VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FITNESS MAPOVÁ APLIKACE

FITNESS MAP APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ HRŮZA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Hrůza Tomáš**
Program: Informační technologie
Název: **Fitness mapová aplikace**
Fitness Map Application
Kategorie: Web

Zadání:

1. Nastudujte funkčnost webových a mobilních aplikací pro fitness (Strava, Endomondo, Garmin, atd.). Nastudujte problematiku OpenStreet Maps a GPS (včetně výměnných formátů).
2. Navrhněte webovou aplikaci, která bude na mapě zobrazovat data ze sportovních trackerů (např. Garmin nebo Strava), případně vytvořte jednoduchou mobilní aplikaci, ze které data nahrajete. Umožněte v jedné mapě zobrazit všechny trasy (vyfiltrované z více GPS záznamů) a také podle Voronoiových diagramů "obsadit" území kolem trasy.
3. Navrženou webovou aplikaci vytvořte a otestujte.

Literatura:

- fitness webová aplikace, strava.com, [online], cit. 20.10.2021
- OpenStreetMap, openstreetmap.org, [online], cit. 20.10.2021

Pro udělení zápočtu za první semestr je požadováno:

- první dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Svět se neustále vyvíjí a moderní doba vyžaduje moderní řešení. Společně s vývojem technologií se mění i očekávání jejich uživatelů. Velké množství populace se snaží udržet zdravý životní styl, co má za následek tvorbu množství různých elektronických zařízení a aplikací pro usnadnění životosprávy.

Účelem bakalářské práce je poskytnout uživatelům fitness aplikací nový způsob motivace ke sportu a to ve formě webové aplikace. Motivace je ve formě minihry, kde uživatel prochází světem a aplikace mu následně přiřadí území pomocí tzv. Voroného diagramu, které zabral svoji aktivitou. Další funkcí aplikace je vytvoření průměrné trasy z velkého množství tras, které vedly podobnou polohou. Toto se hodí v případě, kdy má uživatel zájem aproximovat skutečnou trasu, kterou pravidelně prochází za pomocí průměrování jeho nasbíraných dat.

Součástí práce jsou informace o různých moderních a využívaných fitness aplikacích, ze kterých byla vybrána aplikace Strava pro propojení databází a získání uživatelských dat. V praxi to znamená, že uživatel využívá mobilní aplikaci Stravy pro zaznamenání jeho aktivit a následně je může pomocí vytvořené webové aplikace zobrazit.

Práce pokračuje podrobným popisem postupu pro získání dat a skripty, které byly využity pro výpočet Voroného diagramu České republiky. Tyto data jsou následně vložena do databáze. Při vývoji aplikace je velký důraz kladen na uživatelský komfort. Po implementaci a spuštění aplikace je provedeno testování jednotlivých funkcí a výsledky jsou vidět v příloze k práci.

Abstract

As the world continues to evolve and expand, the need for modern solutions rapidly grows. Large portion of the population tends to keep track of their lifestyle which in turn provides an opportunity for companies to create many different fitness devices and applications.

The goal of this bachelor's thesis is to provide a new kind of motivation to get out into the world, provided by yet another web application. However this application provides its user with motivation in the form of a minigame – the user travels through the world and the application then calculates the area he captures using a Voronoi diagram based on his route. Another feature the application provides is viewing your routes in a merged, averaged form. This comes in handy when the user would like to approximate his actual route over several different activities.

The first part of this bachelor's thesis collects information about different commonly used fitness applications. Out of all those a web application called Strava was chosen as a base for collecting user data. The resulting web application connects to Strava using OAuth and retrieves routes the user has collected using the Strava mobile application.

One of the last chapters focuses on describing the implementation details about used and created algorithms. It additionally contains description of scripts which were used to extract data out of OpenStreetMap database and calculate Voronoi regions based on them. To store and search through calculated regions, a MySQL database is used. The application is developed using modern frameworks and is heavily oriented around a comfortable user interface and responsive web design.

Klíčová slova

OSM, OpenStreet Maps, GPS, GeoJson, Strava, webová aplikace, Python, fitness, Voronoi, Overpass, API, Voroného diagram, C#, Entity Framework, Minimal API.

Keywords

OSM, OpenStreet Maps, GPS, GeoJson, Strava, web application, Python, fitness, Voronoi, Overpass, API, Voronoi Diagram, C#, Entity Framework, Minimal API.

Citace

HRŮZA, Tomáš. *Fitness mapová aplikace*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Fitness mapová aplikace

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Hrůza
11. května 2022

Poděkování

Rád bych poděkoval panu Ing. Jaroslavu Rozmanovi, Ph.D. za vedení mé bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Svět se neustále vyvíjí a moderní doba vyžaduje moderní řešení. Společně s vývojem technologií se mění i očekávání jejich uživatelů. Velké množství populace se snaží udržet zdravý životní styl a to má za následek tvorbu množství různých elektronických zařízení a k nim aplikací pro usnadnění životosprávy.

Účelem bakalářské práce je poskytnout uživatelům fitness aplikací nový způsob motivace ke sportu a to ve formě webové aplikace. Motivace je ve formě minihry, kde uživatel prochází světem a aplikace mu následně přiřadí pomocí tzv. Voroného diagramu území, které zabral svoji trasou. Další funkcí aplikace je vytvoření průměrné trasy z velkého množství tras, které vedly podobnou polohou. Toto se hodí v případě, kdy má uživatel zájem aproximovat skutečnou trasu, kterou pravidelně prochází za pomocí průměrování jeho nasbíraných dat.

Druhá kapitola obsahuje rozvedené informace o současných moderních aplikacích pro fitness. Zde jsou popsány způsoby, jakými získávají koncept GPS a základní teorie o databázi *OpenStreetMap*. V dalších sekcích jsou popsané různé funkce a vlastnosti, které uživatelům poskytují současné fitness aplikace na trhu. Mezi nimi bylo nutné rozhodnout, které budou využity jako hlavní zdroj uživatelských dat pro bakalářskou práci. Zde jsou zváženy i například zveřejněné průzkumy na webových stránkách od jiných uživatelů. V poslední části je popsán samotný koncept Voroného diagramu, který je využit v pozdější fázi implementace.

V následující kapitole je rozebrána tématika vytváření webových aplikací. Zde jsou analyzovány akce, které by aplikace měla uživateli zprostředkovat ve formě modelu případu užití. V sekci 3.2 jsou rozebrány moderní technologie, ze kterých si v dnešní době vývojáři vybírají a jejich případné výhody či nevýhody. Tato sekce je dále rozdělena dle typu informací na serverové a klientské části. V následující sekci jsou popsané druhy databází, které by mohly vyhovovat potřebám aplikace. V předposlední části jsou zváženy různé knihovny, které poskytují zobrazení geografických map a v jakých formátech pracují. Kapitola je ukončena návrhem samotného uživatelského grafického rozhraní.

Kapitola 4 detailně popisuje použité technologie, algoritmy, vytvořené skripty a náhledy do zdrojových kódu. V rámci bakalářské práce je v první sekci znázorněný postup pro vytvoření vlastní instance serveru pro zprostředkování vyhledávání mezi jednotlivými Voroného regiony. Dále je zde popsané, jakým způsobem je možné vyextrahované data vložit do již existující databáze. V sekci 4.4 je probírané téma implementace navázání spojení a přístupů k uživatelským datům, následně je řešena otázka škálování při zpracování dat. Samotná sekce zpracování v sobě obsahuje velké množství implementačních detailů a problémy, které byly řešeny v době vývoje serverové části. V poslední řadě jsou obdobně vysvětleny detaily a různé funkce, které byly implementovány pro uživatelský komfort v klientské části.

V předposlední kapitole je znázorněno testování různých uživatelských funkcí zmíněných v modelu případu užití. Aplikace procházela intenzivním testováním již během samotného vývoje, kdy bylo nutné neustále kompilovat zdrojové kódy pro ověření funkčnosti. Zde popsané testy mají ve většině případů reference na přílohu D, která obsahuje vytvořené obrázky jako výstup testování.

Na konci práce je přidáno několik příloh, které usnadní zájemcům, kteří by chtěli aplikaci otestovat práci. V příloze A se vyskytuje manuál ke spuštění, kde jsou popsané nutné prerekvizity a příkazy pro kompilaci aplikace. Příloha C zobrazuje možné vložení souboru dat do databáze pomocí *Microsoft SQL Server Management Studia*.

Kapitola 2

Problematika OpenStreetMap, GPS a Voroného regionů

V kapitole je popsána teoretická stránka geografických informací a jejich získání, které tvoří základ dat, které aplikace požaduje.

2.1 OpenStreetMap

OpenStreetMap je projekt zabývající se budováním volně dostupné geografické databáze světa. Jejím cílem je získat informační záznam o každém geografickém rysu. Zpočátku se jednalo hlavně o mapování cest a silnic, delší dobu už ale obsahuje i informace o polních cestách, budovách, poštovních schránkách, vodních tocích, lesích, plážích atd. Pro potřeby dnešní doby také obsahuje data o autobusových linkách, různých obchodech a dalšími daty infrastruktury. Drtivá většina databáze je vytvořena díky spolupráci veřejnosti a příspěvkům uživatelů OpenStreetMap. [7]

Tyto data se sbírají například během provozování cyklistiky nebo i chůzí po cestách a v přírodě. Pro záznam těchto dat je využita různá nositelná elektronika (angl. *Fitness Trackers*). Ty se nejčastěji vyskytují ve formě chytrých hodinek a náramků, které jsou spojeny s mobilní nebo webovou aplikací. Většina nositelné elektroniky využívá tzv. *globální polohový systém* neboli GPS ke sběru dat. [11]

2.2 Globální polohový systém

Na počátku stála myšlenka touhy po lehkém určení polohy při cestování. Principiálně GPS satelity vždy znají svoji vlastní polohu. Ke každému zařízení, které se ptá na jeho polohu jsou přiřazeny čtyři satelity. Tři jsou využity pro koncept resekce (angl. *Navigation Resection*)[1], pro výpočet polohy zařízení. Čtvrtý satelit zajišťuje správnou funkci časového čítače přijímacího zařízení. [18]

[1] Výpočet polohy pomocí resekce: https://www.armystudyguide.com/content/army_board_study_guide_topics/land_navigation_map_reading/find-your-location-using-.shtml

Historie

V prosinci 1973 byl schválen program GPS. Během následujících několika let bylo na orbit vysláno několik satelitů. V roce 1993 počet satelitů dosáhl 24. Právě díky těmto satelitům je veřejnosti přístupná technologie globálního zjišťování polohy. [51]

Od této doby se GPS stal nezbytnou součástí infrastruktury celosvětově. Je využíván v letectví, námořnictví, pro armádní účely, ale i v běžném životě pro navigaci pěší nebo v autě za pomoci např. mobilních a webových aplikací. [14]

V roce 1990 vláda USA zavedla selektivní dostupnost. Díky tomu byla přesnost pro GPS systémy pro civilní užití snížena na okolí 100 metrů. V následujících letech si ovšem vláda uvědomila, jaké výhody má GPS i pro civilní využití, v roce 2000 byla selektivní dostupnost zrušena a od té doby se přesnost rapidně zvýšila. [38]

V současné době většina nositelné elektroniky využívá přijímač na jedné GPS frekvenci. To poskytuje uživateli přesnost v rozmezí 4.9 metrů pod širým nebem. Pro profesionální použití lze přesnost dále zvýšit za pomocí více GPS frekvencí, ale kvůli provozní ceně se dvě frekvence nedají využít pro příležitostné uživatele. [17]

2.2.1 Formát GPS

Pro přenos a uchování dat GPS se používá systém zeměpisných souřadnic (angl. *Geographic Coordinate System*). Jednotlivé souřadnice jsou složeny ze zeměpisné šířky a délky. Šířka je úhel mezi olovnicí v daném bodě a rovinou rovníku. Délka je úhel vzepětí mezi rovinou nultého poledníku a rovinou poledníku daného bodu na zemském povrchu. Z předchozí definice vyplývá, že samotný systém je ovlivněný tvarem Země. [43]

2.3 Aplikace vyžívající GPS

2.3.1 Endomondo

Endomondo byla aplikace od populární fitness firmy *Under Armour*. Vyskytovala se výhradně jako mobilní aplikace, existovala placená i neplacená verze. V neplacené verzi se vyskytovaly reklamy, které mohli uživateli znepříjemnit její používání. Jedna z oblíbených funkcí byla možnost malého odpočtu během aktivity. Poskytovala uživateli motivaci k většímu výkonu po určitý časový interval, většinou v rámci vteřin. Jednou z nejvíce chválených funkcí bylo propojení s dalšími uživateli, ti mohli v reálném čase mezi sebou komunikovat a to i během nahrávání aktivit. Aplikace měla možnost povzbuzujících hlášek, které si mohli uživatelé navzájem posílat i během běhu. [42]

K záznamu pohybových dat aplikace využívala GPS polohu, kterou získala z GPS zařízení v mobilním telefonu uživatele. Pro zobrazení dat využívala *Google Maps*. Po dokončení aktivity je bylo možné exportovat do *.gpx* souborů, které jsou popsány v sekci 3.4.1. Podpora aplikace *Endomondo* byla zrušena k 31. prosinci 2020 a byla nahrazena sesterskou aplikací *MapMyRun*. [42]

2.3.2 MapMyRun

MapMyRun[2] byla vytvořena a cílena na komunitu v zahraničí, v USA. Úsudek vychází z přednastavených hodnot při prvním přihlášení a následně implicitní pozice mapy. Po vytvoření uživatelského účtu a pokusu najít vytvořené cesty uživatelů v okolí Brna vyšlo

[2]MapMyRun stránka: https://www.mapmyrun.com/dashboard.

najevo, že v České Republice tato aplikace nemá velké klientské zastoupení, doporučené trasy jiných uživatelů byly až deset let staré – z roku 2012.

Jak bývá mezi fitness aplikacemi obvyklé, pro záznam aktivit se využívá mobilní aplikace. Po zapnutí aplikace je viditelné „Start“ tlačítko, které zapne nahrávání tempa a polohy. Aplikace umožňuje také spočítat spálené kalorie po dokončení aktivity nebo vytvoření šablony tréninkové trasy. Samozřejmostí je uživatelské přizpůsobení domovské stránky, lze nastavit metrický systém nebo třeba formát dnešního data. V placené verzi aplikace poskytuje zvukovou zpětnou vazbu jako je čas, uběhlá vzdálenost a tempo během aktivity. V nástroji lze nastavit cíle, na které uživatele bude aplikace následně upozorňovat, pokud se k nim atlet během aktivity přiblíží. [13, 28]

MapMyRun poskytuje oficiální API (angl. *Application Programming Interface*) pro vývojáře, ovšem pro registrování vlastního API je nutno vyslovit zájem o získání vlastního klíče a poté existuje šance, že povolení bude uděleno. Společně s malým množstvím českých uživatelů by bylo její využití nepraktické.

2.3.3 Garmin

Garmin je firma, která se soustředí na fitness doplňky a životní styl. Problematikou zobrazování uživatelských dat se zaobírá až webová aplikace nazvaná *Garmin Connect*[3]. Stránka vypadá moderně, ale nemá žádné funkce, které by dále zpracovávaly údaje o jednotlivých trasách. Poskytuje uživateli motivaci dále sportovat pomocí cílů, které si může nastavit. Vyskytuje se i možnost sdílet aktivity s přáteli v reálném čase.

Pro záznam polohy *Garmin* využívá svoje vlastní zařízení, jedná se o různé druhy nositelné elektroniky jako jsou hodinky a náramky. Vyrábí i samostatné zařízení GPS, které našly využití i pro vojenské využití. Zařízení od *Garmin* z principu nevyužívají paměťové karty a data o aktivitě zaznamenává do své pevné paměti. Elektronika často obsahuje USB port pro připojení k počítači, zde lze poté využít aplikaci *Garmin Express* pro zobrazení nasbíraných dat. [32]

Garmin a vývojáři

Firma *Garmin* vytvořila vlastní protokol FIT (angl. *Flexible and Interoperable Data Transfer*)[4]. Pomocí tohoto protokolu lze sdílet data mezi různými platformami a zařízeními. Využívá se pro záznam a přenos nasbíraných fitness dat, např. mezi počítačem a mobilem v souborech formátu *.FIT*. [20]

Garmin dále poskytuje přístup k aktivitám uživatele pomocí tzv. REST API[5] (angl. *Representational State Transfer Application Programming Interface*). Pro přístup k této technologii je ovšem nutné být registrovanou firmou a teprve poté lze zažádat o povolení. Pokud by aplikace měla být v budoucnosti publikována širokému publiku, tak se Garmin dá považovat jako možné rozšíření. Ovšem v rámci bakalářské práce je její využité obdobně jako aplikace *MapMyRun* velice nepraktické.

[3] Garmin Connect: https://connect.garmin.com/modern/.
[4] FIT Protokol: https://developer.garmin.com/fit/overview/
[5] Garmin REST API: https://developer.garmin.com/gc-developer-program/activity-api/.

2.3.4 Strava

Pokud člověk vyhledá na webu „Fitness aplikace“, tak jedno z nejvíce opakujících se jmen bude Strava. Strava je poskytnuta uživatelům ve formě webové, ale i mobilní aplikace. Vyskytuje se zde možnost předplatného, kdy následně uživatel dostane přístup k více funkcím a statistikám. Uživatel zaznamenává data výhradně pomocí propojené mobilní aplikace. Před začátkem nahrávání aktivity se Strava zeptá na její typ, správný výběr aktivity následně umožňuje detailnější výpočet uživatelských statistik. Aktivity mohou být například běh, chůze, cyklistika, ale i různé sporty. V dalších sekcích aplikace je možné zúčastnit se „Doporučených aktivit,“ což jsou trasy již zaznamenány a doporučeny jinými uživateli. Samotný proces navrhnutí takovéto trasy se poté vyskytuje ve webové aplikaci. Uživatel má v mobilní aplikaci také možnost znovu spuštění jeho již jednou provedených aktivit za účelem zlepšení času na trati. Pokud má uživatel zájem, může si přidat ostatní uživatelé jako přátelé a následně společně vytvářet a plnit různé fitness výzvy.

