší výhodou těchto aplikací je sdílení určitého množství kódu napříč platformami, v závislosti na výběru frameworku a složitosti aplikace lze sdílet až 100 % kódu [6]. Frameworky pro vývoj multiplatformních mobilních aplikací lze rozdělit na dvě skupiny, a to na hybridní webové aplikace a hybridní nativní aplikace.

Hybridní nativní aplikace jsou mnohdy téměř nerozeznatelné od nativních aplikací co se grafického prostředí týče, jelikož používají stejné UI prvky. Tyto aplikace jsou po zkompilování 100% nativní, využívají prostředků konkrétní platformy a stejně jako klasické nativní aplikace mají přístup k API[2] zařízení. Tyto aplikace jsou při kompilaci překládány do jazyka dané platformy. Typickými zástupci této skupiny jsou Xamarin Forms a React Native.

Hybridní webové aplikace, někdy též označované jako HTML5 aplikace, jsou aplikace vyvíjené pomocí technologií HTML5, CSS a Javascript. Tyto aplikace využívají mobilní platformu Webview, což je zjednodušený webový prohlížeč běžící v rámci aplikace. Webview běží v režimu celého okna a na rozdíl od klasického webového prohlížeče umožňuje přístup k API zařízení. Hybridní webové aplikace lze spouštět na všech mobilních zařízeních s podporou WebView. Typickými zástupci této skupiny jsou Apache Cordova a Ionic.

Webová aplikace

Webové aplikace se načítají v prohlížeči mobilního zařízení, na rozdíl od nativních nebo multiplatformních aplikací není nutná jejich instalace. Jedná se o webové stránky optimalizované pro mobilní zařízení. Webové aplikace také nezabírají žádné úložiště v mobilním zařízení. Vývoj těchto aplikací je poměrně rychlý, nicméně i to má svá úskalí a nevýhody. Hlavní nevýhodou je velmi omezené API zařízení.

Shrnutí

Je velmi obtížné určit, který přístup je pro vývoj nejlepší, vždy záleží na konkrétních požadavcích a očekáváních dané aplikace. V případě této práce byl zvolen přístup hybridních nativních aplikací, konkrétně framework Xamarin Forms se zaměřením na operační systémy Android a iOS. Vývoj této práce probíhal současně s vývojem Indoor lokalizace, v průběhu vývoje tedy nebylo známé, která API zařízení budou pro Indoor lokalizaci potřebná, z tohoto důvodu byl zvolen framework Xamarin Forms oproti hybridním webovým aplikacím. Tento framework byl také zvolen z důvodu možnosti multiplatformního vývoje. Minimální podporovanou verzí systému iOS byla zvolena verze iOS 9.0. Minimální podporovanou verzí systému Android byla zvolena verze Android 5.0 Lollipop.

[2] API – Rozhraní pro programování aplikací

Jedná se o dnes již velmi zastaralou verzi, nicméně oproti verzi 4.4 Kitkat zde proběhly výrazné systémové změny. Předchozí verze, Android 4.4 Kitkat měla pouze 6,53% podíl[3] na českém trhu v prosinci 2018.

3.1.2 Databáze

Dle nefunkčních požadavků aplikace byla pro ukládání dat zvolena grafově orientovaná databáze. Grafově orientovaná databáze se skládá z vrcholů a hran. Jednotlivé vrcholy (anglicky *nodes*) reprezentují rozcestí a místnosti, hrany (anglicky *relationships*) reprezentují propojení mezi vrcholy, jejich parametrem je vzdálenost mezi jednotlivými vrcholy. Tyto databáze již mnohdy nabízejí implementované vyhledávání nejkratší cesty, například pomocí Dijkstrova algoritmu.

Neo4j

Neo4j je čistě grafově orientovaná databáze nabízející edici Community, která je poskytnuta pod licencí GPL v3, tudíž je bezplatná i pro komerční použití. Tato databáze nabízí i možnost použít embedded[4] databázi, nicméně chybí podpora mobilních zařízení. Databáze nabízí možnost rozšíření o pluginy a procedury, oficiální zdroje nabízejí ke stažení mimo jiné i Dijkstrův algoritmus.

OrientDB

OrientDB není čistě grafově orientovaná databáze, jedná se o multi-modelový[5] systém mimo jiné poskytující grafově orientovaný databázový systém [8]. Tato databáze je poskytnuta pod licencí Apache2, je tedy bezplatná i pro komerční použití. OrientDB také nabízí možnost použití embedded databáze, ovšem nenabízí podporu mobilních zařízení.

VelocityDB

VelocityDB je grafově orientovaná databáze, která nabízí mimo jiné i embedded verzi databáze pro operační systém Android a iOS za použití platformy .NET, tedy v případě mobilních aplikací frameworku Xamarin. V době vývoje aplikace nicméně nepodporovala lokální úložiště dat z důvodu chyby v implementaci, tato chyba byla opravena v aktualizaci 8.1.4[6]. Nevýhodou této databáze je téměř neexistující dokumentace a velmi malá komunita vývojářů.

[3] http://gs.statcounter.com/android-version-market-share/mobile-tablet/czech-republic/2018

[4] Embedded databáze je databáze běžící v rámci aplikace.

[5] Multi-modelový databázový systém poskytuje více datových modelů v rámci jednoho systému.

[6] http://wordpress.velocitydb.com/?p=86

Sparksee

Společnost Sparksity Technologies nabízí serverovou i embedded verzi databáze pro operační systém Android a iOS. Nevýhodou těchto embedded verzí jsou rozdílné programovací jazyky API, nelze tedy použít přístup multiplatformního vývoje. Tato databáze je nabízena pouze v placených verzích [9].

Shrnutí

Žádná z výše vyjmenovaných databází nenabízí embedded přístup z platforem iOS a Android s bezplatnou licencí. VelocityDB byla zamítnuta z důvodu téměř neexistující dokumentace a placené licence. Z důvodu neexistujícího vhodného bezplatného řešení embedded grafové databáze pro multiplatformní vývoj bylo odstoupeno od lokálního ukládání dat a byla zvolena databáze Neo4j. Tato databáze byla v březnu 2019 nejrozšířenější [10] grafově orientovanou databází na trhu. Neo4j byla zvolena také z důvodu bezplatného komerčního použití.

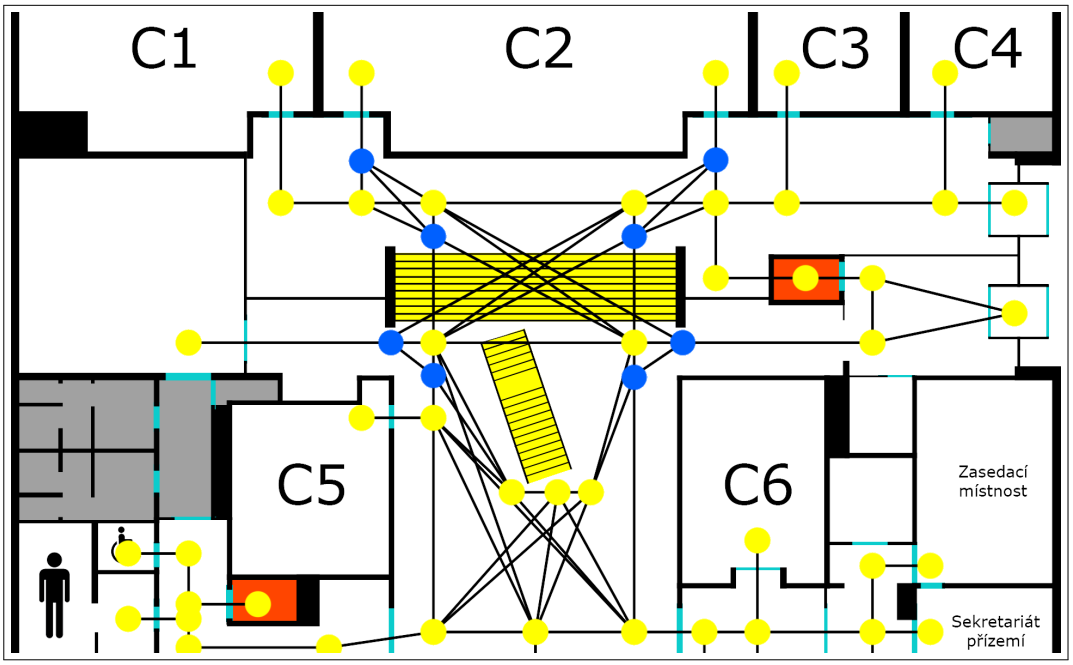
3.1.3 Backend technologie

Pro server-side část byl použit framework ASP.NET Core společnosti Microsoft. Jedná se o platformě nezávislý framework určený k vytváření webových rozhraní API.

