FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Projekt

Webová podpora automatizovaného dopravního průzkumu ulic

Studijní program: S081 – Aplikovaná informatika
Studijní obor: 1802T007 – Informační technologie
Autor práce: **Lilian Luca**
Vedoucí práce: Ing. Jan Kolaja PhD.

Liberec 2024

ZADÁNÍ BAKALÁŘSKÉHO PROJEKTU

Název projektu:
Webová podpora automatizovaného dopravního průzkumu ulic

Zadání:

1. Vytvořte webovou aplikaci, která umožní sběr a vizualizaci dat z mobilního zařízení, které snímá vzdálenosti cyklisty a ostatních vozidel v silničním provozu ve městě.

    1. Vyberte vhodné softwarové nástroje pro realizaci aplikace.

    2. Zajistěte bezpečnost serveru i aplikace.

Prohlášení

Prohlašuji, že svůj projekt jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mého projektu a konzultantem.
Jsem si vědom toho, že na můj projekt se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mého projektu pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li projekt nebo poskytnu-li licenci k jeho využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Beru na vědomí, že můj projekt bude zveřejněn Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

23. 5. 2024 Lilian Luca

Webová podpora automatizovaného dopravního průzkumu ulic

Abstrakt

Cílem této práce je vytvořit webovou aplikaci, která umožní vkládání zeměpisných dat spojených s uživatelskými údaji, konkrétně zeměpisných souřadnic míst, kde došlo k porušení minimální vzdálenosti mezi cyklistou a předjíždějícím autem. Následně aplikace tyto data zpracuje a vygeneruje heatmapu, která vizualizuje úroveň bezpečnosti konkrétních ulic. Další součástí této práce je nasazení výsledné webové aplikace na linuxový server. Na serverové straně byla použita technologie Node.js s frameworkem Express.js a databáze MongoDB pro ukládání dat. Na straně klienta byly využity základní webové technologie, včetně HTML, CSS a JavaScriptu, spolu s knihovnou Leaflet.js pro práci s heatmapou. Pro nasazení byl použit webový server Nginx.

Klíčová slova: Heatmapa, Web, JavaScript, Leaflet.js, MongoDB, Nginx

Web support for automated traffic survey of streets

Abstract

The goal of this work is to create a web application that will provide the ability to insert geographic data into a database. The application will then process this data and generate a heatmap that visualizes the level of safety of specific streets. Another part of this work is to deploy the resulting web application on a Linux server. On the server side, Node.js technology with the Express.js framework and the MongoDB database were used to store the data. On the client side, basic web technologies including HTML, CSS and JavaScript were used, along with the Leaflet.js library to work with the heatmap. The Nginx web server was used for deployment.

Keywords: Heatmap, Web, JavaScript, Leaflet.js, MongoDB, Nginx

Poděkování

Rád bych poděkoval panu Ing. Janu Kolajovi Ph.D za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářského projektu.

Obsah

Seznam zkratek

API	Application Programming Interface
HTML	HyperText Markup Language
XML	Extensible Markup Language
CSS	Cascading Style Sheets
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
W3C	World Wide Web Consortium
SQL	Structured Query Language
NoSQL	Not only SQL
JSON	JavaScript Object Notation
BSON	Binary JSON
ODM	Object Data Modeling

Seznam obrázků

Úvod

Městská doprava je stále důležitějším aspektem života, přičemž bezpečnost cyklistů v provozu představuje klíčový problém moderních měst. I když cyklistika přináší výhody jako snížení emisí a zlepšení zdraví, cyklisté často čelí nebezpečným situacím. Proto je nutné vyvinout efektivní nástroje pro monitorování a analýzu bezpečnosti na silnicích.

Tento projekt se zaměřuje na vývoj webové aplikace, která umožňuje sběr a vizualizaci dat z mobilních zařízení. Tato data zahrnují případy, kdy byla porušena minimální vzdálenost mezi cyklistou a předjíždějícím vozidlem. Hlavním cílem je vytvořit heat mapu, která poskytne užitečné informace o bezpečnosti jednotlivých ulic a pomůže identifikovat oblasti, kde je potřeba zlepšit podmínky pro cyklisty.

První část práce se věnuje výběru vhodných softwarových nástrojů pro realizaci aplikace. Byly vybrány technologie, které zajišťují škálovatelnost a bezpečnost, aby aplikace v budoucnu zvládla zpracovat velké množství dat.

V praktické části práce je popsán samotný vývoj aplikace - zpracování a ukládání dat do databáze a jejich vizualizaci ve formě heat mapy.

Poslední část práce se zabývá zabezpečením vytvořené API a nasazení výsledné aplikace.