Při nahrávání aktivity má uživatel v aplikaci možnost výběru ze dvou druhů zvukových zpětných vazeb. V neplacené verzi aplikace uživateli sděluje jeho současné tempo např. každého půl kilometru, toto chování se odvíjí dle nastavení. Dále zde existuje nástroj *Live Segment Performance*, který uživateli dovolí si předem zvolit, ve kterém úseku by chtěl dostávat zvukové připomínky. Tento nástroj si pečlivý uživatel může nastavit jako pomocnou sílu v obtížných úsecích, jako je např. běh do strmého kopce. [13]

Webová aplikace vypadá velice moderně a poskytuje velké množství funkcí, které by průměrný uživatel očekával. Funkce jako zobrazení již existujících aktivit a jejich správa, je samozřejmostí. Dále zde má uživatel možnost vytvářet vlastní trasy, jak bylo dříve zmíněno. Nástroj pro vytváření má dodatečné funkce na rozdíl od podobného nástroje v *MapMyRun*. Při práci s nástrojem uživatel může povolit funkci doporučení tras. Následně uživateli Strava aktivně navrhuje další zajímavá místa, které jsou oblíbené a doporučeny dalšími uživateli. Tyto populární ulice a místa se také dají zobrazit pomocí jiného režimu mapy, tzv. **teplotní mapa**, (angl. *Heatmap*)[6]. Oddálenou mapu střední Evropy lze vidět v obrázku 2.1. Pomocí zabarvení cest zobrazuje, jak často se uživatelé pohybují ve stejných místech, např. ulicích. Aktivováním předplatného lze vidět data pro přihlášeného uživatele, v neplacené verzi jsou to pouze obecná data všech uživatelů. [50]

Strava a vývojáři

Strava poskytuje oficiální a veřejně dostupné API, které mohou vývojáři použít k práci s daty jednotlivých uživatelů[7]. Postup registrace vlastní aplikace je dále popsaný v sekci 4.3. Sekce 4.4.2 poskytuje podrobnější popis autorizace uživatelům. Z důvodu zvýšení bezpečnosti citlivých uživatelských údajů Strava využívá protokol *OAuth2* pro autorizaci jednotlivých uživatelů do aplikace třetí strany. Po schválení autorizace uživatelem získává aplikace přístup k jeho aktivitám z databáze Stravy.

[6] Heatmapa v aplikaci Strava: https://www.strava.com/heatmap.
[7] Strava, pro vývojáře: https://developers.strava.com/.

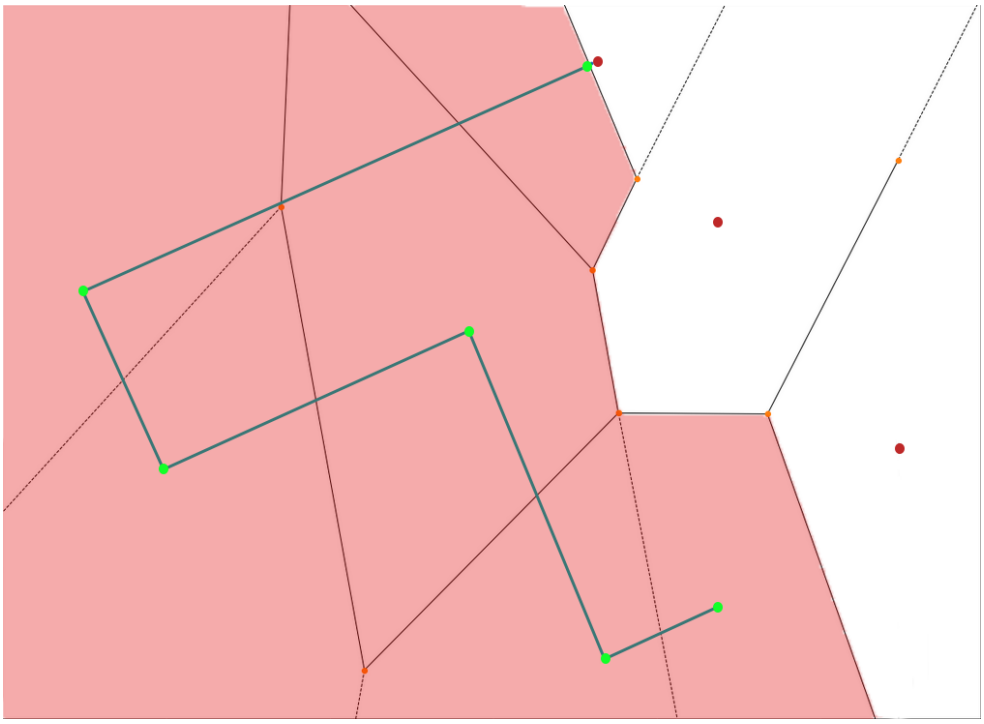
Obrázek 2.1: Heatmapa střední Evropy dle aplikace Strava

2.4 Voroného diagram

Voroného diagram je rozdělení prostoru na základě množiny bodů M. Množina všech bodů prostoru, které jsou blíže k prvku a, $a \in M$ než jakémukoliv jinému bodu z množiny M se nazývá Voroného region. V obrázku 2.2 lze vidět zabrané území uživatelem s modrou barvou trasy. Následně je jeho region vybarvený červenou barvou. Zelené a červené tečky znázorňují středy Voroného regionů. Jednotlivé regiony byly vypočítány pomocí knihovny *Scipy*. [27]

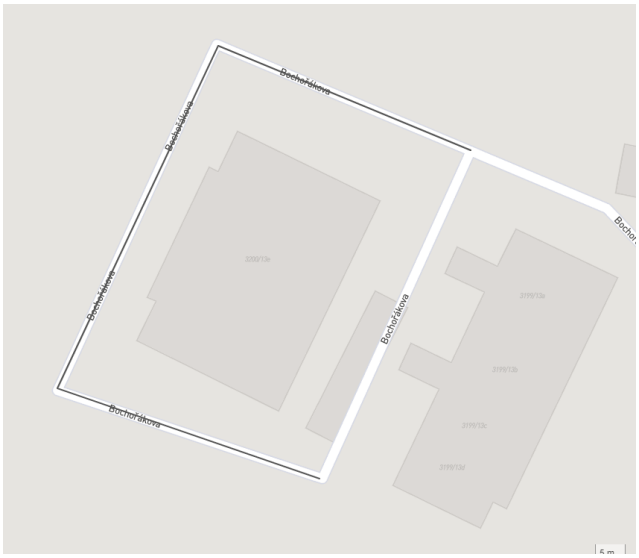
Pro využití Voroného regionů k přiřazení území, které uživatel zabírá, bylo nutno připravit několik elementů:

- Knihovna *Scipy* a její funkce *Voronoi*. Tato funkce počítá hranice jednotlivých Voroného regionů. [8]
- Množina M – databáze OSM (angl. *OpenStreetMap*) obsahující informace o cestách.
- Trasy uživatelů.

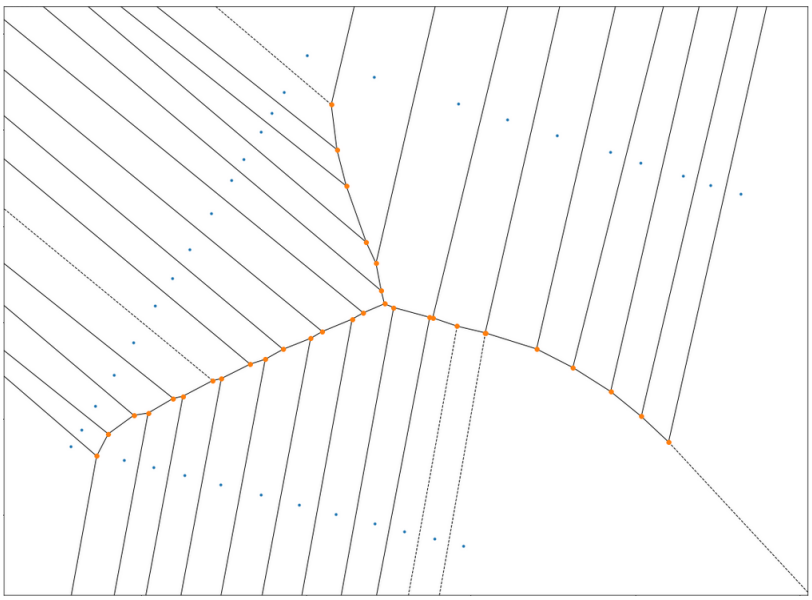


Obrázek 2.2: Vyplnění regionů dle Voroného diagramů

Pro přehlednější ukázku funkce *Voronoi* bylo využito prostředí *Jupyter Notebook*[8]. Původní trasa byla ulice Bochořáková, která je viditelná v obrázku 2.3. Výsledek výpočtu dle funkce lze vidět v obrázku 2.4, zde jsou středy regionů znázorněné modrou tečkou.



Obrázek 2.3: Ulice Bochořákova



Obrázek 2.4: Voroného regiony zobrazené pomocí Jupyter Notebook

[8]Jupyter Notebook dostupný z https://jupyter.org/.

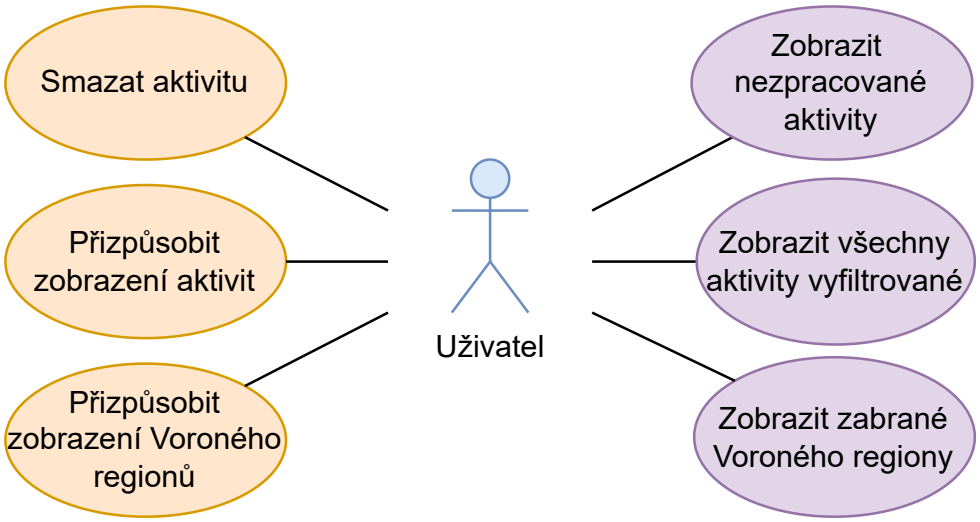
Kapitola 3

Návrh webové aplikace

3.1 Analýza webové aplikace

Webová aplikace se bude nejvíce soustředit na zobrazení různých zpracovaných dat. Je tedy důležité, aby grafický design byl uživatelský příjemný a poskytoval uživateli různé volitelné přizpůsobení. Je vhodné, aby si webovou stránku uživatel také mohl zobrazit z mobilního telefonu, a tedy je zde požadavek responzivity. Akce nutné implementovat pro uživatele dále popisuje model případů užití (obrázek 3.1).

Model případu užití



Obrázek 3.1: Model případů užití

3.2 Webové frameworky

Framework je obecně základní struktura pro systém. V programování je framework nástroj, který poskytuje komponenty, objekty, nástroje a nebo další programy, které jsou připravené pro použití vývojářem. Často jsou strukturovány do tzv. knihoven, jazyk ve kterém jsou napsané záleží hlavně na cílovém použití. Jejich cílem je usnadnit vývoj a poskytnout standard pro programování na nízké úrovni, aby se vývojáři mohli soustředit na implementaci sekcí, které učiní jejich projekt výjimečným. Využití frameworku zvyšuje rychlost programování a zjednodušuje testování. Frameworky aktivně vyvíjené jeho komunitou často mívají náskok i v rámci zjednodušení poskytnutí bezpečnosti a podpory nových vývojářů. Hlavní úskalí může nastat, pokud vývojáři nejsou příliš zkušení s jazykem, na kterém je framework postaven. Další úskalí může nastat, pokud se vývojář až příliš spoléhá na daný framework a přestává implementovat svůj vlastní kód. [44]

Frameworky jsou obecně rozděleny dle jejich specifikace na *Client-Side (Frontend)* a *Server-Side (Backend)* frameworky.

3.2.1 Frontend frameworky

V současné době je na trhu velké množství různých frameworků. Některé jsou principiálně odlišné, některé jsou zaměřené na mobilní aplikace a další pro webové aplikace. Frotendové frameworky jsou z velké části založeny na kombinaci HTML, CSS a JavaScriptu.

- **HTML** (angl. *Hypertext Markup Language*) je značkovací jazyk, používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. Obsahuje strukturu tzv. HTML značek, které tvoří strukturu drtivé většiny webových stránek. [24]
- **CSS** (angl. *Cascading Style Sheets*) je jazyk pro aplikování stylu zobrazení elementů HTML na webových stránkách. [5]
- **JavaScript** je skriptovací jazyk pro obohacení webových stránek o různou funkcionalitu, úzce spolupracuje s HTML elementy na webových stránkách. [16]

React

React je framework vytvořený vývojáři Facebooku. Dle *Stack Overflow Developer's survey 2021* se jedná o nejpoužívanější framework pro tvorbu uživatelského rozhraní. [49]

Hlavním cílem frameworku bylo vylepšit udržovatelnost kódu webových stránek. Jedna z nejvíce vyzdvihovaných vlastností je tzv. *virtual DOM* (angl. *virtual Document Object Model*). To reprezentuje ideální vzhled uživatelského rozhraní, které je následně synchronizováno pomocí knihovny *ReactDOM* promítnutím různých stavů. [45]

Výhody	Nevýhody
• rozdělení do komponentů	• horší dokumentace díky rapidnímu vývoji
• otevřený kód	• delší doba učení
• změna stavu pomocí vDOM	

React je jednodušší pro použití vývojáři, kteří mají dobrý základ vědomostí JavaScriptu. [41]

Angular

Angular je jednoduchý framework založený na *TypeScriptu* firmou Google. Oproti *React*, *Angular* funguje pomocí dvou-směrného vázání dat (angl. *Two-way Data Binding*). Je zde časově závislá synchronizace mezi zobrazením uživateli a modelem stránky. *Angular* je také framework často používaný pro vývoj mobilních aplikací. Oproti *React* má *Angular* podrobnější dokumentaci, ale zároveň je mnohem složitější a hůře se v ní orientuje.

Obecně je tento framework složitější na implementaci i naučení, je doporučený pouze pro větší aplikace, které jsou vyvíjeny většími týmy. [41]

Vue 3

Vue 3 poskytuje vývojáři deklarativní a programovací model na bázi komponentů. Jeho hlavní výhody jsou:

- *Declarative Rendering*, které rozšiřuje HTML syntaxi pomocí šablon. Ty popisují deklarativně výstup vygenerovaného HTML dle stavu JavaScriptových proměnných.
- *Reactivity* znamená, že implementovaná aplikace automaticky sleduje změny proměnných a efektivně aktualizuje webovou stránku pro uživatelský komfort.
- *Single-File Components* při procházení webových stránek v aplikaci se uživateli stránka neobnovuje, pouze je reaktivně přepočítána a zobrazena.

Webová aplikace se skládá ze souborů s příponou *.vue*, které obsahují zdrojové kódy tvořící jednotlivé komponenty. Webová stránka se skládá z těchto komponentů, což umožňuje přehlednou organizaci složek a jednoduché opakované využití elementu ve více různých zobrazení. Technologie dále chytře aktualizuje a přepisuje současně zobrazenou stránku pomocí kombinace URL (angl. *Uniform Resource Locator*) neboli webová adresa a dříve zmíněných proměnných. Ve výsledku uživatel neprochází stránky, takže se zamezuje dlouhému načítání a generování stránky po kliknutí. [53]

3.2.2 Backend frameworky

Ohledně zpracování serverové části, zvažované frameworky se hlavně odvíjí od zvoleného jazyku programování. Omezení se zde nekladou, backend se dá implementovat pomocí *Python, PHP, Ruby, NodeJS, C#, Java* atd. Pro potřeby aplikace byly zvažovány hlavně jazyky *Python* a C#.

ASP.NET Core

ASP.NET Core je neplacený framework s otevřeným kódem založený na jazyku C#. Obsahuje multiplatformní podporu pro vývoj webových aplikací mezi Windows, Linux a Mac. Backend následně využívá naprosto stejný kód mezi všemi platformami a není jej potřeba měnit. Využité technologie v *ASP.NET Core* umožňují programovat za využití méně kódu než obvykle. Vývojáři často označují tento framework jako pohodlný, jelikož je potřeba implementovat menší množství příkazů. Kód je taky dobře udržovatelný, pokročilí vývojáři jsou schopni nutnou údržbu backendu zredukovat bez větších problémů. Ovšem asi největší výhoda je výkon, jelikož C# má obecně velice dobré výsledky při výpočetních úkonech v porovnání s ostatními kompilovanými jazyky. [15]

Django

Django je framework založený na jazyku *DPython*. Následuje systém MVC (angl. *Model-View-Controller*). *Django* je vhodný pro vývoj sofistikovaných webových aplikací silně závislých na databázi (angl. *Database Driven*). [15]

Object-Relational-Mapping umožňuje vývojáři psát přímo jednoduchý kód v pythonu pro tvorbu tabulek, které jsou následně přeloženy do příslušného jazyka databáze, například SQL. Tento systém se také nazývá *Code First approach* dále zmíněný v sekci 4.1.2. V případě *Django* jsou jednotlivé definice tabulek vloženy do jednoho souboru, často nazvaný *models.py*. Toto chování se dá považovat za oboustrannou zbraň, jelikož v případě komplexní databáze se soubor může lehce stát nepřehledným. [37]

3.3 Databáze

Databáze je uspořádaná soustava dat, obecně může být v papírové nebo elektronické formě. V případě webové aplikace se samozřejmě jedná o elektronickou formu. Požadavek uspořádání dále vyžaduje, aby bylo možné soustavu dat lehce upravovat, přidávat do ní další data a nebo v ní vyhledávat. V moderní době existuje velké množství typů databázi a tato sekce dále popisuje některé ze zvažovaných. [31]

3.3.1 MySQL

MySQL je jedna z nejvíce používaných relačních databází. Relační databáze ukládají data do jednotlivých tabulek, ty následně mají mezi sebou určité vztahy, tzv. „relace.“ Struktura databáze je dále rozdělena do fyzických souborů, které jsou optimalizované pro rychlost operací. Vývojář nastavuje pravidla relací mezi jednotlivými tabulkami, např. jeden ku jedné (angl. *One-to-one*), jeden k mnoha (angl. *One-to-many*) nebo mnoho k mnoha (angl. *Many-to-many*). Dále jsou zde tzv. primární a cizí klíče, které jednoznačně identifikují data a relace v databázi. MySQL dodržuje veškerá uživatelská pravidla bez výjimky a zaručuje velice dobrou konzistenci dat. K přístupu do databáze je využito jazyku SQL (angl. *Structured Query Language*), dle vyvíjeného prostředí lze využívat čisté SQL dotazy nebo také existují různé překladače API závislé na programovacím jazyku. [40]

3.3.2 MongoDB

Potřeba objektově orientované databáze vyrostla spolu se zvětšujícím se využitím objektově orientovaných programovacích jazyků při komunikaci s databází. OOD (angl. *Object-Oriented Database*) dokáže pracovat se složitějšími strukturami, takové se kterými se vývojáři často potýkají při využití objektově orientovaného programování. Objekty, stejně jako v programování, mohou mít různé vlastnosti a metody. Databáze jako MySQL jednotlivé objekty překládají a serializují vložené data, mezi kterými následně vytváří relace. Zatímco objektové databáze jako MongoDB si do databáze ukládají přímo celý objekt. [35]

3.4 Formáty geografických dat

V sekci jsou popsány detailnější informace o formátech často využívaných pro uchování a následné zobrazení geografických dat fitness aplikacemi.