3.1.4 Mapové podklady

Technické plány budov bývají pro uživatele příliš složité a mnohdy nepřehledné. Obsahují mnoho nadbytečných informací, jako např. rozvody elektrické či vodovodní sítě, které uživatel nutně nepotřebuje znát při orientaci v budově. V neposlední řadě tyto informace představují možné bezpečnostní riziko. Nejen z těchto důvodů je přinejmenším vhodné vytvořit vlastní plán vycházející z technických plánů budovy a do největší míry jej zjednodušit. Zjednodušené plány v této práci vycházejí z veřejně dostupných požárních plánů budovy C Přírodovědecké fakulty.

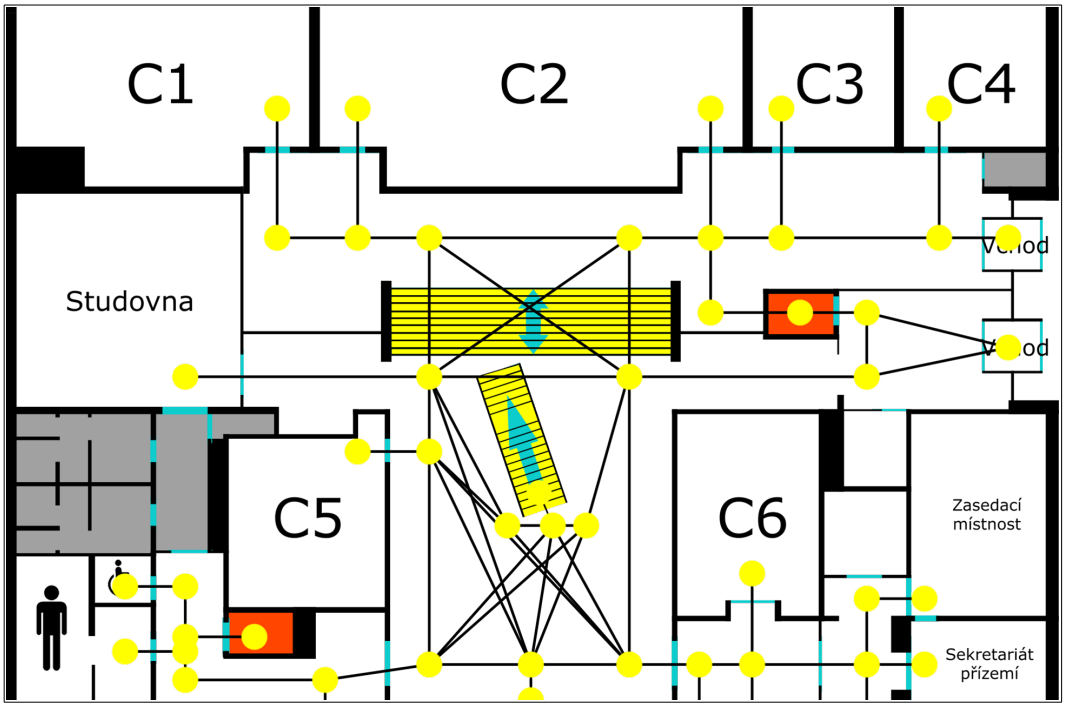
Dalším nezbytným krokem je vhodné rozmístění vrcholů reprezentující místnosti a chodby tak, aby vhodně reflektovaly všechny možné spojnice rozcestí. Tyto spojnice se totiž v aplikaci vykreslují jako trasa. Na Obrázku 3.1 lze vidět vybranou oblast s nadbytečnými vrcholy a spojnicemi, tyto vrcholy jsou vyznačeny jako modré body. Je nežádoucí vytvářet příliš mnoho nadbytečných vrcholů a spojnic, jelikož by to negativně ovlivnilo výkon výpočtu optimální trasy v databázi a následně by do mobilní aplikace bylo předáváno zbytečně mnoho vrcholů. V tomto případě by vykreslení trasy mohlo mít u méně výkonných zařízení vliv na výkon. Určení optimálního množství vrcholů je avšak velmi subjektivní.



Obrázek 3.1: Ukázka nadbytečného množství vrcholů

Obrázek 3.2 zobrazuje zjednodušený mapový podklad přízemí budovy C Přírodovědecké fakulty. Žluté body reprezentují místnosti a možná rozcestí, černé čáry spojující tyto body reprezentují možnou cestu mezi dvěma body. V případě, že body nejsou spojené, není mezi nimi možná cesta z důvodu překážky, nejčastěji stěny. Tyto žluté body jsou uložené v databázi jako vrcholy, černé čáry jako hrany. Každý vrchol uchovává informaci o své pozici na mapovém podkladu, patro, ve kterém se nachází a název fakulty a budovy. Významné orientační body, jako např. vchody, toalety, výtahy a vrátnice, nebo místnosti navíc uchovávají jméno.

Zjednodušený plán obsahuje také názvy místností a barevně odlišené významné části plánu. Šedá barva značí veřejně nepřístupné místnosti či chodby, které jsou přístupné pouze zaměstnancům, žlutou barvou jsou vyznačená schodiště, oranžovou barvou výtahy, světle šedou barvou jsou vyznačeny dveře. Každá místnost obsahuje název z portálů STAG nebo OrgStructure, v případě, že název není v těchto portálech uveden, je místnost popsána názvem ze štítku vyvěšeného u dané místnosti.



Obrázek 3.2: Ukázka vhodného rozložení vrcholů

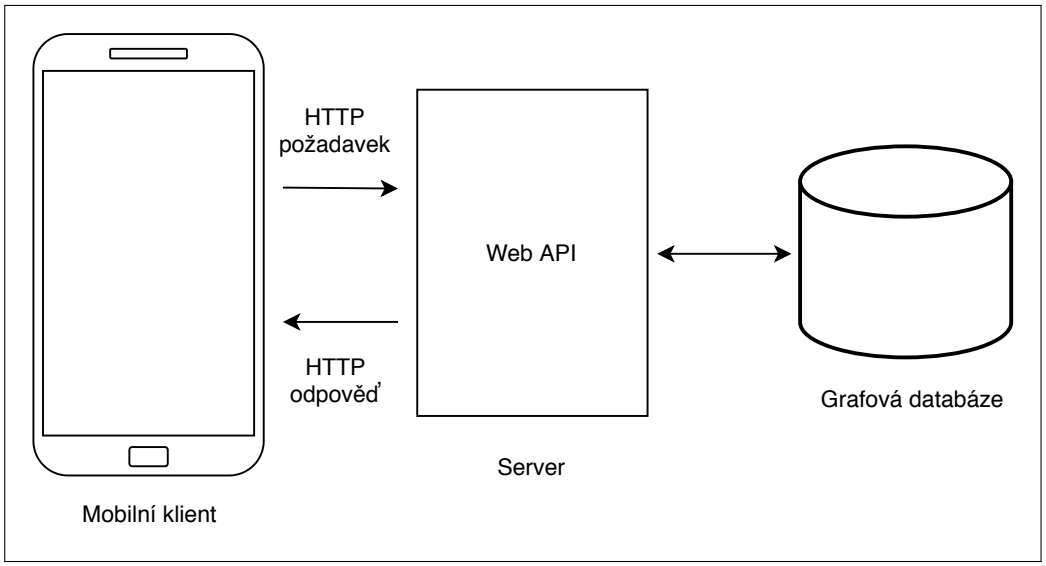
Převedení skutečných vzdáleností do mapy

Na serveru je počítána optimální trasa v centimetrech, ze zjednodušeného mapového podkladu získáme pouze vzdálenosti v obrazových bodech. Požární plán byl převeden do rozměrů 1949x2516 pixelů s rozlišením 300 DPI[7], následně bylo změřeno několik chodeb k získání a ověření převodového poměru. Bylo zjištěno, že 415 pixelů v upraveném plánu odpovídá 1003cm ve skutečnosti a následně byly vypočteny vzdálenosti mezi všemi propojenými vrcholy v daném patře. Díky této skutečnosti je možné získat délku trasy v centimetrech přímo z databáze bez dalších převodů.