3.4.1 GPX

GPX (angl. *GPS Exchange Format*) je jeden ze standardů použitých k popisu geologických informací. Používá syntaxi XML (angl. *Extensible Markup Language*) k ukládání GPS souřadnic a informací o trasách. Zařízení často používají rozdílné formáty pro uchovávání dat. GPX poskytuje vývojářům možnost převodu mezi těmito formáty. [52]

Následující útržek kódu byl převzat ze stránky *Fileformat* [19]:

```
<trk>
  <name>Example GPX Document</name>
  <trkseg>
    <trkpt lat#"47.644548" lon#"-122.326897">
      <ele>4.46</ele>
      <time>2009-10-17T18:37:26Z</time>
    </trkpt>
    <trkpt lat#"47.644548" lon#"-122.326897">
      <ele>4.94</ele>
      <time>2009-10-17T18:37:31Z</time>
    </trkpt>
  </trkseg>
</trk>
```

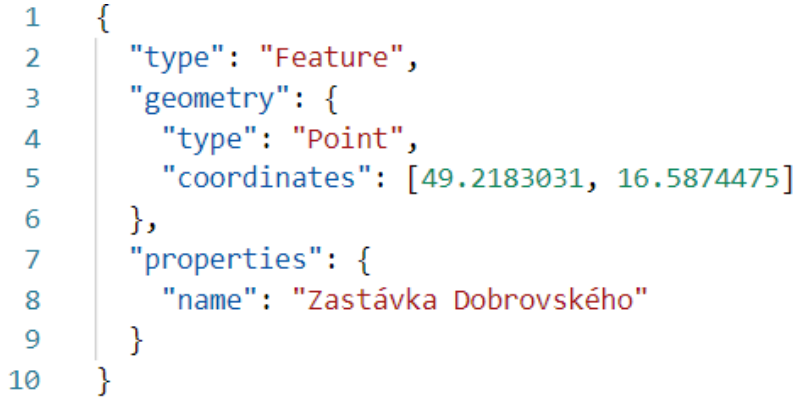
Z útržku lze vyčíst, že trasy se mohou skládat z jednotlivých segmentů, označených pomocí značky *<trkseg>*. Segmenty jsou dále rozdělené na body, které jsou založeny na uživatelově GPS poloze. Navíc si tyto body ukládají informace o čase jejich pořízení a změny nadmořské výšky od prvního bodu v aktivitě.

3.4.2 Geojson

Struktura GeoJSON[1] umožňuje zapsat objekty do syntaxe podobné široce známému formátu JSON. Tyto objekty dále mohou znázorňovat body, úsečky nebo také mnohoúhelníky. Formát GeoJSON je využívaný knihovnou *Leaflet*, o které je možné více informací nalézt v sekci 3.5.2.

Na obrázku 3.2 lze vidět uzel dat vytažený z OSM databáze a konvertovaný do GeoJSON. Konkrétně se jedná o autobusovou zástavku v Brně. Podobně jako *.GPX* je soubor rozdělený do bodů, které jsou následně spojeny dle nastavení nadřazeného typu (angl. *Type*). V obrázku je typ aktivity je nastavený jako bodový, ale zde je možné i nastavit dříve zmíněné úsečky nebo mnohoúhelníky.

[1] GeoJSON: https://geojson.org/.



Obrázek 3.2: Příklad GeoJSON syntaxe

3.5 Zobrazení získaných dát

V sekci jsou rozebrány různé možnosti poskytnutí zobrazení dat uživateli, které jsou zváženy pro další vývoj.

3.5.1 OpenLayers

OpenLayers je knihovna napsaná v JavaScriptu. Dle názvu lze poznat, že se nástroj pohybuje v tzv. vrstvách (angl. *Layers*). Různé vrstvy jsou navrženy pro různé typy zobrazení dat. Vrstvy umožňují jednodušší zobrazení uživateli, koncept lze vysvětlit na příkladu z knížky *OpenLayers 2.10 Beginner's Guide*:

> Představte si dva lidi, kteří se dívají na mapu města. Jeden z nich se zajímá o autobusové trasy zatímco druhý má zájem o cyklistické trasy. Místo vytváření dvou map lze vytvořit dvě vrstvy na jedné mapě a následně si uživatel sám vybere, zda chce vidět pouze jednu nebo dokonce obě vrstvy naráz. [23]

Pro zobrazení dokáže knihovna zpracovat větší množství různých typů dat. Například může používat GML (angl. *Geography Markup Language*), GeoJSON nebo v ní lze také jednoduše zobrazit pole souřadnic jako různé útvary – mnohoúhelníky. Při definici je na výběr velké množství zdrojů samotného obrázku mapy, jako jsou například Google Mapy a OpenStreetMap. [33]

3.5.2 Leaflet

Leaflet je volně dostupná knihovna napsaná v JavaScriptu. Pomocí knihovny lze zobrazit geografická data, např. GPS body na mapě. Tato mapa je externě zobrazena díky databázi OpenStreetMap. Pro zlepšení uživatelského komfortu lze nastavit různé parametry, jako např. úroveň přiblížení a stylování objektů pomocí CSS. Syntaxe zápisu objektů do mapy je definována pomocí výměnného formátu GeoJSON. [1]

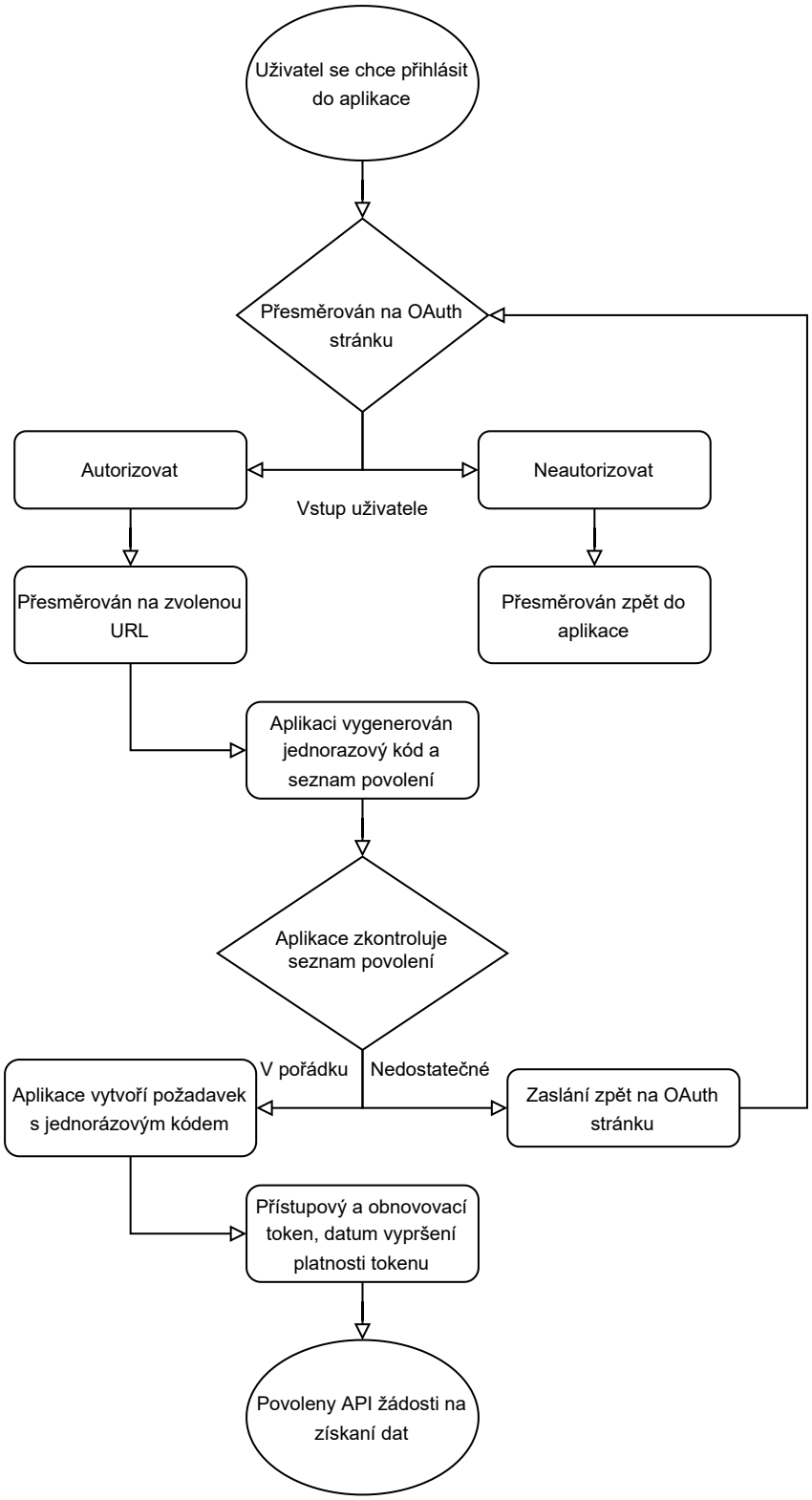
3.6 Návrh získaní dat

V rámci práce bylo zváženo vytvoření mobilní aplikace. Samotný vývoj jednoduché aplikace pro sbírání polohy by nebyl časově náročný, ale aplikace by se nevyrovnala již existujícím alternativám bez vložení početného množství hodin a práce. Proto byl zvolen přístup využití již existující aplikace, kdy hlavní požadavek byl existence moderní mobilní aplikaci pro záznam dat. Dle průzkumu ze sekce 2.3 byla zvolena aplikace Strava.

Pro získání uživatelských dat je doporučené využít volně dostupné API z aplikace Stravy. Zde je jediná podmínka být registrovaný uživatel, následně má každý jednotlivec možnost si zaregistrovat vlastní API instanci pro svůj účet. Pro zaregistrování je potřeba mít připravený název, kategorii a webovou adresu. Webová adresa nemusí být nutně platná, zde se dá vyplnit i adresa lokálního serveru (angl. *Localhost*). API Strava generuje tzv. OAuth autorizační stránku po obdržení požadavku, ten musí mít validní parametry pro identifikaci registrované instance API. Uživatel je na této stránce srozuměn s povoleními a rámcem přístupu, které uděluje fitness aplikaci. Po úspěšné autorizaci dostává aplikace přístup k datům uživatele. Tento proces je názorně zobrazen na obrázku 3.3. [50]

Strava API má omezenou režii, tj. 100 požadavků během 15 minut a maximálně 1000 za 24 hodin. [50]

Pro potřeby bakalářské práce je toto množství požadavků dostačující, ovšem pro produkční použití by bylo potřeba vyjednat úmluvu se Stravou.



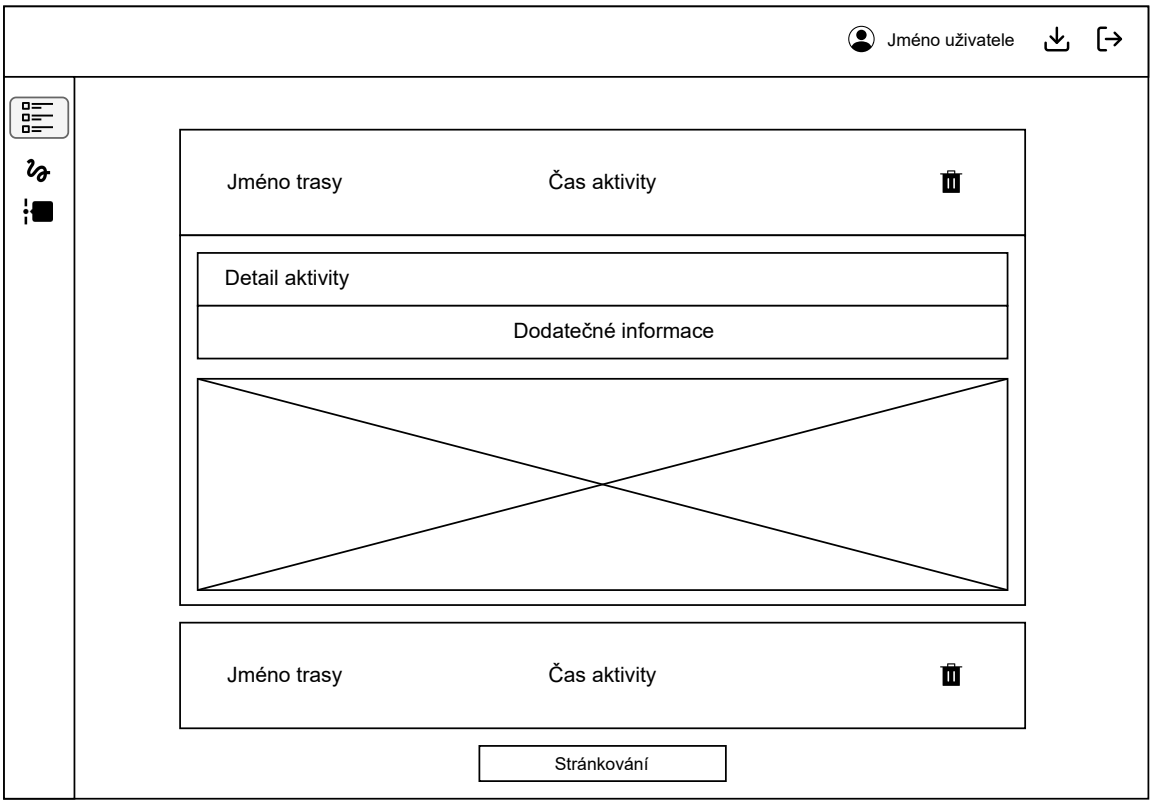
Obrázek 3.3: Diagram autorizace uživatele

3.7 Návrh uživatelského rozhraní

Po přístupu na webovou stránku uživatele uvítá přihlašovací obrazovka. Dle předešlých vysvětlení je možné k autorizaci využít API Stravy. Na této stránce je z daného důvodu nutné implementovat pouze elementy k přesměrování. Struktura aplikace bude rozdělena na 3 části – navigační menu vlevo, hlavička s uživatelským jménem a možností odhlášení. Zbytek stránek tvoří dynamicky se měnící obsah. Navigační menu se bude nacházet na levé straně aplikace, po najetí kursorem se rozevře a zobrazí uživateli názvy jednotlivých webových stránek. V hlavičce se bude vyskytovat název aplikace, uživatelské jméno a tlačítko pro odhlášení. Po kliknutí na jméno nebo ikonku uživatele se zobrazí dodatečné funkce uživatelského účtu.

Seznam aktivit

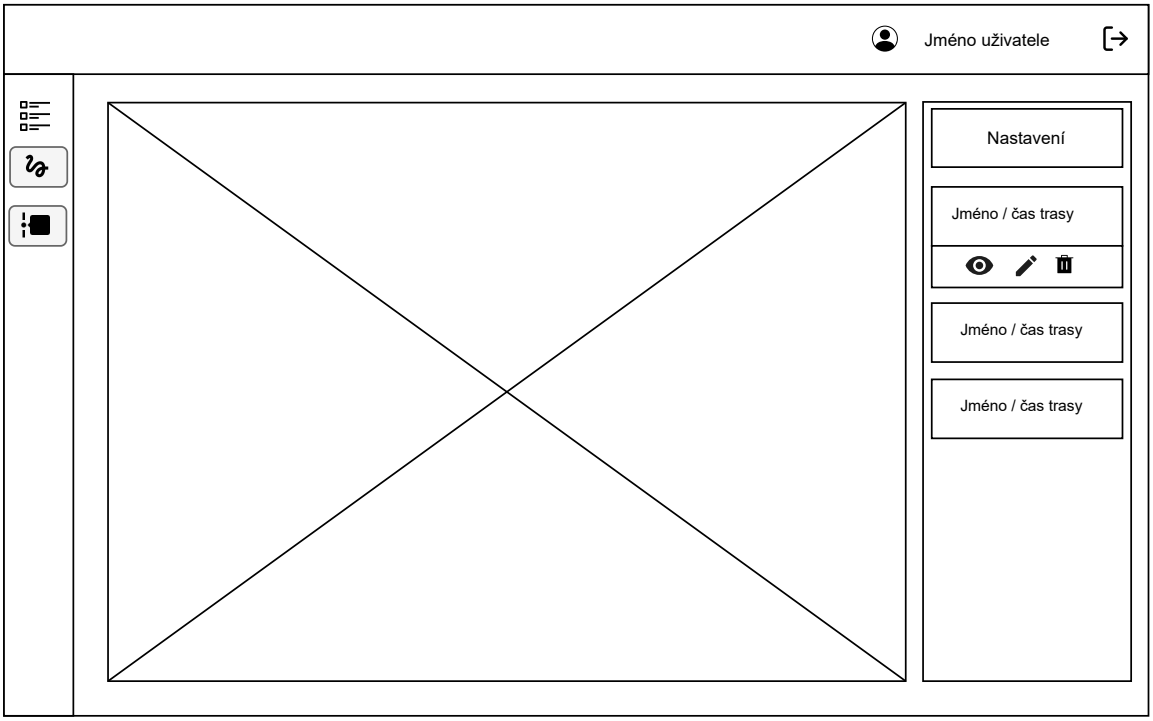
Po úspěšném přihlášení uživatele bude první stránka seznam jeho aktivit. Zde bude mít uživatel možnost kliknout na jednotlivé aktivity, které se následně otevřou a zobrazí uživateli trasu aktivity a dodatečné informace. Každá aktivita také bude mít možnost smazání z aplikace pomocí menšího tlačítka. Ve spodní části stránky se bude vyskytovat stránkování, což uživateli poskytuje možnost jednoduše procházet seznam jeho aktivit. Také si bude moci nastavit, kolik aktivit chce na jedné stránce zobrazit. Návrh této stránky je znázorněný na obrázku 3.4.



Obrázek 3.4: Uživatelské aktivity

Zabrané území a zobrazení filtrace tras

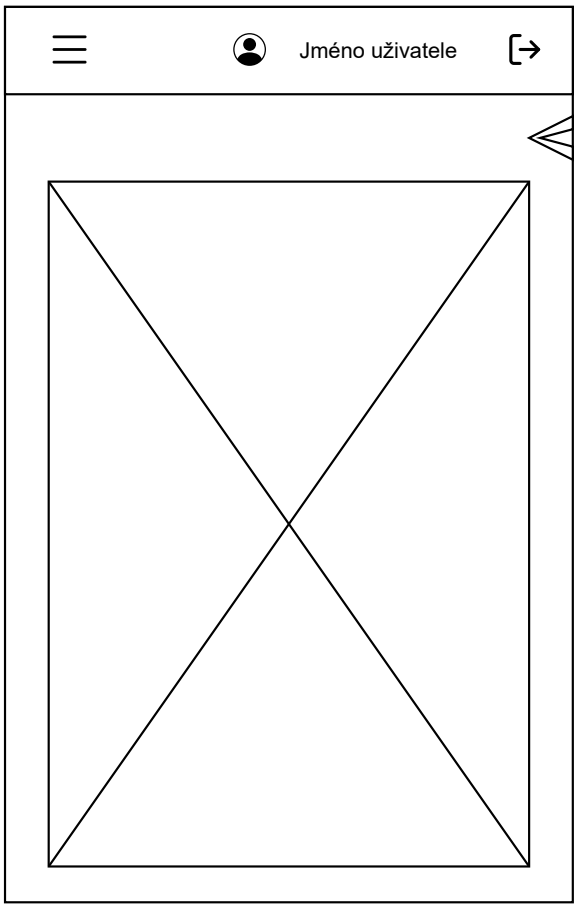
Na stránkách *Filtrované trasy* a *Zabrané území* si bude moci uživatel prohlédnout jeho zpracované data. Na stránce se musí vyskytovat komponent mapy, která uživateli zobrazí jeho data. Součástí stránky bude legenda, která od sebe odliší jednotlivé aktivity. Pro uživatelské pohodlí je vhodné implementovat různé způsoby přizpůsobení. Je vhodné barevně odlišit od sebe jednotlivé trasy a viditelně zobrazit zabrané území. Přizpůsobením se může rozumět například změna barev jednotlivých tras, změna tloušťky hranic apod. Data v legendě a na hlavní mapě se budou lišit dle stránky, ke které uživatel přistoupí. Návrh stránky je vidět v obrázku 3.5.



Obrázek 3.5: Souhrn tras a zabrané území

Responzivita aplikace

Drtivá většina moderních webových aplikací musí počítat s přístupem uživatelů pomocí mobilního telefonu. Je potřeba aby se aplikace přizpůsobila, pokud je šířka prohlížeče omezená. Obrázek 3.6 popisuje například změnu navigačních menu do tzv. *hamburgeru* a dalšího elementu, který zobrazí uživatelské menu přizpůsobení zmíněné na minulé stránce.



Obrázek 3.6: Zobrazení pro mobilní zařízení

Kapitola 4

Implementace řešení

4.1 Příprava serveru

Sekce popisuje části, které byly nutné připravit a implementovat v první fázi.

4.1.1 Konfigurační soubor

K bezpečnému uložení konfigurace je použit soubor *appsettings.json*. Díky uložení většiny důležitých nastavení v jednom souboru je lze v budoucnu jednodušeji změnit a není potřeba zasahovat přímo do zdrojového kódu aplikace. To se hodí například při změně adresy databáze apod.

Soubor je načten v C# pomocí metody `builder.Configuration.AddJsonFile()` a následně se k němu přistupuje pomocí DI (angl. *Dependency Injection*), které je vysvětlené v sekci 4.1.3. Následně je instance definovaná jako třída *IConfiguration*. Příklad struktury v souboru appsettings.json:

```
{
    "Strava": {
        "clientid": "72572",
    }
}
```

V koncovém bodu se následně přistupuje k objektům pomocí jeho cesty:

```
Iconfiguration config;
config["Strava:clientid"]
```

Tímto způsobem jsou do souboru uloženy adresy pro připojení k databázi, k přístupu do Stravy nebo adresa frontendu.

4.1.2 Vytvoření databáze

Databáze je vytvořena pomocí tzv. *Code First* přístupu. To znamená, že jsou vytvořeny objekty a třídy a podle nich vytvořené tabulky v databázi. Za použití C# byl využit pro tento proces EF (angl. *Entity Framework*). EF poskytuje mapování relací objektů (angl. Object Relational Mapping) v platformě .NET díky vytvoření konceptuálního modelu aplikační domény. Tento model popisuje vytvořené třídy a objekty, které následně využívá pro vytvoření dotazů do databáze. V minulosti EF fungoval na bázi vytvoření konceptuálního

modelu pomocí reverzního inženýrství již existující databáze do souboru XML. Pro tento typ souboru bylo vytvořeno prostředí, ve kterém šlo model dále upravovat. Od doby vytvoření EF si vývojáři žádali o možnost psát pouze kód a ve verzi .NET 4.1 byl představen koncept *Code First*. Při psaní *Code First* je nutné si pouze definovat objekty a třídy, které nejsou nutně specifické k EF. Jedná se o klasické C# třídy. EF dokáže rozpoznat větší množství informací o vaší zamýšlené databázi dle různých definic využitých v těchto třídách. EF poskytuje další možnosti manuální konfigurace, což přijde vhod, pokud automatické generování nefunguje dle představ vývojáře, např. špatně mapované relace nebo typy sloupce. [29, 48] Po definování základních tříd a objektů, které tvoří databázi je také potřeba definovat jednotlivé tabulky pro *Entity Framework*. To se děje v souboru *AppDbContext*, tzv. *Properties* definované v této funkci poskytují přímý přístup z aplikační vrstvy do databáze. Také lze zde pomocí metody *OnModelCreating* definovat speciální vlastnosti relací a tabulek, jako je například chování po smazání řádku v tabulce. Jako příklad lze vzít třídu *User*, ta je následně díky *Entity Frameworku* přeložena a v databázi je vytvořena tabulka *Users*. Útržek kódu definice třídy *User* v C#:

```
[Key]
[DatabaseGenerated(DatabaseGeneratedOption.None)]
[Required]
public long Id { get; set; }
public string Name { get; set; } = null!;
[StringLength(50)] public string Username { get; set; } = null!;
[StringLength(300)]
public string? AccessToken { get; set; }
[StringLength(300)]
public string? RefreshToken { get; set; }
public DateTime? AccessExpiresDate { get; set; }
public DateTime LastUpdated { get; set; }
public virtual List<UserActivity> Activities { get; set; } = new();
```

Po zapsání definice entit se pomocí konzolového řádku spustí EF, který vygeneruje tzv. *migraci*. Migrace popisují změny, které proběhly od poslední změny databázového modelu. EF porovnává současný model oproti snímku staršího modelu pro výpočet rozdílných objektů nebo entit. [34]

Migrace se jménem PrvniMigrace by se přidala pomocí příkazu:

```
dotnet ef migrations add PrvniMigrace
```

Následně pro aplikování změn do databáze se používá příkaz:

```
dotnet ef database update
```

4.1.3 Implementace API

Základ API je definován v souboru *Program.cs*, pomocí syntaxe *Minimal API*. To je nový framework uvnitř frameworku *.NET 6* a vyšší, který si dává za cíl minimalizaci množství psaného kódu. C# Minimal Api využívá *Dependency Injection* u všech jeho koncových bodů. Funkce DI je poskytnout metodě všechny relevantní objekty, které potřebuje. Metoda samotná nehledá instance objektů, ale naopak jsou ji předány jako parametry. V implementaci *Minimal API* je třída *builder.Services* je prostředník pro DI a následně jsou do ní vložené nutné služby.

Jednoduché vysvětlení poskytuje otázka z knížky Dependency Injection Principles, Practices, and Patterns – „Jak vysvětlit Dependency Injection 5letému dítěti?“ [46]:

> „ Pokud si zkusíš vzít jídlo z ledničky pro sebe, je možné, že nastanou problémy. Můžeš nechat otevřené dveře ledničky, můžeš si vzít něco, co by tvoje mamka a taťka nechtěli, aby jsi měl. Nebo také můžeš v ledničce hledat něco, co tam vůbec není nebo už je prošlé. Místo hledání je lepší říci: „Potřebuji nějaké pití k obědu“ a poté se rodiče postarají o to, aby jsi měl něco k pití. “

Do DI je přidáno několik dalších služeb, např. *Hangfire*, který je dále probraný v sekci 4.5. Za zmíňku stojí nastavení CORS (angl. *Cross-Origin Resource Sharing*). CORS dovoluje prohlížeči klienta vytvářet a posílat požadavky na API mezi různými doménami. Toto se může hodit v případě, že API je hostované na jiném serveru a případně doméně než samotný server pro frontend. Pokud by tento případ nastal, požadavky by byly automaticky zablokovány prohlížeči a uživatelé by nemohli komunikovat s API. Zde se objevuje koncept CORS, který tuto komunikaci povoluje. [26]

Webová aplikace byla po dobu testování a vývoje spouštěna na lokálním serveru na portu 3000, zatímco API běželo na portu 7040. Už toto nastavení jiných portů vyžaduje povolení CORS. Pro přidání CORS do služeb v C# *Minimal API* se využívá metoda `builder.Services.AddCors()`. Příklad konfigurace pro povolení CORS na lokálním serveru:

```
builder.Services.AddCors(policyBuilder =>
    policyBuilder.AddPolicy("AllowLocalhost", x=>
        x.AllowAnyMethod()
         .AllowAnyHeader()
         .AllowCredentials()
         .SetIsOriginAllowed(origin =>
            new Uri(origin).Host == "localhost")
         ));
```

Pomocí *builder.Services.AddAuthentication()* se zde také nastavuje využití souborů cookie pro autentizaci, dále probrané v sekci 4.4.

4.1.4 Koncové body

Koncové body představují základ komunikace mezi API a klientem. Každý koncový bod definuje svoji URL cestu, ke které klient přistupuje pomocí tzv. HTTP (angl. *Hypertext Transfer Protocol*) požadavků. Ty jsou dále vysvětleny v sekci 4.6.3. *Minimal API* umožňuje mapovat funkce přímo na koncový bod. Po přijetí požadavku se API podívá zdali existuje koncový bod s definovanou URL cestou a pokud ano, tak se spustí namapovaná funkce. Tento způsob plně podporuje *Dependency Injection*, předávání parametrů pomocí URL parametrů nebo také jako součástí těla požadavku. Při definici koncového bodu lze také definovat zdali uživatel musí být přihlášen nebo mít specifickou roli v systému. Pokud klient nesplňuje dané přístupové povolení, tak se automaticky odepře přístup.
Příklad mapování koncového bodu:

```
app.MapPost("/api/register", AuthRepository.Register)
    .AllowAnonymous();
```

A následně *Minimal API* využije *Dependency Injection* pro předání následujících parametrů:

```
public static async Task<IResult> Register
    (AppDbContext dbContext, HttpClient httpClient,
    HttpContext context, IConfiguration config,
    IBackgroundJobClient backgroundJobs,
    [FromQuery] string state, [FromQuery] string code,
    [FromQuery] string scope)
{...}
```

Obdobným způsobem jsou definovány všechny koncové body v API, které klient potřebuje.

4.2 Naplnění statických dat

V sekci je popsána příprava a způsob získání dat nutných k funkci aplikace v pozdější fázi.

4.2.1 Overpass pro OSM

Overpass je API, které dovolí svým klientům přistupovat k datům uložených v databázi OSM pomocí speciálních výrazů, tzv. *query*. Existuje množství oficiálních serverů, které jsou veřejně přístupné. Servery lze využít v případě malé režie, tj. jsou omezeny zhruba na 10000 dotazů denně. [39]

Pro potřeby práce byl vytvořen virtuální stroj v portálu *Azure*. V době vytváření toho virtuálního stroje bylo pro licenci *Azure for Students* dostupný pouze region v Austrálii. To značně omezuje rychlost aplikace, v kombinaci s levnější volbou procesorových jader mají veškeré požadavky následně okolo 500 milisekund zpoždění při spuštění lokální verze aplikace z *Dockeru*. Úplný návod pro spuštění aplikace pomocí *Dockeru* se nachází v příloze A. [3].

Pro instalaci *Overpass* API na virtuální stroj bylo postupováno podle oficiálního návodu pro vytvoření vlastní instance serveru[1]. V rámci instalace API byl zprovozněn *Apache* server, který zprostředkuje přístup k databázi OSM. [9]

Příprava OSM dat

K naplnění databáze byl využit soubor OSM dat pro Českou Republiku dostupný ze serverů VUT FIT[2]. Ze souboru byly následně extrahovány data pomocí skriptu *Osmosis*[3]. Na výstupu jsou data, která zobrazují pouze různé cesty v Brně a okolí.

Soubor je ořezaný o informace infrastruktur a dalších nepodstatných prvků pomocí příkazu:

```
./osmosis --read-pbf czech_republic-2021-10-23.osm.pbf
          --tf accept-ways highway=*
          --tf reject-ways highway=sidewalk,crossing
          --tf reject-relations
          --bounding-box
            top=49.25589126316103 left=16.475088825520327
            bottom=49.1459949173186 right=16.7047932397780
          --used-node --write-xml brno.osm
```

[1] Více informací na: https://wiki.openstreetmap.org/wiki/Overpass_API/Installation.
[2] OSM data pro ČR: https://osm.fit.vutbr.cz/extracts/czech_republic/.
[3] Osmosis dostupné z: https://wiki.openstreetmap.org/wiki/Osmosis.

4.2.2 Příprava databáze serveru

Myšlenka přístupu k regionům je tvořena na principu uložení jednotlivých středů regionů do databáze společně s body, které ohraničují území a tvoří Voroného region. Zdroj dat je soubor z předešlého odstavce, následně jsou jeho data dále extrahována pomocí python skriptu `extract_nodes_from_osm.py`. Soubor zkopíruje jméno vstupního souboru a přidá mu příponu `_nodes`. Na výstupu jsou čisté souřadnice všech středů regionů, jako jsou např. silnice, ulice a cesty. Stejný princip by bylo možné využít i globálně, aby aplikace nebyla omezena pouze na oblast Brna. Největší problém jsou zde potom hardwarová omezení, jako je např. kapacita úložiště v portálu Azure a rychlost prohledávání SQL databáze.

Další krok spočívá ve vypočítání hranic Voroného regionů. Díky skriptu `run_voronoi.py` byly všechny body ze souboru `*_nodes.txt` podrobeny funkci *Voronoi* z knihovny *Scipy*, zmiňované v sekci 2.4. Na výstupu je vytvořen soubor `*_nodes_regions.txt`, který obsahuje jako první dvě položky zeměpisné souřadnice středu regionu a následně řetězec znaků jako pole hraničních souřadnic Voroného regionu. Dělitel sloupců byl nastavený jako znak „;“. Počet Voroného regionů pouze pro Česko přesáhl 9,5 milionu. To znamená, že i do databáze samotné by bylo následně vloženo 9,5 milionu řádků. Z důvodu implementace databáze na online placeném portálu *Azure* byl tento soubor dále zmenšen pro potřeby bakalářské práce pouze na oblast Brna a okolí. To je ve výsledku necelých 200 tisíc řádků. Značně to ulevilo rychlosti celé aplikace při zpracování aktivit, přičemž další oblasti by nebyl problém v budoucnu vložit. Samotný proces vložení byl proveden pomocí nástroje *Microsoft SQL Server Management Studio*[4].

Příklad jednoho řádku z exportovaného souboru `*_nodes_regions.txt`:

```
16.5001211;49.1761032;'[16.49919818031525,49.174517706363304],
[16.500758940325838,49.17438858468905],
[16.500975032570768,49.17589229164807],
[16.49952847265777,49.17635098941457],
[16.499449708003834,49.176045363691514],
[16.49919818031525,49.174517706363304]'
```

Vložení regionů do databáze

Po navázání připojení s databází v aplikaci *SQL Server Management Studio* je potřeba otevřít konkrétní model databáze, který obsahuje vytvořené tabulky. Pravým tlačítkem se otevře kontextové menu a pod tlačítkem *Tasks* se nachází možnost *Import Data*. Toto lze vidět na obrázku C.1. Po otevření dialogu je nutno zvolit typ zdroje. V tomto případě se jedná o *Flat File Source*. Jelikož soubor *VoronoiRegion.src* neobsahuje názvy sloupců, je potřeba relevantní zaškrtávací políčko odškrtnout. Následně je nutno v liště správně navolit oddělovací znaky. Sloupce jsou rozděleny pomocí znaku „;“ a řádky pomocí „{LF}“. Pod lištou *Advanced* je doporučeno nastavit typ zdrojové proměnné. V případě bakalářské práce se jedná o typ *double-precision float [DT_R8]* pro první dva sloupce a *text_stream [DT_TEXT]* pro třetí sloupec. V *Preview* lze jednoduše zkontrolovat, zda rozdělení dat na sloupce a řádky funguje dle očekávání. Správně rozdělení do sloupců je znázorněno na obrázku C.2. V poslední řadě je potřeba sloupce správně namapovat. Jelikož je databáze vytvořena pomocí *Code First* přístupu, tak se vkládají data již do existující tabulky *StaticRegions*. Obrázky zobrazující vložení dat jsou k nalezení v příloze C.