3.2 Architektura systému

Tato podkapitola se zabývá architekturou systému, která je zobrazena na Obrázku 3.3. Mobilní aplikace posílá šifrované HTTP dotazy na Web API, které dotazy dále zpracuje a vrátí mobilní aplikaci odpověď. Zpracování dotazů Web API spočívá ve spuštění příslušných dotazů do databáze. Aktualizace dat probíhá z externího zdroje, který není součástí této práce.

[7]DPI – Obrazové body na palec



Obrázek 3.3: Architektura systému

3.3 Architektura mobilní aplikace

Při vývoji aplikace byl kladen důraz na dodržení architektury MVVM[8] dle nefunkčních požadavků. Jedná se o moderní architekturu mobilních aplikací. Hlavní myšlenkou této architektury je čisté oddělení business a prezentační prezentační logiky od uživatelského rozhraní. Udržování čistého oddělení mezi aplikační logikou a uživatelským rozhraním pomáhá řešit mnohé problémy s vývojem a umožňuje aplikaci snáze testovat, udržovat a vyvíjet [11].

Modelové třídy reprezentují datové objekty. Tyto třídy reprezentují business logiku aplikace. View reprezentuje uživatelské rozhraní. V ideálním případě by view nemělo obsahovat business logiku, výjimkou mohou být metody implementující vizuální efekty, jako jsou např. animace. ViewModel implementuje tzv. vlastnosti (anglicky *properties*) a příkazy (anglicky *commands*), na které se může view nabindovat, následně přes ně ViewModel notifikuje view o jakýchkoliv změnách přes notifikační události. ViewModel je zodpovědný za koordinaci interakcí view s modelem. Tyto třídy reprezentují prezentační logiku.

3.4 Wireframes

Drátěné modely neboli wireframes, se používají k nastínění funkcí a struktury jednotlivých obrazovek aplikace. Jedná se o zjednodušené modely, které nezobrazují konečnou grafickou podobu aplikace. Zamýšlená mobilní aplikace se skládá ze šesti hlavních obrazovek a čtyř vyskakovacích oken. Wireframy hlavních obrazovek jsou znázorněny na Obrázku 3.4.

[8]MVVM – Model-View-ViewModel

MainPage

Jedná se o výchozí obrazovku aplikace. Primárně slouží k zobrazení posledních pěti vyhledávaných výrazů. Po stisknutí položky je uživatel přesměrován na obrazovku *ResultPage*, kde je ve vyhledávacím panelu nastaven vyhledávaný výraz a ve výsledcích hledání je zobrazen vyhledávaný výraz. Dále má uživatel možnost stisknout tlačítko *Vybrat* u dané položky, které má stejnou funkci jako popisované stisknutí položky. V horní části obrazovky se také nachází vyhledávací panel, který poskytuje fulltextové vyhledávání, po stisknutí lupy je uživatel přesměrován na obrazovku *ResultPage*. V dolní části obrazovky se nachází tlačítko *Prohlížení* sloužící pro prohlížení mapových podkladů, které přesměruje uživatele na obrazovku *FindRoomByMapPage*.

ResultPage

Tato obrazovka slouží primárně k zobrazení výsledků hledaných uživatelem. V horní části obrazovky se nachází vyhledávací panel pro fulltextové vyhledávání, které obsahuje všechny zaměstnance, místnosti a důležité orientační body (např. výtahy nebo vchody), které uživatel může následně vybrat. Tyto výsledky se zobrazují ve zbytku obrazovky. Po stisknutí jednotlivých položek výsledků se uživateli zobrazí veškeré dostupné informace o hledaném výrazu. Po stisknutí tlačítka *Vybrat* je přesměrován na obrazovku *StartNavigationPage*.

StartNavigationPage

Obrazovka slouží k upřesnění parametrů vyhledávané trasy a zadání výchozí pozice uživatele. Zadání výchozí pozice lze provést pomocí výběru v horní části obrazovky třemi způsoby, a to:

1) Fulltextovým vyhledáváním.
2) Rozšířeným vyhledáváním.
3) Stisknutím bodu na mapě.

Fulltextové vyhledávání využívá obrazovku *ResultPage*, pouze s odlišným parametrem, díky této skutečnosti nedochází k nadbytečnému vytváření nových obrazovek. Zbylé způsoby výběru výchozí pozice jsou popsány níže. Dále se na této obrazovce nachází informace o vybrané výchozí pozici a cíli, popisující místnost, fakultu, budovu a patro. Ve spodní části obrazovky má uživatel možnost zvolit bezbariérovou trasu nebo použití výtahu, tyto hodnoty jsou ve výchozím nastavení vypnuty. Na této obrazovce se také nachází tlačítko *Spustit navigaci*, které po stisknutí a validním vyplnění výchozí pozice přesměruje uživatele na obrazovku *NavPage*.

FindRoomByAdvancedSearchPage

Tato obrazovka slouží k určení výchozí pozice pomocí upřesňovacích filtrů, je možné vyfiltrovat fakultu, budovu a podlaží, poté se zobrazí výsledný seznam místností odpovídající filtru.

FindRoomByMapPage

V této obrazovce, stejně jako v obrazovce *FindRoomByAdvancedSearchPage*, se nacházejí filtry, nicméně pro výběr místnosti se zde nachází tlačítko pro zobrazení mapového podkladu. Uživatel pomocí filtrů upřesní fakultu, budovu, patro a následně je přesměrován na obrazovku *SelectPointPage*. V případě, že je obrazovka *FindRoomByMapPage* spuštěna s parametrem pro získání výchozí pozice, uživatel může vybrat výchozí bod stiskem obrazovky na mapovém podkladu obrazovky *SelectPointPage* a následně je přesměrován na předchozí obrazovku. V opačném případě může uživatel pomocí stisku obrazovky na mapovém podkladu pouze zobrazit informace o dané místnosti.

NavPage

V této obrazovce se zobrazují zjednodušené plány pater budov s vykreslenou optimální cestou do cílové místnosti. V horní částí obrazovky se nachází tlačítko s ikonou info, která po stisknutí zobrazí vyskakující okno s informacemi o trase. V dolní části obrazovky se nachází dvě tlačítka, která po stisku přepínají mezi zjednodušenými plány budov použitými při vykreslení cesty. Mezi těmito tlačítky se nachází informace o momentálně zobrazeném patře ze všech potřebných k dosažení cíle.

MapLegendPopup

Toto vyskakovací okno slouží k zobrazování legendy mapového podkladu. V této obrazovce se zobrazuje vzdálenost trasy, ikony znázorňující výchozí bod, cílovou destinaci, následující patro a barevné vysvětlivky. Jednotlivé vysvětlivky jsou zobrazeny pomocí barev a ke každé je přidělen slovní popis, stejně tak i k ikonám.

RoomDetailPopup

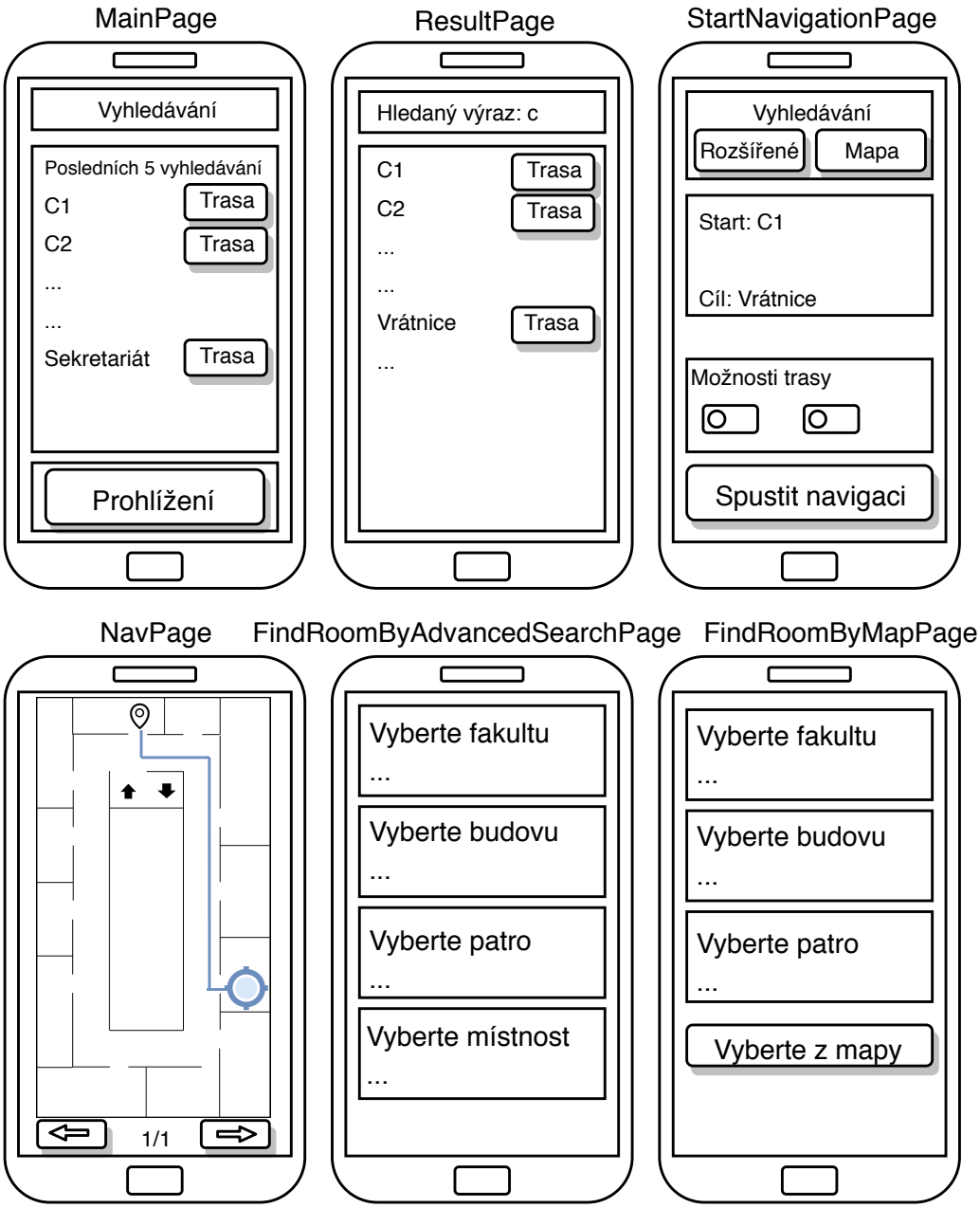
Toto vyskakovací okno zobrazuje informace o vybrané místnosti. Nachází se zde informace o fakultě, budově a patře ve kterém se místnost nachází, dále toto okno obsahuje seznam zaměstnanců sídlící v dané místnosti včetně jejich telefonu a emailu.

EmployeeDetailPopup

V tomto vyskakovacím okně se nachází informace o vybraném zaměstnanci. Toto okno obsahuje email a telefon zaměstnance, dále kancelář s informacemi o fakultě, budově a patře ve kterém se kancelář nachází.

ErrorPopup

Toto vyskakovací okno slouží k zobrazování chybových zpráv uživateli.



Obrázek 3.4: Wireframes mobilní aplikace

4. Implementace

Tato kapitola je věnována popisu implementace systému indoor navigační aplikace. Implementace reflektuje návrh popsaný v předchozí kapitole.

4.1 Zásady vývoje

Pro zachování srozumitelnosti a udržitelnosti kódu i pro osoby nepodílející se na vývoji, tudíž bez znalostí implementace, je nutné dodržovat definované zásady vývoje. Mezi definované zásady vývoje patří:

- Proměnné pojmenované dle specifikací daného programovacího jazyka,
- Proměnné pojmenované dle jejich významu,
- Metody pojmenované dle jejich funkce,
- Každá třída ve vlastním souboru pojmenovaná dle jejího významu,
- Veškerý obsah psaný v anglickém jazyce,
- Dokumentování kódu včetně složitých metod a částí kódu,
- Pravidelné verzování projektu,
- Dodržení správného odsazení vnořeného kódu,
- Dodržení architektury MVVM mobilní aplikace.

4.2 Vývojové prostředí

Pro implementaci aplikace bylo nezbytné zvolit vhodné vývojové prostředí. Pro vývoj bylo použito IDE[1] Visual Studio 2017 společnosti Microsoft, pro které nabízí Jihočeská univerzita svým studentům bezplatnou licenci. Toto vývojové prostředí nabízí virtuální zařízení s různými verzemi systému Android, které lze použít pro testování.

4.3 Správa kódu

Verzovací nástroje slouží k uchování historie všech provedených změn, tyto nástroje umožňují se kdykoliv k jakékoliv předchozí verzi vrátit. Při vývoji této práce byl použit verzovací nástroj Git se soukromým repozitářem, poskytovatelem webového repozitáře je webová služba *Gitlab* nabízející bezplatný hosting pro projekty [12].

[1] IDE – Integrované vývojové prostředí

4.4 Backend

V této podkapitole je popsána implementace back-endových technologií.

4.4.1 Web API

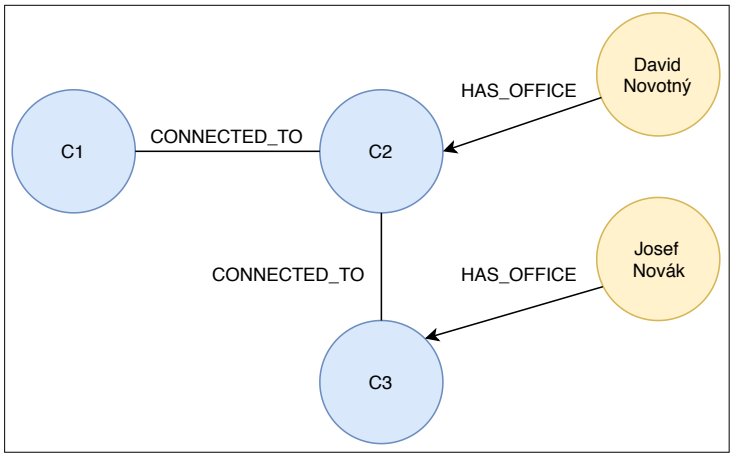
Hlavním úkolem web API je zpracovat požadavek od klienta a zaslat mu odpověď. Komunikace mezi klientem a Web API probíhá šifrovaně pomocí protokolu HTTPS. Zpracování požadavku probíhá spuštěním patřičných dotazů do databáze a následným zasláním výsledku klientovi.

4.4.2 Databáze

Na základě nefunkčních požadavků a návrhu popsaném v kapitole 3.1.2 byla zvolena grafově orientovaná databáze Neo4j jako řešení pro ukládání dat. Grafově orientované databáze se skládají z vrcholů a hran, které reprezentují vztah mezi vrcholy. V rámci zamýšlené aplikace vrcholy reprezentují místnosti, významné body a rozcestí, entita *Room*, a zaměstnance, entita *Employee*. Rozdíl mezi místností a rozcestím je ten, že místnost a významný bod mají navíc atribut *Name*. Významný bod je takový bod, který je důležitý pro orientaci, může jím být například vchod do budovy či výtah, přičemž ani jeden z nich není ve skutečnosti místnost jako taková.