[4]SQL Server Management studio dostupné z: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

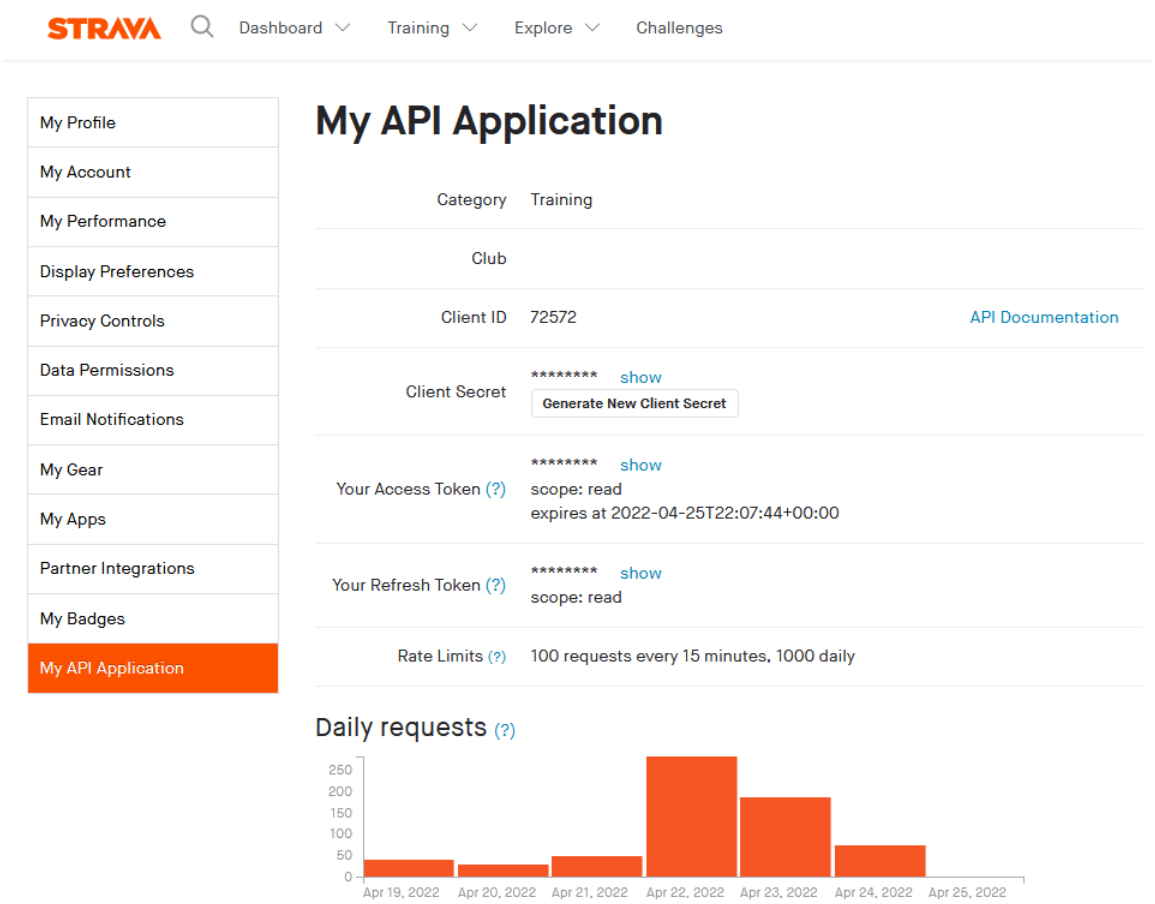
4.3 Registrování API pro Stravu

Každý registrovaný uživatel má možnost si ve Stravě zaregistrovat vlastní aplikaci pro přístup k Strava API. To je dostupné na stránce `https://www.strava.com/settings/api`, v kategorii „My API Application“. Tuto stránku lze vidět na obrázku 4.1.

Díky tomuto procesu je umožněn přístup k uživatelským datům z databáze Stravy. Každá aplikace pak má přidělené jednoznačné identifikační prvky:

- Client Id.
- Client Secret.

Tyto parametry Strava využívá pro autentizaci uživatelské aplikace do Stravy. Díky nim dokáže Strava přesně říci, jak velkou režii mají jednotlivé aplikace a k jakým datum přistupují.



Obrázek 4.1: Sekce My Api Application na stránce Strava

4.4 Propojení uživatele, aplikace a Stravy

V sekci je detailně probrané téma implementace navázání spojení a přístupu k uživatelským datům.

4.4.1 Propojení z pohledu uživatele

Po kliknutí na tlačítko „Přihlásit“ je uživatel přesměrován na stránku Stravy, kde se přihlásí do svého účtu. V případě, že je již přihlášen, je mu zobrazena stránka s potvrzením povolení přístupu. Výběr k jednotlivým oprávněním se uživateli zobrazí ve formě zaškrtávacích políček. Doporučené povolení je čtení všech aktivit, aplikace nežádá žádný zápis do Stravy.

Po udělení oprávnění mapové aplikaci je uživatel přesměrován na seznam aktivit. Pokud se uživatel přihlásil do aplikace poprvé, tak se na straně API začnou zpracovávat uživatelovi aktivity, viz sekce 4.5. Diagram komunikace fitness aplikace s API aplikace Stravy je zobrazený na obrázku 3.3.

4.4.2 Propojení z pohledu API

Po přijetí požadavku ze strany webové aplikace je vygenerována adresa URL. Jako parametry jsou dále přidány identifikační prvky pro Stravu[5]:

- URL Stravy.
- Client ID.
- URL pro přesměrování (angl. *Redirect_uri*).
- Požadované povolení (angl. *Scope*).

Následná URL je předána zpátky klientovi jako odpověď a klient se přesměruje. Šablona URL jako příklad pro komunikaci:

```
https://www.strava.com/oauth/authorize?client_id=72572
&response_type=code
&redirect_uri=https://localhost:3000?register=/api/register
&scope=profile:read_all,activity:read_all
```

Jakmile uživatel autorizuje aplikaci přístup na stránce Stravy, tak je klient přesměrován na dříve zadanou URL. Po autorizaci Strava dále přidává do URL několik parametrů:

- *Code* – jednorázový kód pro získání obnovovacího a přístupového tokenu.
- *Scope* – přístup udělený uživatelem.

Klient zpracuje tyto parametry z URL a dále je předá do API pomocí koncového bodu. Zde je následně použít parametr *Code* pro získání obnovovacího a přístupového tokenu ze Stravy. Toto chování požadavků a odpovědí je součástí protokolu *OAuth 2.0*[6].

[5] Dokumentace pro autentizaci do Stravy: https://developers.strava.com/docs/authentication/#tokenexchange

[6] OAuth online dokumentace: https://oauth.net/2/

OAuth 2.0

OAuth je standardní protokol pro autorizaci. Cílem protokolu je jednoduchý přístup k autorizaci při zachování integrity a bezpečnosti uživatelských dat. Díky tomuto protokolu je možné přistupovat k uživatelským datům bez nutnosti předávání citlivých uživatelských dat, jako jsou přihlašovací jméno a heslo. Přístupové tokeny mají standardně dobu platnosti. Do doby vypršení lze za pomocí těchto tokenů posílat požadavky na API – v případě fitness aplikace se jedná o API aplikace Strava. Díky těmto přístupovým tokenům Strava dokáže rozpoznat, zdali uživatel doopravdy udělil požadované aplikaci přístup ke svým datům a následně vygeneruje odpověď. V případě, že vyprší doba platnosti přístupového tokenu, tak API posílá požadavek pro nový přístupový token za pomocí obnovovacího tokenu. Tento protokol také umožňuje uživateli lehce zrušit přístup ke svému účtu cizí aplikaci. To může přijít v hod v případě, že uživatel nemá potřebu aplikaci nadále využívat nebo dojde k narušení bezpečnosti jeho účtu. [10]

Na množství stránek lze vidět tlačítka „Přihlásit pomocí Google“ nebo „Přihlásit pomocí Facebooku“. Tyto tlačítka v pozadí fungují na principu *OAuth 2.0*. Pro přihlášení do mapové aplikace je tento protokol také využit. Po udělení přístupu aplikace k uživatelským datům je vytvořena kopie uživatelského účtu v databázi aplikace, také je k účtu uložen přístupový a obnovovací token. V poslední řadě se začnou stahovat uživatelské aktivity ze Stravy. Proces zpracování je dále objasněn v sekci 4.5.

Opakované přihlášení

Pro vyvrácení nutnosti přihlášení uživatele při každém navštívení stránky byla řešena otázka, zdali využívat soubory cookie (angl. *Cookie-based Authentication*) nebo JWT[7] (angl. *JSON Web Token*). [21]

JSON Web Token je standardizovaný způsob přenosu informací mezi dvěma objekty během autorizace. Jakmile je uživatel přihlášen, každý následující požadavek v sobě obsahuje JWT, na což reaguje API a zpřístupní uživateli koncové body. Věrohodnost je doložena elektronickým podpisem. Mohou být podepsány pomocí algoritmu využívající tzv. *hashování*. Při hashování dat pomocí dané funkce je vždy vytvořený stejný řetězec znaků. Z výsledného řetězce nelze zpětně zjistit vstupní data. Další možnost pro zajištění věrohodnosti je využití veřejného a soukromého klíče. Z obecného hlediska je JWT modernější způsob, který je složitější na implementaci, ovšem bezpečnější a má lepší škálovatelnost. JWT se skládá z hlavičky, dat a podpisu. Hlavička se většinou skládá ze dvou částí – první popisující typ tokenu a druhá část si ukládá informaci o použitém podepisovacím algoritmu. V datové části se vyskytují nároky (angl. *Claims*), díky nim si klient dokáže i uložit role uživatele systému, případně i další informace. Na výstupu je řetězec znaků oddělený tečkami, který je zašifrovaný pomocí tajného klíče. [21, 6]

HTTP Cookies jsou základním kamenem moderního internetu. Jsou to malé textové soubory uložené na straně uživatele, které obsahují data. Při využívání souborů cookie je jedna z největších obav CSRF (angl. *Cross-site Request Forgery*) útok. Funkce prohlížeče při otevření webové stránky je poslání všech relevantních cookies právě pro danou stránku. Tento útok pracuje na bázi přesměrování uživatele a využití tohoto chování. Při úspěšném útoku jsou následně vytvořeny zfalšované požadavky na API pomocí cizí identity. [25, 55]

CSRF útok by ovšem pro fitness aplikaci neměl velký význam. Aplikace nemá práva zasáhnout do dat, které jsou uloženy v databázi Strava. V podstatě největší škodu, kterou

[7] JWT online dokumentace: https://jwt.io/

útočník může napáchat je smazání učtu v mapové aplikaci. Pro potřeby aplikace byla tedy zvolena technologie souborů cookie. Po přihlášení klienta je do prohlížeče následně vložena tzv. *Session Cookie* obsahující identifikační řetězec. Tato cookie nemá nastavenou expirační dobu, co má za následek, že aplikace si pamatuje přihlášeného uživatele, dokud se uživatel manuálně neodhlásí.

4.4.3 Načtení dat

Po získání přístupového a obnovovacího tokenu API začne načítat uživatelské informace. Strava nazývá uživatelské trasy jako aktivity. Po vyslání požadavku dostává API jako odpověď jednoduchý seznam uživatelských aktivit[8]. Tento seznam a základní informace o uživateli jsou ihned předány zpět klientovi. API dále prochází jednotlivé aktivity a vytváří požadavky na tok aktivity (angl. *Activity Stream*)[9]. Jednotlivým požadavkům je přidán parametr *Keys* s hodnotou *Latlong*. To způsobí, že odpověď na požadavek je pouze pole souřadnic uživatelské aktivity. Následně jsou požadavky asynchronně poslány. Odpověď je uložena do databáze aplikace a začne zpracování dat.

4.5 Zpracování dat

První část sekce obsahuje obecné informace o zpracování dat v pozadí API pomocí knihovny *Hangfire*[10]. Následně je popsána tématika filtrování pomocí společné navštívené cesty. Poslední část řeší problematiku využití Voroného diagramu k zabírání území uživatelem.

4.5.1 Zpracování v pozadí

Při vytváření aplikace nastala otázka škálování. Zpracování dat je velice náročná procesorová operace, která zabere větší množství času. To má za následek vysoké využití serveru a následné omezení jeho funkčnosti. Otázka nastala, protože když byl server vytížený tak API server odpovídal na požadavky s větším zpožděním. Z dříve zmíněných důvodů byla využita služba *Hangfire*.

Hangfire

Hangfire je široce dostupná služba. Lze ji používat v .NET projektech, ASP.NET webových aplikacích, ale i webových aplikacích jiného typu. Dále podporuje konzolové aplikace nebo i služby Windows. Služba Hangfire vytváří úlohy běžící v pozadí pomocí vláken. Úlohy mají několik různých typů:

- Jednorázové (angl. *Fire-and-forget*).
- Opakované (angl. *Recurring*).
- Opožděné (angl. *Delayed*).

[8] Strava API dokumentace seznam aktivit: https://developers.strava.com/docs/reference/#api-Activities-getActivityById

[9] Strava API dokumentace toku aktivity: https://developers.strava.com/docs/reference/#api-Streams-getActivityStreams

[10] Dokumentace Hangfire: https://docs.hangfire.io/en/latest/

Práce v pozadí může být nastavena na opakované volání metod v rozmezí vteřin, hodin až dní. Není potřeba vytvářet nové statické nebo instanční metody, Hangfire využívá již implementované metody. Použití Hangfire pomáhá se škálováním a perzistencí aplikace. Pro zachování perzistence Hangfire využívá trvalé uložiště, to umožňuje vytvořeným úlohám přetrvat restarty nebo výpadky serveru. Trvalé uložiště můžou být např. SQL Azure nebo SQL Server. Úlohy jsou zpracovány pomocí Hangfire serveru. Server je implementován jako soubor vláken, která vyzvedávají úlohy z uložiště. Server se také stará o udržování čistého uložiště a odstraňuje zastaralá data. [22]

Implementace Hangfire v C#

Hangfire se do koncových bodů vkládá pomocí *Dependency Injection*. Konkrétně pomocí třídy *IBackgroundJobClient*.
Pro využití Hangfire musí metody být serializovatelné. To je hlavně z důvodu ukládání dat do trvalého uložiště. V praxi to znamená, že parametry metod nesmí být objekty, ale pouze jejich unikátní identifikátory. Pro každý jednotlivý úkol je nutno vytvořit novou instanci. V případě, že by se toto nedělo, docházelo by k problémům s *Garbage Collector* v C#, kdy reference na databázi může být uklizena dříve, než je úkol dokončen. Pro proces instanciace je vytvořena třída *ProcessActivityClass*. Následně jsou v ní implementované metody pro filtrování (průměrování) cest a přiřazení Voroného regionů uživateli. Příklad předávání pomocí identifikátorů:

```
public async Task RunAveraging(long activityId, long userId){...}
```

Zde je místo odkazu na třídu aktivity a uživatele předáno pouze jejich unikátní ID a to je následně vyhledáno v databázi.

4.5.2 Filtrování cest

Nejprve je potřeba zprůměrovat cestu v rámci sebe sama v případě, že v jedné cestě uživatel projde stejnou trasu dva a vícekrát. Z původní cesty se vytvoří kopie, následně je vybrán první bod a je prohledán zbytek této cesty. Po vybrání všech bodů, které jsou vzdáleny méně jako 15 metrů, je provedeno zprůměrování jejich souřadnic a následně je přepsána jejich hodnota v kopii trasy.

Vzdálenost souřadnice

Souřadnice jsou objekty třídy *RouteCoordinate*, která má instanční metodu *GetDistanceTo*. Ta bere jako parametr druhý objekt třídy *RouteCoordinate* a následně vypočítá vzdálenost podle *Haversine formula*[11].

Ukázka implementace *Haversine formule* v C# ze třídy *RouteCoordinate*:

```
public double GetDistanceTo(RouteCoordinate other)
{
    double R = 6371e3;
    double num1 = Latitude * (Math.PI / 180.0);
    double num2 = other.Latitude * (Math.PI / 180.0);
    double d1 = (Latitude-other.Latitude) * (Math.PI / 180.0);
    double d2 = (Longitude-other.Longitude) * (Math.PI / 180.0);
```

[11] Výpočet vzdálenosti dle Haversine formule: https://www.movable-type.co.uk/scripts/latlong.html

```
        double a = Math.Sin(d1 / 2) * Math.Sin(d1 / 2) +
                   Math.Cos(num1) * Math.Cos(num2) *
                   Math.Sin(d2 / 2) * Math.Sin(d2 / 2);
        double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
        return R * c;
    }
```

Optimalizace

V případě průměrování mezi dvěma cestami existuje optimalizace. Třída *RouteCoordinate* má relaci na pole objektů třídy *CoordinateIndexes*. Tyto objekty jsou jednotlivé indexy všech dalších souřadnic v rámci jedné cesty, které byly pod prahem[12] algoritmu průměrování. A tedy jsou to všechny souřadnice, které mají stejnou hodnotu zeměpisné šířky a délky. Z důvodu použití SQL databáze se zde jedná o instanci třídy. V jiných případech použití by tato třída šla nahradit jako pole čísel.

Z důvodu způsobu zobrazení tras není možné lehce odstranit nadbytečné souřadnice. V případě odstranění indexů by cesta měla nelineární průchod a bylo by nutné složitě iterovat skrze pole *CoordinateIndexes* a upravovat všechny zasažené indexy. Náročnost zmíněného prohledávání je příliš vysoká.

Filtrace cesty

V první části se průměruje cesta sama mezi sebou. Pokud se při iteraci objeví skupina souřadnic, která se nachází pod prahem, do objektu první souřadnice se uloží relace na novou instanci třídy *CoordinateIndexes*. Tato instance je naplněna dříve zmíněnou skupinou souřadnic. Všechny souřadnice jsou také přepsány na stejnou zeměpisnou šířku a délku.