V rámci zamýšlené aplikace jsou používány dva typy hran vyjadřující vztah mezi vrcholy. Prvním vztahem je *CONNECTED_TO*, který se používá k vyjádření vztahu mezi entitami *Room*. Tento vztah vyjadřuje, zda je mezi danými vrcholy možná cesta. Druhým vztahem je *HAS_OFFICE*, který vyjadřuje vztah mezi entitou *Room* a *Employee*. Tento vztah říká, že daný zaměstnanec sídlí v dané místnosti.

Na Obrázku 4.1 je znázorněná ukázka dat reflektující popis dat v předchozím odstavci. Entity *C2* a *C3* reprezentují místnost a jsou uloženy jako datový typ *Room*, tyto entity mají vztah *CONNECTED_TO*, tudíž mezi nimi existuje cesta. Entity *David Novotný* a *Josef Novák* reprezentují zaměstnance, jsou uloženy jako datový typ *Employee* a mají vztah *HAS_OFFICE* vůči místnosti *C2*, resp. *C3*, který říká, že sídlí v dané místnosti.



Obrázek 4.1: Ukázka dat

Popis implementace

Databáze Neo4j používá vlastní dotazovací jazyk Cypher, který se od rozšířeného jazyka SQL výrazně liší. Následující ukázka kódu zobrazuje vytvoření třech vrcholů, přičemž první vrchol *a1* reprezentuje místnost, jelikož má atribut *Name*, následující vrcholy *a2* a *a3* nikoliv. Následně jsou tyto vrcholy mezi sebou propojeny pomocí hran. U všech vrcholů reprezentující místnost a následujícím vrcholem je zvolena nulová vzdálenost, jelikož pro výpočet již není rozhodující. U všech ostatních hran je vzdálenost vypočtena způsobem popsaným v kapitole 3.1.4.

```
CREATE
    (a1:Room {Name: 'C1', PosX: '505', PosY: '424',
        Faculty: 'Přírodovědecká', FacultyShort: 'prf', Building: 'C',
        Floor: '0'}),
    (a2:Room {PosX: '505', PosY: '662', Faculty: 'Přírodovědecká',
        FacultyShort: 'prf', Building: 'C', Floor: '0'}),
    (a3:Room {PosX: '653', PosY: '662', Faculty: 'Přírodovědecká',
        FacultyShort: 'prf', Building: 'C', Floor: '0'}),
    ...
    (a1)-[:CONNECTED_TO {distance : 0 }]->(a2),
    (a2)-[:CONNECTED_TO {distance : 357 }]->(a3),
    ...
```

Ke komunikaci databáze Neo4j a backendu slouží knihovna Neo4jClient [15]. V případě knihovny Neo4jClient a použitím její třídy `GraphClient` se dotazovací jazyk Cypher mírně liší v zápisu. Následující ukázka metody `GetRooms` vyhledá všechny vrcholy *room* typu *Room*, najde všechny zaměstnance sídlící v dané místnosti, vybere pouze ty vrcholy, které nemají prázdný atribut *Name* a vrátí je jako typ *RoomTo*.

```
public List<RoomTo> GetRooms() {
    var result = dbFactory.Client.Cypher
        .Match("(room:Room), (employee:Employee)")
        .OptionalMatch("(employee)-[:HAS_OFFICE]->(room)")
        .Return((room, employee) => new RoomTo {
            Room = room.As<Room>(),
            Employees = employee.CollectAs<Employee>()
        }).Results.Where(n => n.Room.Name != null).ToList();
    return result;
}
```

4.4.3 Optimální trasa

Vyhledání optimální trasy probíhá na straně serveru. Z mobilního zařízení jsou pomocí volání API získány výchozí pozice, cílová destinace a parametry trasy. Následně je zavolána metoda `GetClosest` s argumenty výchozí pozice, cílové destinace a parametry trasy, která zjistí, zda v databázi neexistují více záznamů se stejným atributem *Name* jak pro výchozí pozici, tak i pro cílovou destinaci. Dále je vypočtena nejmenší vzdálenost pro každý pár s ohledem na parametry trasy. Tento mechanismus zajišťuje nalezení nejbližšího vchodu místnosti v případě, že má místnost více vchodů nebo nalezení nejbližší toalety v případě, že se jich v daném patře či budově nachází více. V poslední fázi je navrácena hodnota obsahující nejmenší vzdálenost. V případě, že v databázi neexistuje více záznamů se stejným atributem *Name*, je jako návratová hodnota zavolána metoda `GetPath`, která vrací optimální cestu pro daný výchozí bod a cílovou destinaci s ohledem na parametry.

Následující část kódu popisuje vyhledání nejkratší trasy mezi dvěma vrcholy.

```
public PathTo GetPath(string startID, string endID,
    bool barrierFree, bool elevator) {
    List<Room> rooms = new List<Room>();
    long distance = 0;
    string condition = barrierFree && elevator ? "r.barrierFree = true)" :
        elevator ? "r.elevator = true OR r.elevator IS NULL)" :
        "r.elevator = false OR r.elevator IS NULL)";

    using (var session = dbFactory.Driver.Session()) {
        session.WriteTransaction(tx => {
            var result = tx.Run("MATCH (start:RoomName:$startID)," +
                "(end:RoomName:$endID)," +
                p = shortestPath((start)-[:CONNECTED_TO*]-(end))" +
                "WHERE ALL(r in relationships(p) WHERE " + condition +
                "RETURN p", new { startID, endID });
            foreach (var record in result) {
                // zpracovani vysledku
            }
        });
    }
    return new PathTo() { Distance = distance, Rooms = rooms };
}
```

Při výpočtu nejkratší trasy se také vypočítává její cena, tedy skutečná vzdálenost mezi dvěma body. Výsledná optimální trasa a vzdálenost je zapouzdřena do transportního objektu `PathTo`, který je následně vrácen jako odpověď mobilní aplikaci.

Implementace parametrizace trasy

Parametrizace trasy je realizována kombinací booleovských hodnot parametrů `barrierFree` a `elevator` metody popsané výše a je realizována následujícími způsoby:

- **Bezbariérová trasa** - Vyhledávání prochází všechny hrany, kde atribut `barrierFree` má hodnotu `true`. Tato hodnota je nastavena u všech hran reflektující bezbariérovou trasu včetně výtahu. Výsledkem vyhledávání bude vždy bezbariérová trasa.
- **Výtah** - Vyhledávání prochází všechny hrany, kde atribut `elevator` má hodnotu `true` nebo `null`. Tato možnost bude vždy při výpočtu optimální trasy brát zohledňovat možnost použití výtahu.
- **Výchozí nastavení** - Vyhledávání prochází všechny hrany, kde atribut `elevator` má hodnotu `false` nebo `null`. Výsledkem tedy bude trasa neobsahující výtah.

4.5 Mobilní aplikace

V této podkapitole je popsána implementace mobilní aplikace. Tato kapitola také popisuje strukturu projektu a jeho celky.

4.5.1 Adresářová struktura projektu

Projekt mobilní aplikace obsahuje tři podprojekty, a to:

- NavJU - obsahující třídy a adresáře se sdíleným kódem napříč platformami,
- NavJU.Android - obsahující třídy a adresáře pro platformu Android,
- NavJU.iOS - obsahující třídy a adresáře pro platformu iOS,
- NavJU.Tests - obsahující testovací třídy.

Níže jsou popsány adresáře projektu *NavJU*, který obsahuje téměř 100 % kódu mobilní aplikace. Třídy jsou shlukovány do adresářů, z nichž každý adresář obsahuje třídy poskytující podobnou funkcionalitu či třídy patřící to stejné kategorie. Projekt *NavJU* obsahuje následující adresáře:

- **NavJU.Models** - třídy reprezentující datové objekty,
- **NavJU.Views** - třídy reprezentující uživatelské obrazovky,
- **NavJU.Views.Popup** - třídy reprezentující vyskakovací okna mobilní aplikace,
- **NavJU.ViewModels** - třídy reprezentující ViewModel.
- **NavJU.Services** - třídy servisní vrstvy,
- **NavJU.Data** - třídy zodpovědné za výměnu dat mezi mobilní aplikací a Web API.

4.5.2 Manipulace s gesty

V rámci implementace zamýšlené aplikace je důležité zajistit zpracování následujících akcí:

- Posun obrazovky,
- Přiblížení a oddálení obrazovky,
- Stisk obrazovky.

Zpracování gest lze řešit ve frameworku Xamarin Forms několika způsoby, avšak ne všechny disponují funkcionalitou potřebnou pro zamýšlenou aplikaci. Prvním z možných řešení je knihovna *SkiaScene.TouchManipulation* implementující knihovny *SkiaScene* a *TouchTracking* [17][18].

Knihovna podporuje manipulaci s SVG soubory, nabízí přiblížení, posun či stisknutí obrazovky. Problémem této knihovny je její špatná optimalizace a absence možnosti získat pozici stisknutého bodu při stisku obrazovky. Z těchto důvodů bylo od tohoto řešení odstoupeno.

Dalším možným řešením je využití knihovny *SkiaSharp* ke zpracování dotykových událostí. Tato knihovna je bezplatná a přímo podporuje manipulaci s SVG soubory [16]. Nevýhodou této knihovny je její špatná optimalizace a velmi složitá implementace zpracování gest. Z těchto důvodů bylo od tohoto řešení odstoupeno.

Dalším z možných řešení je knihovna *Mr.Gestures*, která není nabízena s bezplatnou licencí, nicméně licence není vázána na počet uživatelů či aktivací, nýbrž na název aplikace [19]. Tato knihovna nabízí široké API pro zpracování gest včetně všech výše zmíněných požadavků na zamýšlenou aplikaci, např. je zde již implementována akce dlouhého stisku obrazovky.

V následující podkapitole je popisován postup řešení přiblížení, resp. oddálení obrazovky. Ostatní gesta jako je např. posun nebo dvojité stisknutí obrazovky v této podkapitole popisované nejsou, jelikož se nejedná o složitější implementaci oproti přiblížení, resp. oddálení obrazovky.

Matematický popis

Prvním krokem je výpočet nového přiblížení, které lze získat vynásobením stávajícího přiblížení relativní vzdáleností mezi dvěma prsty dotýkající se obrazovky. Poměr přiblížení se získá poměrem hodnoty nového a stávajícího přiblížení.

$$noveP riblizeni = priblizeni \cdot relativniV zdalenost$$

$$pomerP riblizeni = \frac{noveP riblizeni}{priblizeni}$$

Dále je třeba vypočítat posun levého horního rohu, který vznikl při přiblížení. Nejprve je třeba vynásobit relativní vzdálenost mezi dvěma prsty osy X, resp. osy Y se šířkou, resp. výškou komponenty elementu zobrazující mapový podklad. Následně se od této hodnoty odečte posun matice plátna osy X, resp. osy Y.

$$posunX = (relativniV zdalenost_x \cdot element_{sirka}) - posunMaticePlatna_x$$

$$posunY = (relativniV zdalenost_y \cdot element_{vyska}) - posunMaticePlatna_y$$

Na závěr je nutné získat posun levého horního rohu v pixelech, aby nebylo nutné tuto hodnotu v dalších výpočtech převádět. Postup je vyjádřen následujícími vzorci.

$$novyPosun_x = posunX - pomerPriblizeni \cdot (posunX - posun_x)$$

$$novyPosun_y = posunY - pomerPriblizeni \cdot (posunY - posun_y)$$

Implementace řešení

Následující část kódu popisuje implementaci přiblížení, resp. oddálení obrazovky reflektující výše popsané vzorce. V rámci zamýšlené aplikace bylo také omezeno maximální přiblížení na trojnásobnou hodnotu.

```
private void StackLayout_Pinching(object sender,
        MR.Gestures.PinchEventArgs e) {
    if (model.Scale <= 3 && model.Scale >= 1) {
        float newScale = (float)(model.Scale * e.DeltaScale);
        float scaleRatio = newScale / model.Scale;

        float translatedX = (float)(e.DeltaScaleX * Width) -
            _canvasTranslateMatrix.TransX;
        float translatedY = (float)(e.DeltaScaleY * Height) -
            _canvasTranslateMatrix.TransY;

        float newX = translatedX - scaleRatio *
            (translatedX - model.TransX);
        float newY = translatedY - scaleRatio *
            (translatedY - model.TransY);

        if (newScale < 1 || newScale > 3) {
            return;
        }
        UpdateImageProperties(newX, newY, newScale);
    }
}
```

Výše popsaný algoritmus pro přiblížení, resp. oddálení je velmi důležitý, jelikož hodnota posunu levého horního rohu je důležitá mimo jiné i pro výpočty popsané v kapitole 4.5.4.

4.5.3 Vyskakovací okna

Framework Xamarin Forms nabízí pouze jednoduchá vyskakovací okna s velmi omezenou možností úpravy, z tohoto důvodu byla zvolena knihovna *Rg.Plugins.Popup*, která je nabízena s bezplatnou licencí [20]. Knihovna nabízí velké možnosti přizpůsobení, nabízí všechny komponenty frameworku Xamarin Forms, mimo jiné knihovna nabízí i řadu animací.

4.5.4 Získání pozice bodu vůči mapě

Při stisku obrazovky je nutné převést tuto pozici na pozici v mapě, jelikož obrazové jednotky, se kterými pracuje Mr.Gestures jsou jiné, než jsou jednotky mapových podkladů. Dalšími faktory, které je nutné brát v potaz je měřítko, se kterým se mapový podklad zobrazuje, odsazení komponenty zobrazující element mapového podkladu od okraje obrazovky a případný posun podkladu a jeho měřítko v případě přiblížení nebo oddálení obrazovky. Všechny tyto faktory je nezbytné uvažovat při výpočtu hodnoty na mapovém podkladu.

Matematický popis řešení

V úvodu je nutné převést stisknutý bod na požadované jednotky, v rámci této aplikace na pixely. Tyto hodnoty, označené jako `ptX` a `ptY`, lze získat vynásobením pozice X, resp. pozice Y stisknutého bodu šířkou, resp. výškou komponenty zobrazující element mapového podkladu a následným poměrem se šířkou, resp. výškou plátna.

$$ptX = \frac{element_{sirka} \cdot bod_x}{platno_{sirka}}$$

$$ptY = \frac{element_{vyska} \cdot bod_y}{platno_{vyska}}$$

V dalším kroku je nutné spočítat odsazení komponenty zobrazující element mapového podkladu od okraje obrazovky, stejná hodnota platí jak pro odsazení od levého okraje, tak i od horního okraje obrazovky. Tato hodnota je označena jako `offset`.

$$offset = \frac{obrazovka_{sirka} - element_{sirka}}{2}$$

Následně je třeba vzít v úvahu možné přiblížení obrazovky a hodnoty `ptX` a `ptY` vydělit hodnotou měřítka přiblížení, tyto hodnoty jsou označené jako `scaledPtX` a `scaledPtY`.

$$scaledPtX = \frac{ptX}{priblizeni}$$

$$scaledPtY = \frac{ptY}{priblizeni}$$

V neposlední řadě je nutné spočítat reálnou pozici levého horního kraje obrazovky pro případ, že obrazovka byla přiblížena, resp. oddálena či posunuta. Tyto hodnoty lze spočítat vynásobením posunu obrazovky po ose X, resp. po ose Y s měřítkem šířky, resp. výšky vykresleného mapového podkladu a následným poměrem hodnoty měřítka přiblížení.