Po dokončení průměrování cesty samo sebou začne API postupně průměrovat všechny současné již alespoň jednou zprůměrované trasy. V případě, že je nalezena alespoň jedna skupina souřadnic, která se nachází pod prahem průměrování, tak jsou dotyčné aktivity zařazeny do relace skupiny. Tato relace je v budoucnu použita pro chytré přepočítávání aktivit v případě odebrání (smazání) aktivity z účtu aplikace.

Postup algoritmu průměrování dvou cest je obdobný průměrování jedné cesty. Algoritmus iteruje přes dvě cesty. Pokud se při iteraci skrze druhou cestu některý z bodů nachází pod prahem, je uložen do pomocného pole. Když algoritmus dosáhne konce druhé cesty, podívá se na pomocné pole a vytvoří průměrnou hodnotu ze všech nalezených souřadnic. Zde se také dívá na pole *CoordinateIndexes* všech souřadnic, které byly označeny jako pod prahem. Všechny indexy, které jsou v tomto poli, jsou následně označený jako zkoumané a dále je při iteraci algoritmus ignoruje. Také jsou souřadnice na těchto indexech upraveny dle nové zprůměrované zeměpisné šířky a délky.

Názorná ukázka výstupu filtrace dvou cest je viditelná na obrázcích 4.2 a 4.3. Po dokončení filtrace jsou následně zprůměrované do jedné cesty, viditelné v obrázku 4.4.

[12] Prah = maximální vzdálenost dvou bodů.

Detaily implementace

Pseudokod třídy *RouteCoordinate*:

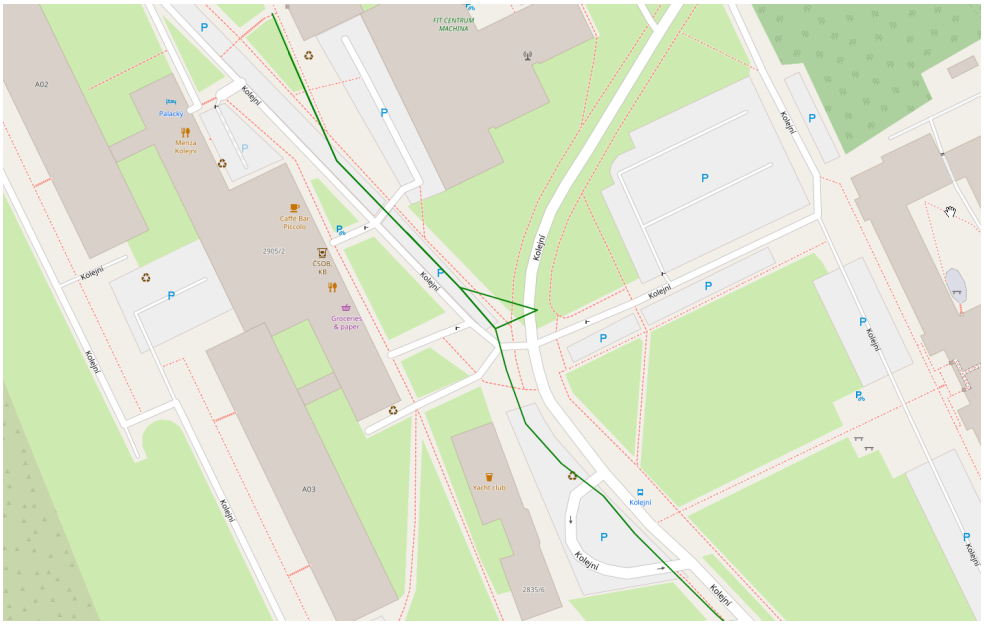
```
double Šířka,
double Délka,
bool Centrovaná,
CoordinateIndexes[] Indexy,
Vzdálenost(RouteCoordinate souřadnice)
```

Pseudokód algoritmu:

```
Na vstupu pole: cestaA, cestaB
Měj pole: IndexyAGlobal, IndexBGlobal
podPrahem = false, Prah = 15
Dokud indexA < délka(cestaA):
    souřadniceA = cestaA[indexA]
    Měj pole: IndexyvIteraci, stejnáŠířka, stejnáDélka

    Dokud indexB < délka(cestaB):
        souřadniceB = cestaB[indexB]
        Pokud indexA v IndexyAGlobal:
            break;
        Pokud indexB není v IndexyBGlobal:
            vzdalenost = souřadniceA.Vzdalenost(souřadniceB)
            Pokud Vzdalenost < Prah:
                podPrahem = true
                Přidej souřadniceB.Šířka do stejnáŠířka
                Přidej souřadniceB.Délka do stejnáDélka
                Přidej souřadniceB.Indexy do IndexyvIteraci
                Přidej indexB do IndexyvIteraci
                Přidej indexB do IndexyBGlobal
        Pokud indexB je délka(cestaB)-1:
            Přidej souřadniceA.Šířka do stejnáŠířka
            Přidej souřadniceA.Délka do stejnáDélka

            Projdi indexy v souřadniceA.Indexy a~přepiš cestaA
            dle stejnáŠířka.průměr() a~stejnáDélka.průměr()

            Projdi indexy v souřadniceB.Indexy a~přepiš cestaB
            dle stejnáŠířka.průměr() a~stejnáDélka.průměr()

            Přidej indexA do IndexyAGlobal
            přidej souřadniceA.Indexy do IndexyAGlobal
        Zvyš indexB
    Zvyš indexA
Vrať podPrahem
```

Obrázek 4.2: Filtrace cesty A

Obrázek 4.3: Filtrace cesty B



Obrázek 4.4: Kombinace cesty A a B

4.5.3 Přiřazení Voroného regionů

Po dokončení filtrace uživatelské aktivity dochází k přiřazení regionů k aktivitě. Pro zlepšení výkonu API a je využita optimalizace v C#, která spočívá v hromadném posílání požadavků. Cesta je rozdělena do skupin po 20 souřadnicích a následně je pro každou vytvořen tzv. *Task*. Následně se čeká na výsledek všech požadavků pomocí metody *Task.WhenAll(tasks)*. Ty jsou na nižší programovací úrovni zpracovány paralelně pomocí vláken. [2]

Úryvek kódu pro zpracování skupiny požadavků:

```
var centeredRoute = new List<RouteCoordinate>();
var numberOfBatches = (int)Math.Ceiling((double)route.Count / batchSize);
for (var i = 0; i < numberOfBatches; i++)
{
    var originalCoords = route.Skip(i * batchSize).Take(batchSize);
    var tasks = originalCoords.Select(x =>
    GetCenteredCoordinate(x.Latitude, x.Longitude);
    centeredRoute.AddRange(await Task.WhenAll(tasks));
}
```

V kódu lze vidět výpočet množství jednotlivých skupin dle nastavené velikosti – proměnná `numberOfBatches`. Následně je z pole vybráno požadované množství prvků a je vytvořené vlákno pro každou souřadnici. Po zpracování požadavků skrze *Overpass API* je výstup přiřazený do pole `centeredRoute`.

Pro každý požadavek je následně zavolaná metoda s následujícím pseudokódem:

```
Na vstupu souřadnice zeměpisné šířky a délky: lat, lon
Opakuj dokud není nalezen alespoň jeden střed regionu:
    Pošli požadavek:
        Nalezni středy regionů a okruh 10 metrů od lat,lon
    Pokud žádné neexistují:
        Zvyš okruh o 20 metrů
Pokud je více jak jeden střed:
    Najdi nejbližší střed k lat, lon
Vrať souřadnice středu
```

GET Požadavek je vytvořen dle následující šablony:

```
{URL}/api/interpreter?
     data=[out:json];
     node(around:{lookDistance},{lat},{lon});
     out geom;
```

Proměnná lookDistance je okruh, ve kterém se hledají středy regionu. Lat a lon jsou počáteční souřadnice z uživatelovy trasy. Proměnná URL se může změnit dle konfigurace serveru, viz sekce 4.1.1.

Po zpracování celé cesty je cesta uložena jako samostatná, tvořená pouze ze středu regionů. Pokud se v základní uživatelské cestě vyskytl více jak jeden bod uvnitř regionu, poté jsou i ve výsledné cestě duplicitní hodnoty. Proto je před vytvořením dotazů je cesta zkontrolována a duplicitní hodnoty jsou odstraněny. Toto podstatně ulehčí množství vyhledávání, které je nutné dělat v databázi.

Pro získání hranic regionu je následně využita databáze popsaná v sekci 4.2.2. Pro každý střed regionu je vytvořen SQL dotaz. Toto je nejvíce procesorově náročná část z důvodu množství regionů, a tedy i řádků v databázi. Při testování byla průměrná doba zpracování jednoho dotazu kolem 500 milisekund. Relace s výsledným regionem je uložena do pole a zapsána do databáze. Je nutno vyhledávat dle desetinných čísel, tedy při vytvoření dotazu je potřeba zohlednit nepřesnost. Následující úryvek kódu popisuje zmíněný SQL dotaz:

```
var region = dbContext.StaticRegions.FirstOrDefault(x =>
    x.Index2 - 0.0000000001 <= coordinate.Latitude &&
    x.Index2 + 0.0000000001 >= coordinate.Latitude &&
    x.Index1 - 0.0000000001 <= coordinate.Longitude &&
    x.Index1 + 0.0000000001 >= coordinate.Longitude);
```

Přesnost je nastavena na 10 desetinných míst a odvíjí se od formátu zeměpisných souřadnic, které často dosahují většího množství desetinných míst. Přesnost samotná nemění náročnost vyhledávání dle testování.

4.6 Zobrazení dat

Zobrazení dat je zprostředkované pomocí webové aplikace. Sekce probírá využité technologie a některé implementační detaily.

4.6.1 Technologie webové aplikace

Aplikace je implementovaná pomocí *Vue*, dříve zmíněno v sekci 3.2, s použitím *Vuetify*.

Implementace rozhraní

Pro implementaci rozhraní byl využit balíček nástrojů *Vuetify*. Ten je postavený na základě dříve zmíněného *Vue.js*. Cílem projektu je poskytnout vývojářům množství moderně vypadajících komponent, které se často na webu opakují. Následně je úkolem vývojáře je kreativně využít pro vytvoření unikátního webu. *Vuetify* je vyvinuto na základě tzv. *Material Design*, který specifikuje, aby každý komponent byl modulární, responzivní a výkonný. V nedávné minulosti bylo do světa vypuštěno *Vue 3* a díky aktivní komunitě je již dostupné i *Vuetify 3*, i když stále v beta verzi. Právě tato beta verze byla využita pro implementaci uživatelského rozhraní. [54]

Změna zobrazení

Pro konfiguraci přístupu uživatele je využitý nástroj *Vue Router*. Zároveň se i stará o změnu zobrazení komponentů dle nastavené cesty v prohlížeči. Manuálně je zde nastavena konfigurace za použití perzistentního úložiště. Když uživatel není přihlášen, tak ho router nenechá přistoupit na žádné jiné stránky, kromě přihlašovací. A naopak, pokud je uživatel již přihlášen, tak není schopný se vrátit na přihlašovací obrazovku, aby nebyl vytvořený případný konflikt. Útržek kódu z nastavení *Vue Router*:

```
{
  path: "/auth",
  name: "auth",
  meta: {
    requiresGuest: true,
  },
  component: () => import("@/views/AuthView.vue"),
}
```

Následně router poskytuje metodu *beforeEach*, která je zavolaná před každou změnou stránky. Zde je implementovaná konfigurace kontroly meta informací pro danou stránku v porovnání s přihlašovacím stavem uživatelského prohlížeče z perzistentního úložiště.

Perzistentní uložení

Při implementaci aplikace byla řešena otázka dočasného uložení malých uživatelských dat. Jedná se o věci jako uživatelské jméno a v pozdějších fázi uložení přizpůsobení viditelnosti tras, uložení pozice a přiblížení mapy.

Pro zajištění tohoto chování bylo implementované perzistentní úložiště pomocí nástroje *Pinia*. V minulosti *framework Vue* využíval známý a rozšířený nástroj *Vuex*. Počátkem vývoje byl *Pinia* experiment pro prozkoumání budoucích metod pro uložiště, co mohlo být *Vuex 5*. Vývojáři ovšem zjistili, že *Pinia* již implementuje větší množství základních konceptů, které by očekávali ve *Vuex 5*, a proto se *Pinia* stala hlavním doporučeným nástrojem pro perzistentní uložení. *Pinia* má jednodušší rozhraní a obecně jednodušší přístup k metodám, které poskytuje vývojáři. [36]

V úložišti jsou například uchovány informace:

- Stav přihlášení.
- Jméno uživatele.

- Pozice a přiblížení mapy.
- Přiřazené barvy k aktivitě a k regionu.

V úložišti se také vyskytují proměnné, které si klient ukládá pro jednodušší komunikaci mezi jednotlivými prvky stránky. Například, chybové hlášení při selhání požadavku na API je reaktivně nastaveno na proměnnou, která je uložena v hlavním úložišti. Nejvíce uživatel ocení perzistentní uložení, když prochází mezi jednotlivými režimy zobrazení a aplikace udržuje stav, který si uživatel nastavil v předchozích krocích.

4.6.2 Komponent mapy

Jako komponent pro zobrazení dat byl vybrán nástroj na bázi *OpenLayers*. Hlavním důvodem byla jednoduchá integrace s *Vue frameworkem* pomocí podpurné knihovny *vue3-openlayers*[13]. Ta poskytuje komponent mapy kompatibilní s reaktivními vlastnostmi *vue3*. Mapa následně může zobrazovat data na množství rastrových a vektorových vrstvách. [4]

4.6.3 Komunikace s API

Komunikace s API je zajištěna pomocí HTTP požadavků. Na nízké úrovni je syntaxe každého požadavku tvořena z definice typu metody požadavku, hlaviček HTTP a následně dat. Nejvíce používané metody požadavku jsou tzv. *GET, PUT, POST, DELETE* metody. Dále existují i metody *HEAD, TRACE, CONNECT*, ty ovšem nebyly pro implementaci komunikace využity. *GET* metoda je nejspíše nejpopulárnější metoda ze všech, je to typ metody, který prohlížeč využívá pro stáhnutí obsahu webové stránky po kliknutí na jakékoliv URL. Metoda *GET* se také používá v HTML formulářích, kdy při definice formuláře je možné specifikovat použitou metodu. Zde se ovšem vyskytuje bezpečnostní riziko. V požadavku typu *GET* se obecně nevyužívá tělo pro předání dat, ale parametry adresy jako tzv. *query*. V aplikaci je *GET* metoda použita hlavně na požadavky již zpracovaných dat, tedy pro získání dat uživatele. Implementaci metody *POST* v minulosti lze považovat za jeden z největších pokroků webové komunikace. Umožňuje webovým stránkám být doopravdy interaktivní. V aplikaci je využita pro sdělení serveru, aby provedl určitou akci. Jmenovitě dává API sdělení, že si uživatel přeje např. načíst nové aktivity ze Stravy, přihlásit a odhlásit se. *PUT* není tak obvyklá metoda jako *GET* nebo *POST*, využívá se nejčastěji k poslání serveru větší množství dat. V rámci aplikace nebylo nutné ji nikde využít. *DELETE* má opačné využití oproti *PUT*, je využit pro sdělení serveru pro odstranění určitého obsahu. V rámci aplikace se zde jedná hlavně o smazání jednotlivých aktivit nebo smazání svého uživatelského účtu. [47]

Pro zprostředkování požadavků byla v aplikaci využita knihovna *Axios*. Je to JavaScriptová knihovna s podporou prohlížečů a *Node* (základní *framwork* pro *Vue, Vuetify*), využívá i koncept *slibů* (angl. *Promises*). *Promises* nám v aplikaci umožňují asynchronní zpracování požadavků, dá se říci, že určité data jsou „slíbeny“ a následně aplikace může počkat na odpověď API i při využití asynchronních požadavků pomocí známých příkazů *await* a *async*. V aplikaci *Axios* usnadňuje posílání jednotlivých požadavků díky globální konfiguraci. Zde se dají jednoduše nastavit nutné parametry, jako je URL API, typ obsahu a umožnění komunikace mezi webovými doménami – CORS, dále rozebrané v sekci 4.1.3. [12]

[13] Github knihovny: https://github.com/MelihAltintas/vue3-openlayers

Následuje útržek kódu konfigurace *Axios*, je v něm vidět nastavení adresy serverového API, kde se zpracovávají požadavky aplikace. Je nutné také přikládat hlavičky a povolit CORS pomocí *crossDomain* atributu.

```
const config = {
    baseURL: "http://localhost:7040/",
    headers: {
    "Content-type": "application/json",
    },
    withCredentials: true,
    crossDomain: true,
};
```

Dále je volání jednotlivých požadavků implementováno v souboru *UserService.ts*, což je třída zapsaná pomocí *TypeScript*.

Zpracování odpovědí

V API jsou data po komunikaci s databází uložena do C# tříd. Následně v reakci na požadavek na daný koncový bod je *Entity Framework* serializuje do podoby JSON (angl. *JavaScript Object Notation*) a jsou odeslány jako odpověď. Pro jednoduché zpracování na straně klienta jsou obdobným způsobem vytvořeny třídy v *TypeScript*. *TypeScript* je nadstavba *JavaScriptu*, přibližuje svět programování viditelnou část webové stránky blíže k programování *.NET* pomocí objektově orientovaného programování. Za cíl si uložil tvorbu svého vlastního standardu pro tvorbu moderních webových aplikací, k dosažení tohoto cíle pomáhá fakt, že *Typescript* je otevřený software, s otevřeným zdrojovým kódem a jeho vývojáři naslouchají uživatelům při dalším vývoji. [30]

Následující útržek kódu ze souboru *activity.ts* třídy Activity popisuje základní informace o aktivitě, které *frontend* očekává v reakci na požadavek:

```
export default class Activity {
    public name: string;
    public id: number;
    public elapsedTime: number;
    public startDate: string;

    constructor(name: string, id: number,
    elapsedTime: number, startDate: string) {
        this.name = name;
        this.id = id;
        this.elapsedTime = elapsedTime;
        this.startDate = new Date(startDate).toLocaleDateString();
    }
}
```

4.7 Funkce webu pro uživatele

Pro uživatelský komfort bylo implementováno několik funkcí. Jedná se například o komponentu zobrazující načítání zpracování aktivit. Když uživatel přistoupí na stránky, kde se obvykle zobrazují Voroného regiony nebo zprůměrované aktivity, ale server stále nedokončil zpracování těchto aktivit, uživateli se zobrazí komponent kolečka s číselným zobrazením současného stavu načtení v %. Toto je implementované pomocí opakovaných požadavků ze strany klienta na API, následně server jako odpověď vrátí stav zpracování, který je uložen do proměnné. Ta je reaktivně propojena s komponentem načítání. Přestože není dokončeno zpracování, tak si uživatel může zobrazit jednotlivé aktivity v základní podobě načtené ze Stravy. Pokud má uživatel nahráno větší množství aktivit, je zobrazen komponent stránkování. Zde si uživatel může zvolit, kolik aktivit chce vidět na jedné stránce a procházet jednotlivými stránkami. Po kliknutí na aktivitu se aktivita rozevře a uživatelovi se klientské zobrazení automaticky upraví tak, aby mapa aktivity byla celá viditelná na jeho obrazovce. Dále jsou zde k přečtení zajímavé detaily o aktivitě, které jsou dostupné ze Stravy. Po dokončení zpracování aktivit se uživateli umožní možnost smazat jednotlivé aktivity. Tyto tlačítka jsou vypnuté během zpracování z důvodu práce v pozadí na serveru a následně složitému zastavování práce, která již probíhá v *Hangfire*.

Při zobrazení regionů nebo filtrovaných tras si uživatel dále může upravit zobrazení dle své libosti. Při prvním načtení stránky jsou k jednotlivým trasám přiřazeny náhodně barvy ze seznamu barev. Ty si uživatel může změnit v menu, které se zobrazuje na pravé straně obrazovky. Zde jsou na výběr možnosti jako změna barvy regionu nebo jednotlivých tras a tloušťka hranice regionů. Popřípadě se v tomto nastavení lze i nastavit hranice jako průhledné.

Uživatel může smazat jednotlivé aktivity z databáze aplikace. Již při zpracování si server ukládá informace o tom, které aktivity se navzájem ovlivňují. Například, pokud se dvě aktivity navzájem filtrovaly, tak si server ukládá tuto informaci. Po smazání dojde k chytrému přepočítání aktivit, zároveň se uživateli znovu zobrazí komponent pro stav načítání, která se aktualizuje. Pokud si uživatel přeje vrátit některé ze smazaných aktivit, tak si může „Resetovat účet“ po kliknutí na svoje jméno v pravém horním rohu obrazovky. Aplikace aktualizuje a stahuje nové aktivity po každém přihlášení. Ovšem uživatel se nepřihlašuje příliš často díky nastavení aplikace na pamatování uživatele, takže bylo přidáno tlačítko pro manuální spuštění aktualizace aktivit. Během dalšího vývoje je možné využít tzv. *Webhooks*, které poskytuje Strava. Ve své podstatě jsou to požadavky, které pošle Strava na registrované API, pokud jeden z jeho autorizovaných uživatelů nahraje novou aktivitu[14]. Toto nebylo implementované z důvodu využití lokálních serverů po drtivou většinu vyvíjeného času, *Webhooks* fungují pouze na veřejně dostupném API.

Aplikace byla vytvořena pomocí moderních technologií, které umožnily lehkou implementaci responzivního chování. Pokud uživatel přistoupí na webovou aplikaci pomocí klienta s omezenou šířkou prohlížeče, tak se aplikace přizpůsobí. Tyto změny jsou hlavně přesunutí hlavního navigačního menu do tzv. *hamburgeru* ve formě tří horizontálních úseček signalizující rozevírací seznam. Po kliknutí na tuto ikonku se znovu objeví celý navigační seznam. Podobné chování se zároveň děje na pravé straně obrazovky, kdy se menu pro přizpůsobení zobrazení aktivit a regionů schová. Místo menu se objeví šipka, která po kliknutí seznam znovu zobrazí uživateli.

[14]Strava API Webhooks dostupné z: https://developers.strava.com/docs/webhooks/

Kapitola 5

Testování

Kapitola ukazuje výsledky implementace a její následné testování proti uživatelské chybě.

5.1 Přihlášení uživatele

Při řešení přihlášení bylo hlavně testované případné zamítnutí autorizace. Pokud klient nezíská správné povolení, uživateli se zobrazí chybová hláška vysvětlující problém. Toto chování lze vidět v obrázcích D.1 a D.2.

5.2 Seznam aktivit

Obrázek D.3 zobrazuje webovou stránku po úspěšném přihlášení uživatele. Je zde vidět seznam aktivit i stránkování. Součásti testu bylo měnění komponentu stránkování, například vložení textu na špatné místo, což by aplikaci mohlo rozbít – dle výsledku testu nelze pomocí stránkování zobrazený seznam rozbít. Po plném načtení aktivit se tlačítka pro smazání sama zpřístupnila, dle dokumentace v sekci 4.7. Obrázek D.4 zobrazuje výsledek kliknutí na aktivitu, kdy se otevřel její detail obsahující i trasu běhu. Zde jsou také vidět, že po najetí kursorem se uživateli zobrazí dodatečné akce účtu jako jeho resetování nebo smazání.

5.3 Mazání aktivit

Po smazání aktivit, které se neovlivňují, nedošlo k žádnému přepočítávání a aplikace ihned zpřístupnila plnou funkcionalitu. Před úplným smazáním jakéhokoliv prvku se uživateli vždy zobrazí potvrzovací dialog, který dynamicky mění obsah dle mazaného objektu, a tedy zmenšuje šanci náhodné a nechtěné uživatelovi akce. Dialog je viditelný v obrázku D.5.

Po smazání všech aktivit z účtu aplikace správně zobrazila stránku, která uživatele informuje o neexistenci žádných aktivit dle obrázku D.6. Reakce na jednotlivé akce jsou v některých případech opožděné, to se děje hlavně z důvodu vzdálenosti databáze a lokálního API. Během doby zpracování aktivit se uživateli korektně zobrazoval komponent pro načítání, jak lze vidět v obrázku D.7.

Obdobně bylo testováno i mazání a resetování účtu na stránkách zobrazující zabrané území a filtrované cesty.

5.4 Voroného mapa a filtrované cesty

Stránky zobrazující vypočítané zabrané území uživatele a filtrované trasy fungují responzivně a zobrazují data. Voroného regiony nejsou naprosto přesné, v malém procentu případu se stává, že uživatel projde region, ale není mu přiřazen. V rámci testování jediný možný důvod, který může stát za nepřesností, je výpočet samotných regionů dle knihovny *Scipy*. Toto chování může nastat obzvláště, pokud uživatel projde blízko hranice samotného regionu. Ukázka responzivní aplikace je na obrázku D.8.

V předešlých sekcích (obrázek 4.4) byla trasa zobrazena pomocí *Leaflet*. Nyní lze na obrázku D.9 ve webové aplikaci vidět stejnou trasu, zobrazenou pomocí *OpenLayers*.

Kapitola 6

Závěr

Na závěr jde říci, že všechny body bakalářské práce byly splněny. Cílem bylo vytvořit webovou aplikaci zobrazující území rozdělené dle Voroného regionů, které se přiřazují dle trasy.

V kapitole 2 byl vykonaný průzkum problematiky *OpenStreetMap, GPS* a dostupných fitness aplikací. Na základě průzkumu bylo pro záznam dat rozhodnuto využití již existující mobilní aplikace. Vývoj nové mobilní aplikace by v rámci bakalářské práce byl možný, ale nikdy by se kvalitou zpracování nevyrovnal již existujícím konkurentům. Toto propojení navíc umožňuje uživateli využít jeho dříve zaznamenané aktivity. Jako hlavní zdroj dat byla vybraná aplikace Strava, která má nejlepší podporu vývojářů třetí strany.

V následující kapitole byly shrnuty uživatelské případy užití, pro jednoduchost a přehlednost zobrazeny ve formě modelu. Na základě průzkumu popsaném v sekci 3.2 byla pro implementaci serverové a klientské části zvolena technologie *Vue* a *ASP .NET*. Po porovnání různých typů vyšla najevo jasná výhoda při využití relační databáze. Dle popisu v sekci 3.5 nebylo zřejmé, která technologie se více hodí pro implementaci uživatelského zobrazení. Pro názorné zobrazení návrhu uživatelského rozhraní byly vytvořeny tzv. *Wireframe*, viz sekce 3.7.

Kapitola čtvrtá se dá považovat za nejdůležitější kapitolu. Obsahuje popsané základy celé implementace, včetně využitých technologií. Na počátku je vysvětlené, jakými způsoby jsou vypočteny data jednotlivých Voroného regionů a jak jsou vloženy do databáze. Pro tvorbu samotné databáze byla zvolen *Code-First* přístup, který poskytuje *Entity Framework v C#*. Následně jsou zde řešeny otázky přihlášení, autentizace a zpracování dat. Pro filtraci cest byl implementovaný vlastní algoritmus na základě vzdálenosti jednotlivých souřadnic aktivit. Pro přiřazení zabraného území uživateli byla využita kombinace API od vývojářů OpenStreetMap – Overpass adatabáze. Pro zobrazení těchto vypočítaných dat byla pomocí technologií *Vue a Vuetify* vytvořena webová aplikace. V kapitole byla také řešena otázka responzivity z důvodu velkého množství uživatelů mobilních telefonů.

Z testování vyplynulo, že v určitých částech rychlost zpracování aktivit může být snížena z důvodu hardwarových omezení. Také je zde malá šance, že region nebude přiřazen kvůli problému přesnosti.

Literatura

[1] Agafonkin, V. *Leaflet* [online]. [cit. 17-01-2022]. Dostupné z: https://openlayers.org/.

[2] Agafonov, E. *Multithreading with C# Cookbook*. Packt Publishing Ltd, 2016. ISBN 9781785884009.

[3] Alassouli, D. H. *Creating and Managing Virtual Machines and Networks Through Microsoft Azure Services for Remote Access Connection*. 1. vyd. Packt Publishing, 2021. ISBN 978-1716155390.

[4] Altintaş, M. *Vue3 OpenLayers Documentation* [online]. [cit. 30-04-2022]. Dostupné z: https://vue3openlayers.netlify.app/#usage.

[5] Asghar, M. *Master in HTML and CSS*. Skill Seeker Publications PVT LTD., 2017.

[6] Auth0. *Introduction to JSON Web Tokens* [online]. [cit. 20-04-2022]. Dostupné z: https://jwt.io/introduction.

[7] Bennett, J. *OpenStreetMap*. Packt Publishing Ltd, 2010. ISBN 9781847197511.

[8] Blanco Silva, F. *Mastering SciPy*. 1. vyd. Packt Publishing, 2015. ISBN 9781783984749.

[9] Bloom, R. *Apache Server 2.0: The Complete Reference*. 1. vyd. McGraw Hill Professional, 2002. ISBN 978-0072228038.

[10] Boyd, R. *Getting Started with OAuth 2.0*. 2012.

[11] Buckey, A. *Personal Data Collection*. 1. vyd. ABDO, 2019. ISBN 9781532173110.

[12] Camden, R., Francesco, H. D., Gurney, C., Kirkbride, P. a Shavin, M. *Front-End Development Projects with Vue.js*. Packt Publishing Ltd, 2020. ISBN 9781838981044.

[13] Cavanaugh, R. *Strava vs. MapMyRun* [online]. Květen 2018 [cit. 06-05-2022]. Dostupné z: https://www.digitaltrends.com/health-fitness/strava-vs-mapmyrun/.

[14] Chaplain, C. *Global Positioning System (GPS)*. DIANE Publishing, 2011. ISBN 9781437939774.

[15] Clark, J. *Top 10 backend frameworks* [online]. [cit. 06-05-2022]. Dostupné z: https://blog.back4app.com/backend-frameworks/#7_ASP_NET_Core.

[16] Crockford, D. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008. ISBN 9780596554873.

[17] Diggelen, F. van a Enge, P. *The World's first GPS MOOC and Worldwide Laboratory using Smartphones*. Tampa Convention Center, 2015.

[18] El Rabbany, A. *Introduction to GPS*. Artech House, 2002. ISBN 9781580531832.

[19] Fileformat. *GPX File Format* [online]. [cit. 17-01-2022]. Dostupné z: https://docs.fileformat.com/gis/gpx/.

[20] Garmin. *Flexible and Interoperable Data Transfer SDK* [online]. [cit. 17-01-2022]. Dostupné z: https://developer.garmin.com/fit/overview/.

[21] Gupta, D. *Cookie-based vs. Cookieless Authentication: What's the Future?* [online]. [cit. 21-04-2022]. Dostupné z: https://www.loginradius.com/blog/engineering/cookie-based-vs-cookieless-authentication/.

[22] Hangfire. *Hangfire Documentation* [online]. [cit. 23-04-2022]. Dostupné z: https://docs.hangfire.io/en/latest/.

[23] Hazzard, E. *OpenLayers 2.10 Beginner's Guide*. Packt Publishing Ltd, 2011. ISBN 9781849514132.

[24] IRFANULLAH, M. *HTML*. Sanria Books, 2014. ISBN 9781503389304.

[25] Kaspersky. *What are Cookies?* [online]. [cit. 22-04-2022]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/cookies.

[26] Khan, O. M. A. *C# 7 and .NET Core 2.0 High Performance*. Packt Publishing Ltd, 2018. ISBN 9781788474603.

[27] Klein, R. *Concrete and Abstract Voronoi Diagrams*. 1. vyd. Springer, Berlin, Heidelberg, 1989. ISBN 9783540520559.

[28] Lee, M. *Under Armour MapMyRun Review* [online]. Duben 2021 [cit. 05-05-2022]. Dostupné z: https://www.juiced29.com/mapmyrun-review/#MapMyRun_vs_Strava.

[29] Lerman, J. a Miller, R. *Programming Entity Framework: Code First*. O'Reilly Media, Inc., 2011. ISBN 9781449323844.

[30] Maharry, D. *TypeScript Revealed*. Apress, 2013. ISBN 9781430257257.

[31] Marek, L. *SQL: Podrobný průvodce uživatele*. Grada Publishing a.s., 2018. ISBN 9788027121540.

[32] McNamara, J. *GPS For Dummies*. John Wiley & Sons, 2007. ISBN 9780470199237.

[33] MetaCarta. *OpenLayers* [online]. [cit. 30-04-2022]. Dostupné z: https://openlayers.org/.

[34] Microsoft. *Migrations Overview* [online]. [cit. 25-04-2022]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli.

[35] MongoDB. [online]. [cit. 07-05-2022]. Dostupné z: https://www.mongodb.com/.

[36] Morote, E. S. M. *Introduction to Pinia?* [online]. [cit. 30-04-2022]. Dostupné z: https://pinia.vuejs.org/introduction.html.

[37] Nader, Y. *What is Django?* [online]. [cit. 06-05-2022]. Dostupné z: https://hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django.

[38] NCOPNT. *GPS Accuracy* [online]. [cit. 05-04-2022]. Dostupné z: https://www.gps.gov/systems/gps/performance/accuracy/.

[39] OpenStreetMap. *Overpass API* [online]. [cit. 25-04-2022]. Dostupné z: https://wiki.openstreetmap.org/wiki/Overpass_API.

[40] Oracle. *MySQL* [online]. [cit. 06-05-2022]. Dostupné z: https://dev.mysql.com/doc/.

[41] Patel, J. *10 Best Front end Frameworks to Use For Web Development* [online]. Prosinec 2021 [cit. 06-05-2022]. Dostupné z: https://www.monocubed.com/blog/best-front-end-frameworks/.

[42] Perry, B. W. *Fitness for Geeks: Real Science, Great Nutrition, and Good Health.* O'Reilly Media, Inc., 2012. ISBN 9781449336929.

[43] Ranjan, R. *Geographic coordinate system* [online]. IBM Corporation, březen 2021 [cit. 03-05-2022]. Dostupné z: https://www.ibm.com/docs/en/informix-servers/12.10?topic=data-geographic-coordinate-system.

[44] Ranjan, R. *What is a Framework in Programming* [online]. Net Solutions, říjen 2021 [cit. 25-04-2022]. Dostupné z: https://www.netsolutions.com/insights/what-is-a-framework-in-programming/.

[45] ReactJs. *Virtual DOM and Internals* [online]. [cit. 06-05-2022]. Dostupné z: https://reactjs.org/docs/faq-internals.html.

[46] Seemann, M. a Deursen, S. van. *Dependency Injection Principles, Practices, and Patterns.* Simon and Schuster, 2019. ISBN 9781638357100.

[47] Shiflett, C. *HTTP Developer's Handbook.* Sams Publishing, 2003. ISBN 9780672324543.

[48] Smith, J. *Entity Framework Core in Action, Second Edition.* Simon and Schuster, 2021. ISBN 9781617298363.

[49] Stackoverflow. *Stack Overflow Developer's survey 2021.* Stackoverflow, 2021. Dostupné z: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks.

[50] Strava. *Strava Dashboard* [online]. [cit. 25-01-2022]. Dostupné z: https://www.strava.com/dashboard.

[51] Tsui, J. B.-Y. *Personal Data Collection.* John Wiley & Sons, 2005. ISBN 9780471712572.

[52] Turner, A. *Introduction to Neogeography.* 1. vyd. O'Reilly Media, Inc., 2006. ISBN 9780596529956.

[53] VueJs. *What is Vue?* [online]. [cit. 26-04-2022]. Dostupné z: https://vuejs.org/guide/introduction.html#the-progressive-framework/.

[54] Vuetify. *Why Vuetify?* [online]. [cit. 27-04-2022]. Dostupné z: https://next.vuetifyjs.com/en/introduction/why-vuetify/.

[55] Yeung, A. *Introduction to Cross-Site Request Forgery (CSRF)* [online]. [cit. 21-04-2022]. Dostupné z: https://www.loginradius.com/blog/engineering/introduction-to-cross-site-request-forgery-csrf/.