Tyto hodnoty jsou označené jako `cornerX` a `cornerY`. Měřítko mapy lze získat poměrem šířky, resp. výšky mapového podkladu šířkou, resp. výškou komponenty zobrazující element mapového podkladu.

$$cornerX = \frac{posun_x \cdot meritkoMapy_{sirka}}{priblizeni}$$

$$cornerY = \frac{posun_y \cdot meritkoMapy_{vyska}}{priblizeni}$$

Na závěr je nutné zpracovat výše vypočtené hodnoty a získat z nich bod odpovídající mapovému podkladu. Nejprve je nutné vynásobit hodnotu `scaledPtX`, resp. `scaledPtY` měřítkem šířky, resp. výšky mapového podkladu. Následně se od této hodnoty odečte hodnota `offset` a hodnota `cornerX`, resp. `cornerY`. Vypočtené hodnoty jsou označené jako `translatedPtX` a `translatedPtY`.

$$translatedPtX = (scaledPtX \cdot meritkoMapy_{sirka}) - offset - cornerX$$

$$translatedPtY = (scaledPtY \cdot meritkoMapy_{sirka}) - offset - cornerY$$

Výše popsaným způsobem lze získat stisknutý bod a převést jej na bod odpovídající v mapovém podkladu ať už byla obrazovka posunuta, přiblížena, resp. oddálena nebo přiblížena, resp. oddálena a následně posunuta.

Implementace řešení

O výpočet hodnoty bodu vůči mapovému podkladu se stará statická metoda `CalculatePoint` ze třídy `TouchCalculator`, která je zobrazena v části kódu níže. Výpočty zobrazené v následující části kódu reflektují vzorce popsané výše.

```
public static SKPoint CalculatePoint(MainViewModel model,
        float pointX,
        float pointY) {
    float ptX = (float)(model.CanvasView.CanvasSize.Width *
        pointX / model.CanvasView.Width);
    float ptY = (float)(model.CanvasView.CanvasSize.Height *
        pointY / model.CanvasView.Height);
    float offset = (float)(Device.Info.PixelScreenSize.Width -
        model.CanvasView.CanvasSize.Width) / 2;

    float tappedX = ptX / model.Scale;
    float tappedY = ptY / model.Scale;

    float cornerX = model.TransX * ScaleConverter.PictureScaleX /
        model.Scale;
    float cornerY = model.TransY * ScaleConverter.PictureScaleY /
        model.Scale;

    float tappedScaledX = (tappedX * ScaleConverter.PictureScaleX) -
        offset - cornerX;
    float tappedScaledY = (tappedY * ScaleConverter.PictureScaleY) -
        offset - cornerY;

    return new SKPoint(tappedScaledX, tappedScaledY);
}
```

4.5.5 Získání výchozí pozice pomocí mapy

K uskutečnění tohoto způsobu musí uživatel zvolit alespoň budovu a patro, na kterém se nachází, poté se mu zpřístupní tato možnost výběru výchozí pozice.

Matematický popis řešení

K získání tohoto bodu je nutné znát rozměry plátna, rozměry zobrazované mapy a v neposlední řadě vybraný bod. Výběr bodu se realizuje pomocí dlouhého stisku a následného stisku tlačítka *Trasa* na zobrazeném vyskakovacím okně. Nicméně vybraný bod reprezentuje pouze pozici vůči obrazovce a jelikož nemusí odpovídat pozici v mapě, je nutné tuto pozici převést. Pozici v mapě získáme zavoláním statické metody `CalculatePoint` ze třídy `TouchCalculator` popsanou v předchozí podkapitole.

V dalším kroku je nutné nalézt nejbližší vrchol vůči vypočtenému bodu vráceným metodou `CalculatePoint`, to je realizováno pomocí vzorce Euklidovské vzdálenosti.

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Implementace řešení

V následující části kódu je zobrazena implementace výpočtu Euklidovské vzdálenosti v jazyce C# s nalezením nejbližšího bodu ze třídy `NodeService`.

```
public static Room FindClosestNode(MainViewModel model,
        string building, string floor,
        double pointX, double pointY) {
    var list = model.AllRooms.Where(room => room.Floor == floor &&
        room.Building == building).ToList();
    ...
    double minDistance = double.MaxValue;
    Room closestRoom = null;

    foreach (Room room in list) {
        double distance = Math.Sqrt((pointX - room.PosX) *
            (pointX - room.PosX) + (pointY - room.PosY) *
            (pointY - room.PosY));
        if (distance < minDistance) {
            minDistance = distance;
            closestRoom = room;
        }
    }
    if (minDistance > 400) {
        return null;
    }
    ...
    return closestRoom;
}
```

Metoda `FindClosestNode` z volající třídy získá z parametrů model, momentálně zobrazovanou budovu, patro a pozice X, resp. Y stisknutého bodu. Díky vstupním parametrům se v počátku metody vyfiltrují pouze ty místnosti, které jsou na daném patře a v dané budově, tato skutečnost zredukuje množství výpočtů Euklidovské vzdálenosti. Výpočet Euklidovské vzdálenosti probíhá v cyklu `foreach`, který pro každou položku z kolekce provede výpočet a porovná, zda vypočtená hodnota není nižší než doposud vypočtená nejnižší hodnota. V závěru metody je podmínka, která ošetřuje případy, kdy vstupní bod je příliš vzdálen od nejbližší místnosti, tato skutečnost zajišťuje toleranci při nepřesném kliknutí.

4.5.6 Optimální trasa

Mobilní aplikace se zabývá pouze vykreslením nejkratší trasy, za samotný výpočet je zodpovědný server. Aplikace po zadání výchozí a cílové pozice zavolá metodu `FindPath` ze třídy `PathFinding`, která vrací výsledek volání API ze třídy `RestService`. Volání API aplikaci vrátí kolekci všech bodů trasy s odpovídajícími hodnotami X a Y vůči levému hornímu rohu mapy. Jelikož mají mobilní zařízení různá rozlišení a různé poměry stran, je nutné vzít tyto skutečnosti v potaz. Před vykreslením samotného mapového podkladu se nastaví statické *properties* `ScaleX` a `ScaleY` ze třídy `ScaleConverter` na hodnotu poměru šířky plátna a výchozí šířky mapového podkladu, resp. výšky plátna a výchozí výšky mapového podkladu. Díky takto řešenému přepočtu měřítka je zajištěna nezávislost na rozlišení mapových podkladů.

4.5.7 Vykreslování

Mapové podklady byly zpracovány pomocí vektorové grafiky, díky tomu se zachovává kvalita obrázku i při přiblížení. Samotné vykreslení je implementováno pomocí knihoven SkiaSharp a SkiaScene. Knihovna SkiaScene rozšiřuje knihovnu SkiaSharp o možnost použití vektorových souborů SVG.

Aplikace vykresluje pouze momentálně zobrazované patro, to je realizováno prostřednictvím dotazovacího jazyka LINQ[2] v metodě `RenderPath` třídy `Pathrendering`, pomocí kterého se vyfiltrují pouze ty body nalezené optimální cesty, které jsou shodné s momentálně zobrazeným patrem. Díky statickým *properties* `ScaleX` a `ScaleY` je zaručeno vykreslení bodů na správných pozicích na zařízeních s různým rozlišením. Hodnoty `ScaleX`, resp. `ScaleY` lze získat poměrem šířky, resp. výšky komponenty zobrazující element mapového podkladu šířkou, resp. výškou mapového podkladu. Dále je třeba brát v potaz posun mapového podkladu a přiblížení, respektive oddálení obrazovky. Tato závislost je popsána následujícími vzorci.

$$bod_x = poziceBodu_x + \frac{\frac{posun_x}{ScaleX}}{priblizeni}$$

$$bod_y = poziceBodu_y + \frac{\frac{posun_y}{ScaleY}}{priblizeni}$$

Vykreslení trasy následně probíhá vykreslováním čáry v cyklu, vždy se vykreslí čára mezi momentálně zpracovávaným bodem a následujícím. Tato metoda je také zodpovědná za vykreslení označení výchozí pozice a cílové destinace v případě, že se jeden z těchto bodů nachází na právě vykreslovaném patře.

[2]LINQ – Language Integrated Query

4.5.8 Vykreslování doplňujících informací

V mobilní aplikaci se při zobrazení optimální trasy vykresluje nejen trasa, ale také doplňující informace, jenž jsou reprezentovány pomocí ikon ve formátu SVG. Tyto ikony doplňují informace k vykreslené trase, např. výchozí bod uživatele, cílová destinace či následující patro. O vykreslování přislušných ikon se stará metoda `RenderIcons` ze třídy `Pathrendering`, která má mimo jiné parametr obsahující všechny body pro momentálně zobrazené patro, tím je zaručeno, že se ikony vykreslí na správných místech pro dané patro. Určení pozice pro vykreslení příslušné ikony probíhá tak, že se vybere první a poslední element kolekce obsahující body pro dané patro a následně se pro vybrané elementy vykreslí příslušné ikony.

5. Testování

Aplikace byla otestována dle případů užití na fyzickém zařízení OnePlus 5 s operačním systémem Android 9.0 Pie. Dále byla aplikace testována ve virtuálních zařízeních prostředí Visual Studio 2017 ve verzích Android 8.1 a Android 5.0. Aplikace byla testována průběžně při vývoji. V rámci mobilní aplikace byly vytvořeny unit testy. REST API endpointy byly testovány manuálně vývojářem v průběhu vývoje.

Testován byl také operační systém iOS. Testování probíhalo na virtuálních zařízeních prostředí Xcode na fyzickém zařízení Apple Macbook Air. Vzhledem k tomu, že testování iOS aplikací vyvíjeních ve frameworku Xamarin Forms je možné provádět pouze za přítomnosti fyzických počítačů společnosti Apple a autor této práce měl toto zařízení pouze zapůjčené po omezenou dobu, testování bylo provedeno až ke konci vývoje mobilní aplikace.

Na závěr byla aplikace poskytnuta pěti uživatelům k důkladnému otestování. Uživatelům byl poskytnut uživatelský manuál, testovalo se především správné zpracování gest a vyhledání a vykreslení optimální trasy. Uživatelům byla poskytnuta testovací data obsahující mapové podklady dvou pater, které obsahovaly dvacet místností a dvacet zaměstnanců.

Závěr

Výsledkem této práce je funkční aplikace, která umožňuje vyhledávat místnosti a zaměstnance Jihočeské univerzity, nalézt optimální trasu s ohledem na parametry a zobrazit dostupné informace o místnostech a zaměstnancích.

Aplikace poskytuje intuitivní prostředí, které bylo navrženo s důrazem na jednoduchost a snadné ovládání. Tvorba mapových podkladů probíhala s důrazem na přehlednost, jednoduchost a jednoznačnost. Rozmístění vrcholů reprezentující místnosti a chodby bylo provedeno s ohledem na vytvoření co nejvíce možných cest při zachování co nejmenšího počtu vrcholů. Aplikace také umožňuje prohlížení mapových podkladů.

V počátku vývoje byly sestaveny základní požadavky na aplikaci, následně byly vytvořeny případy užití. Na základě případů užití a doplňkového dotazníku byly sestaveny funkční požadavky. Následně byly vybrány vhodné technologie, sestaveny wireframy a navržena architektura systému. Vývoj probíhal dle zvolené metodiky Scrum, jednotlivé úkoly byly přidány do backlogu, byla jim přiřazena priorita a následně dle priority byly naplánovány do sprintů. Vývoj probíhal v moderním vývojovém prostředí a byl pravidelně verzován pomocí nástroje Git.

Uživatelská dokumentace je realizována pomocí přiložené textové dokumentace popisující snímky obrazovek.

Aplikace je připravena na rozšíření o více dat a příslušných mapových podkladů. Díky implementaci je aplikace závislá pouze na datech a příslušných mapových podkladech, je tedy možné přidat jakoukoliv budovu či patro s jakýmkoliv rozlišením mapového podkladu. Aplikace je také připravena na rozšíření o automatické určování pozice uživatele, nicméně tato funkcionalita nebyla cílem této práce.

Na základě výše uvedených skutečností lze konstatovat, že cíle bakalářské práce byly splněny.

Seznam použité literatury

[1] *Home / Scrum Guides* [online]. 2018 [cit. 2019-11-07]. Dostupné z: https://www.scrumguides.org/

[2] *What is a Daily Scrum?* [online]. 2019 [cit. 2019-11-07]. Dostupné z: https://www.scrum.org/resources/what-is-a-daily-scrum

[3] *Git* [online]. 2019 [cit. 2019-11-30]. Dostupné z: https://git-scm.com/

[4] Dua, Kinjal. *A Guide to Mobile App Development: Web vs. Native vs. Hybrid* [online]. 2018 [cit. 2019-03-16]. Dostupné z: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/

[5] Saccomani, Pietro. *Native Apps, Web Apps or Hybrid Apps? What's the Difference?* [online]. 2018 [cit. 2019-03-16]. Dostupné z: https://www.mobiloud.com/blog/native-web-or-hybrid-apps/

[6] *Multiplatformní vývoj mobilních aplikací – druhy a nástroje / Blog / Krosapp* [online]. Krosapp, 2018 [cit. 2019-03-16]. Dostupné z: http://www.krosapp.cz/Blog/multiplatformni-vyvoj-mobilnich-aplikaci-druhy-nastroje

[7] *Neo4j Documentation* [online]. Neo4j, 2019 [cit. 2019-03-16]. Dostupné z: https://neo4j.com/docs/

[8] *Why OrientDB / Open Source NoSQL Multi-model Database / OrientDB* [online]. Callidus Software, 2019 [cit. 2019-03-16]. Dostupné z: https://orientdb.com/why-orientdb/

[9] *Sparsity-technologies: Sparksee high-performance graph database* [online]. Sparsity Technologies, 2019 [cit. 2019-03-16]. Dostupné z: http://www.sparsity-technologies.com/

[10] *DB-Engines Ranking - popularity ranking of graph DBMS* [online]. 2019 [cit. 2019-03-17]. Dostupné z: https://db-engines.com/en/ranking/graph+dbms

[11] *The Model-View-ViewModel Pattern - Xamarin / Microsoft Docs* [online]. 2017 [cit. 2019-11-20] Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[12] *Gitlab* [online]. 2019 [cit. 2019-11-30] Dostupné z: https://about.gitlab.com/

[13] *Xamarin.Forms - Xamarin / Microsoft Docs* [online]. Microsoft, 2019 [cit. 2019-03-16]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/

[14] *Dokumentace k ASP.NET / Microsoft Docs* [online]. Microsoft, 2019 [cit. 2019-03-16]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/

[15] *GitHub - Readify/Neo4jClient: .NET client binding for Neo4j* [online]. 2019 [cit. 2019-11-11] Dostupné z: https://github.com/Readify/Neo4jClient

[16] BRITCH, David, Craig DUNN a Charles PETZOLD. *SkiaSharp Graphics in Xamarin.Forms - Xamarin | Microsoft Docs* [online]. Microsoft, 2019 [cit. 2019-03-16]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/graphics/skiasharp/

[17] KUNC, Ondřej. *GitHub - OndrejKunc/SkiaScene: Simple API to transform SkiaSharp objects using functions like zoom and rotate or using gestures like pan and pinch.* [online]. 2017 [cit. 2019-03-16]. Dostupné z: https://github.com/OndrejKunc/SkiaScene

[18] BRITCH, David. *Vyvolání událostí z efektů - Xamarin | Microsoft Docs* [online]. 2018 [cit. 2019-11-09] Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/app-fundamentals/effects/touch-tracking

[19] RUMPLER, Michael. *MR.Gestures - Handle all the touch gestures in your Xamarin.Forms mobile apps* [online]. 2018 [cit. 2019-11-08]. Dostupné z: https://www.mrgestures.com/

[20] LYUBIMOV, Kirill. *GitHub - rotorgames/Rg.Plugins.Popup: Xamarin Forms popup plugin* [online]. 2019 [cit. 2019-11-08]. Dostupné z: https://github.com/rotorgames/Rg.Plugins.Popup

Seznam obrázků

Přílohy

Přiložený komprimovaný soubor obsahuje zdrojové kódy vytvořených aplikací, elektronickou verzi textu této práce a uživatelskou dokumentaci.