Seznam symbolů, veličin a zkratek

API	Application Programming Interface – Programovací rozhraní pro aplikace
CORS	Cross-Origin Resource Sharing – Mechanizmus sdílení zdrojů na jiné doméne
CSRF	Cross-site Request Forgery – Falšování požadavků napříč weby
CSS	Cascading Style Sheets – Kaskádové styly
DI	Dependency Injection – Injekce závislosti
DOM	Document Object Model – Objektový model dokumentu
EF	Entity Framework – Framework pro objektově relační mapování
FIT	Flexible and Interoperable Data Transfer – Flexibilní a interoperabilní přenos dat
GML	Geography Markup Language – Geografický značkovací jazyk
GPS	Global Positioning System – Globální polohový systém
GPX	GPS Exchange Format – Formát výměny GPS
HTML	Hypertext Markup Language – Hypertextový značkový jazyk
HTTP	Hypertext Transfer Protocol – Hypertextový přenosový protokol
JSON	JavaScript Object Notation – Zápis objektu JavaScript
JWT	JSON Web Token – JSON webový token
MVC	Model-View-Controller – Model architektury aplikace
OOD	Object-Oriented Database – Objektově orientovaná databáze
OSM	OpenStreetMap – OpenStreetMap
REST	Representational State Transfer – Styl na řízení návrhu a vývoje architektury aplikace
SQL	Structured Query Language – Strukturovaný dotazovací jazyk
URL	Uniform Resource Locator – Webová adresa
XML	Extensible Markup Language – Rozšiřitelný značkovací jazyk

Seznam příloh

Příloha A

Spuštění webové aplikace

Pro základní spuštění aplikace lokálně je potřeba mít nainstalované na zařízení následující software:

- .NET 6.0 a vyšší[1]
- Node.js v17.9.0. a vyšší[2]
- Docker Compose, který je součástí Docker Desktop, verze 4.7.0 a vyšší[3]

Zdrojové soubory jsou dostupné z paměťového média přiloženého k bakalářské práci. Pokud máte v plánu využít médium, zkopírujte soubory z něj soubory na pevný disk vašeho počítače, jinak nelze zajistit funkcionalitu.

Další možnost získání aplikace je naklonování repozitáře bakalářské práce z `https://github.com/Tricer121/BP`.

A.1 Kompilace a spuštění

Kompilace je jednoduchá díky využití *Dockeru*. Pro spuštění stačí navigovat do kořenového adresáře práce, kde se nachází soubor *docker-compose.yml*. Následně proveďte dva příkazy:

- docker compose build
- docker compose up

V konzolové řádce se vám zobrazí adresa, na které běží frontend aplikace – `http://localhost:8080`. Backend běží na portu 7040, což není ovšem pro spuštění relevantní informace.

A.2 Spuštění po červenci 2022

I lokální instance aplikace se spoléhá na existenci vzdáleného serveru s databází *MySQL* a *Overpass API*. Pokud máte zájem znovu zprovoznit aplikaci, můžete mě kontaktovat nebo postupovat dle informací v sekci 4.1, která popisuje nastavení serveru. Poté je potřeba upravit soubor *./backend/appsettings.json* s novou URL serveru databáze.

[1] .NET 6.0 dostupné z `https://dotnet.microsoft.com/en-us/download/dotnet/6.0`
[2] Node.js dostupné z `https://nodejs.org/en/download/`
[3] Docker Desktop: `https://docs.docker.com/compose/install/`

Příloha B

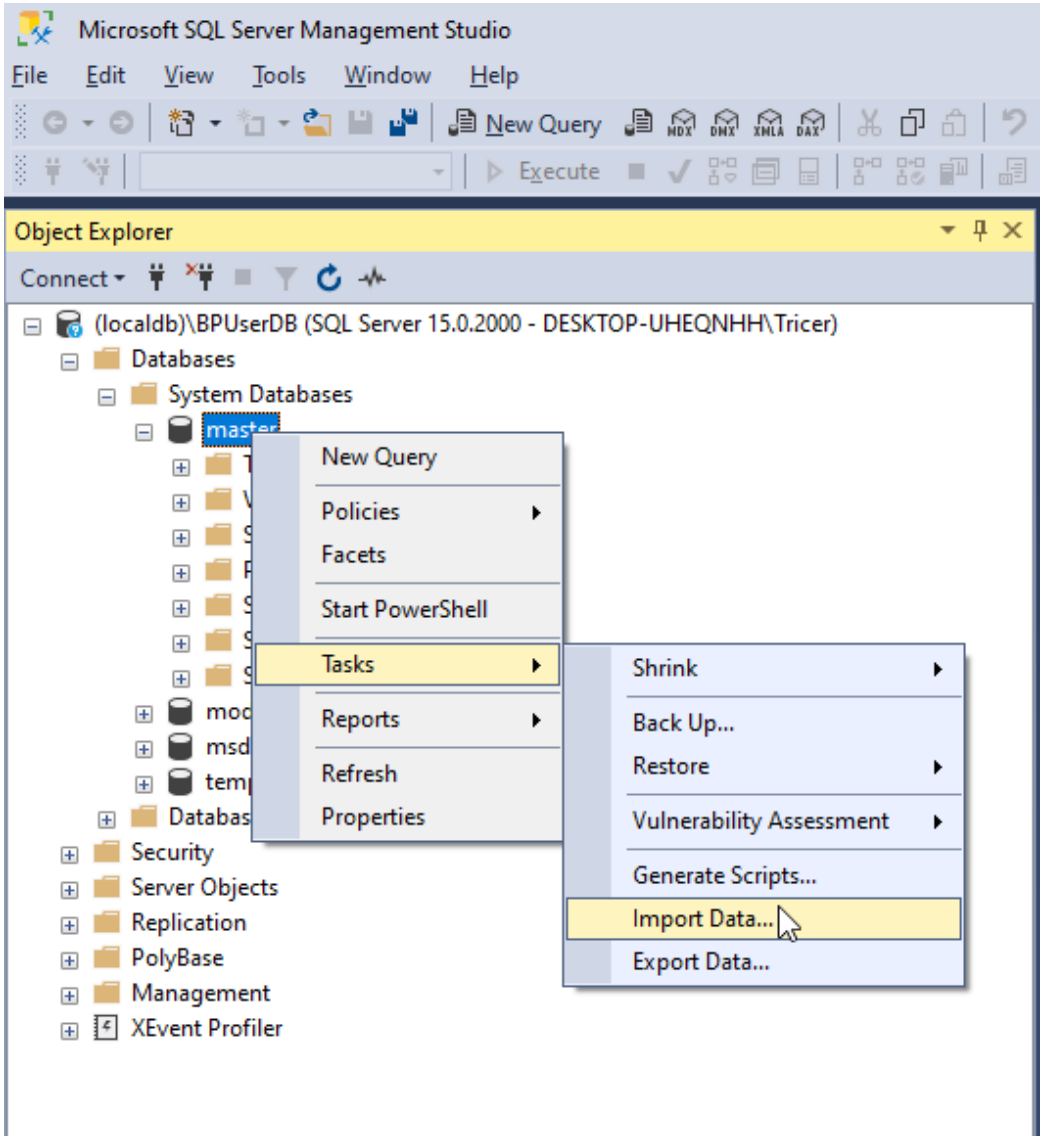
Obsah paměťového média

Adresářová struktura paměťového média je následovná:

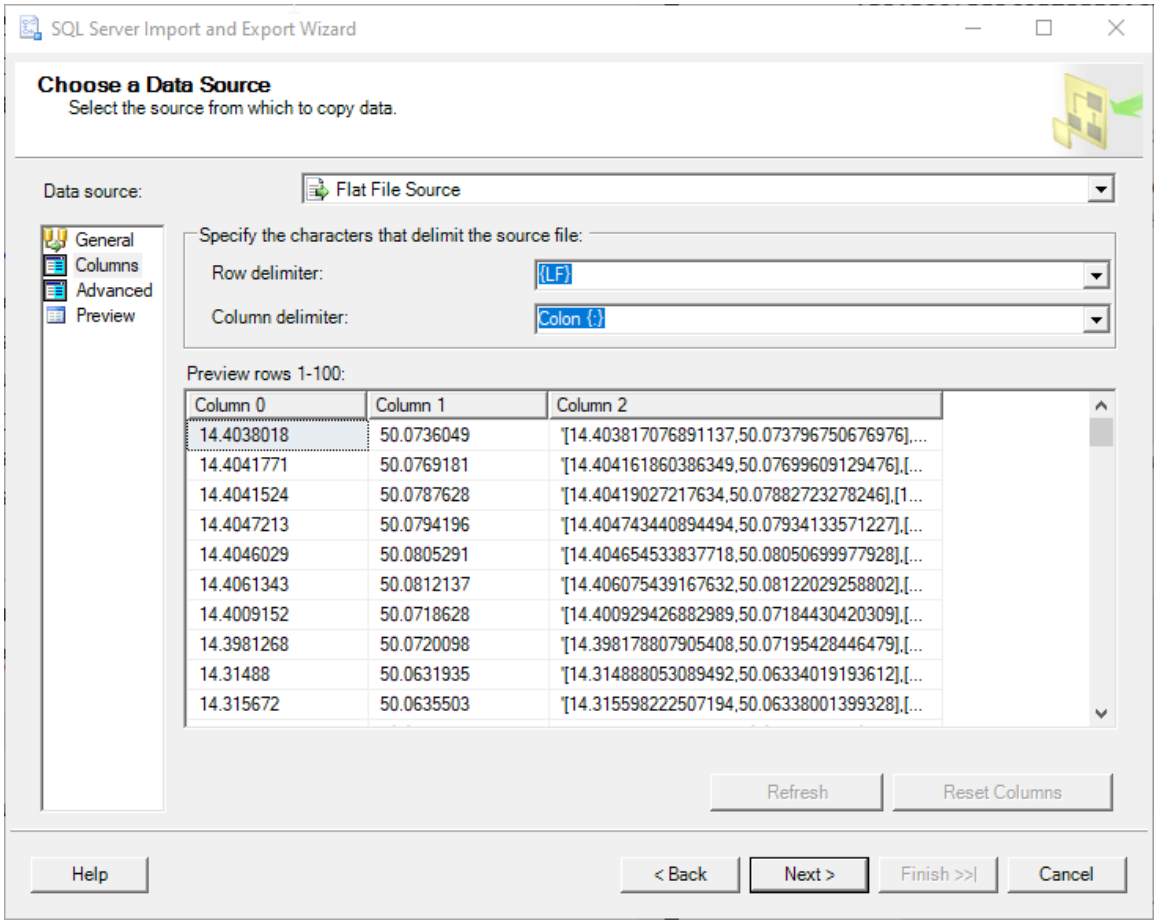
- `/` — kořenový adresář média
  - `docker-compose.yml` — konfigurační soubor dockeru
  - `latexsrc` — složka se zdrojovým kódem BP
  - `backend` — adresář serverové části aplikace
    - `Classes` — pomocné třídy
    - `Common` — pomocné enumy
    - `Migrations` — databázové migrace
    - `Models` — definice entit pro EF
    - `Repositories`
    - `AppDbContext.cs`
    - `Dockerfile` — kompilační soubor dockeru
    - `Program.cs` — předvolený soubor ASP.NET aplikací
    - `appsettings.json` — konfigurační uživatelský soubor
  - `frontend` — adresář klientské části aplikace
    - `src`
      - `assets` — použité obrázky
      - `components` — jednotlivé komponenty
      - `models` — použité třídy
      - `plugins`
      - `router` — Vue směrování
      - `services` — Služby
      - `store`
      - `styles` — definované barvy, proměnné, písmo
      - `views` — jednotlivé stránky
      - `App.vue`
      - `main.ts` — konfigurační soubor
    - `Dockerfile` — kompilační soubor dockeru
    - `index.html` — základní stránka aplikace
    - `package.json` — konfigurační soubor Vue
  - `osm zpracovani` — skripty pro výpočet Voroného regionů
    - `extract_nodes_from_osm.py` — skript k extrakci dat z .osm souboru
    - `run_voronoi.py` — skript pro výpočet Voroného regionů

Příloha C

Použití SQL Studia



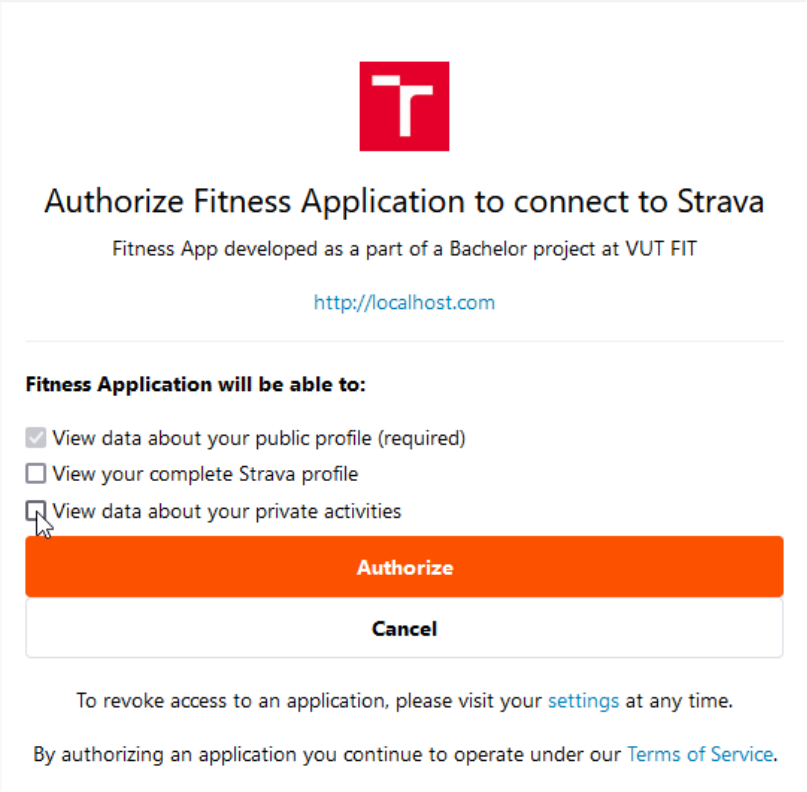
Obrázek C.1: Kontextové menu v SQL Server Management Studiu



Obrázek C.2: Rozdělení dat do sloupců v SQL Server Management Studiu

Příloha D

Testování aplikace



Obrázek D.1: Špatně nastavené povolení uživatele v aplikaci Strava

Obrázek D.2: Selhání přihlášení uživatele

Obrázek D.3: Seznam aktivit

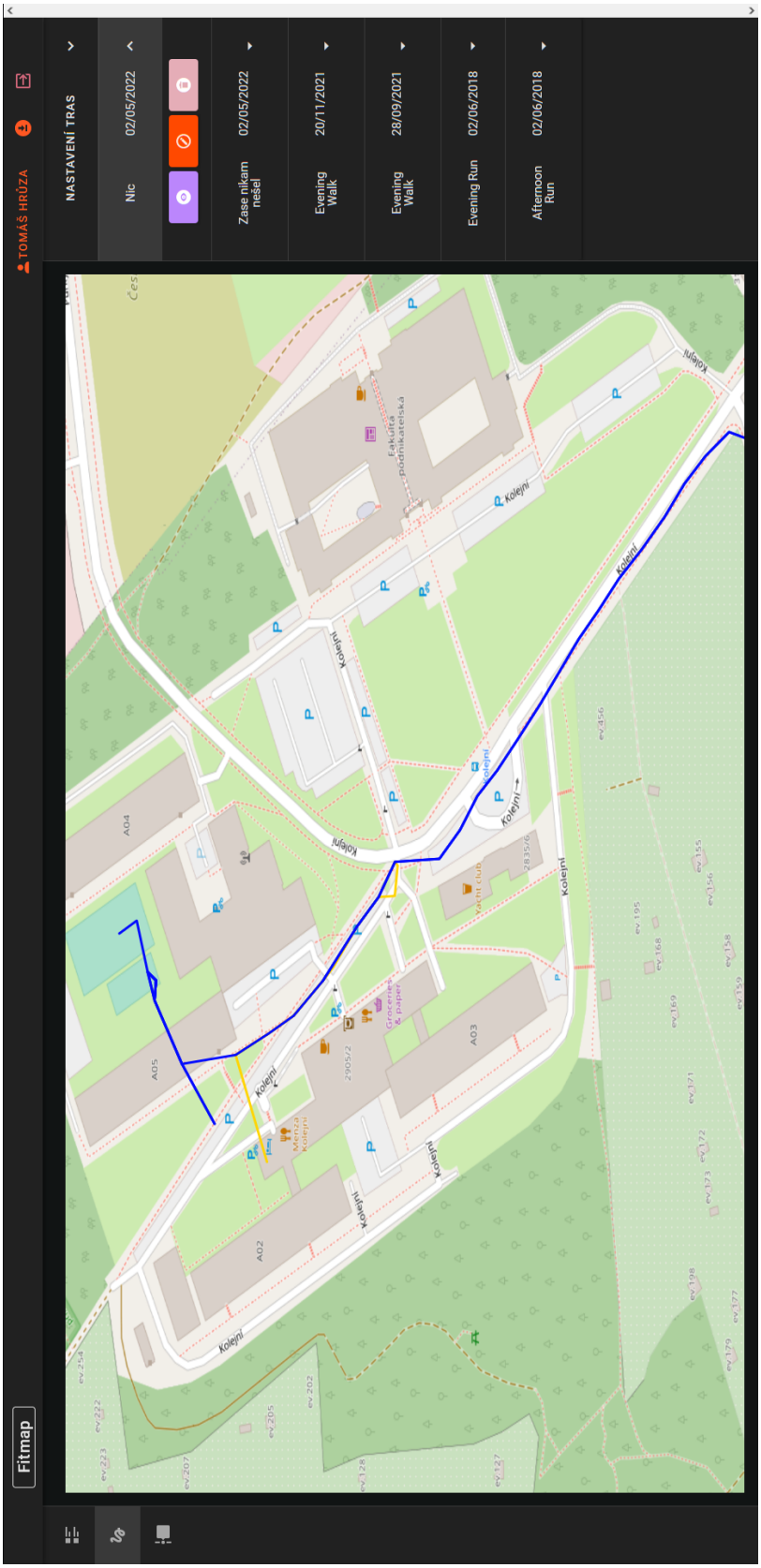
Obrázek D.4: Detail aktivity

Obrázek D.5: Varovný dialog

Obrázek D.6: Zobrazení bez aktivit na účtě

Obrázek D.7: Načítací komponenta při zpracování aktivit

Obrázek D.8: Responzivní zobrazení zabraného území



Obrázek D.9: Filtrace tras