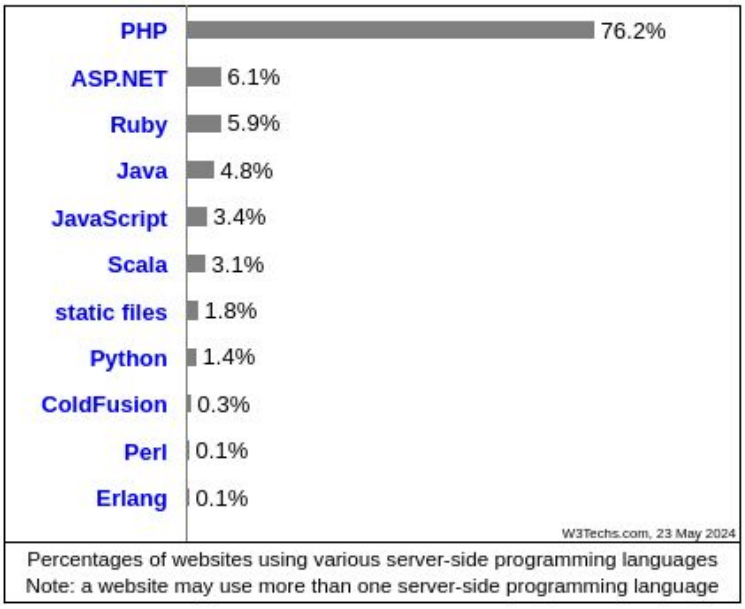
1 Rešerše technologií

V této kapitole bude proveden průzkum možností technologií v několika klíčových oblastech spojených s vývojem webových aplikací. Konkrétní pozornost bude věnována frontendu, backendu a databázím.

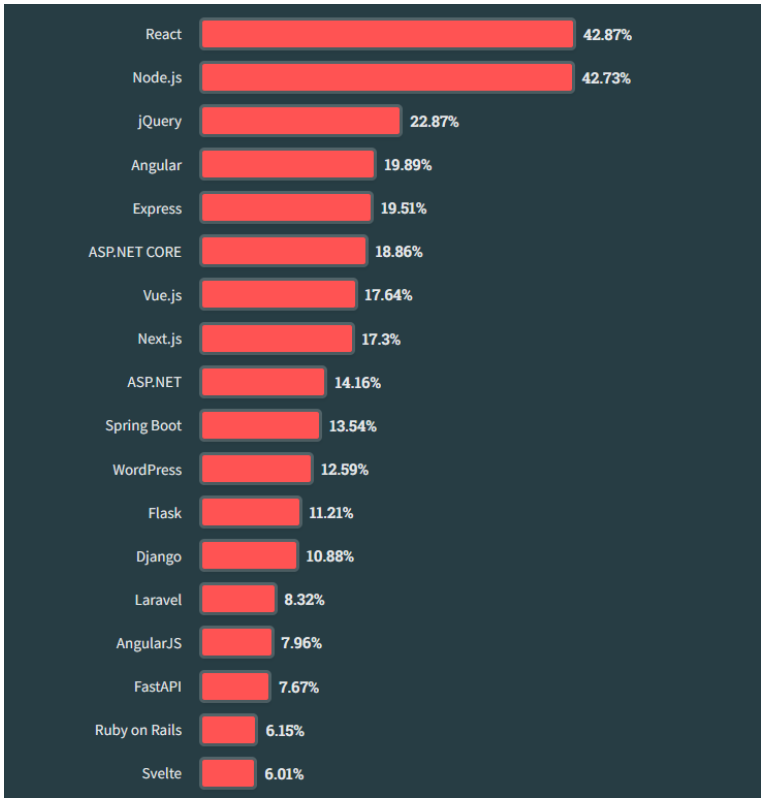
1.1 Technologie pro backend

Jelikož cílem této práce je vytvořit webovou aplikaci, která bude poskytovat možnost ukládání dat v databázi, tak je potřeba mít backend, který bude zpracovávat data a komunikovat s databází.

V současnosti mezi nejpoužívanější backendové programovací jazyky patří PHP, ASP.NET, Ruby a Java (viz Obr. 1.1). Mezi nejpoužívanější frameworky patří Node.js, Express.js, Spring Boot, Flask, Django, Laravel a mnoho dalších (viz Obr. 1.2). Volba Node.js a Express.js pro backend byla učiněna z důvodu jejich flexibility. Tyto technologie umožňují psát kód od jednoduchých až po složitějších API.



Obrázek 1.1: Nejpoužívanější backendové programovací jazyky [15]



Obrázek 1.2: Nejpoužívanější frameworky v roce 2023 [12]

1.2 Technologie pro frontend

Dalším úkolem bylo zobrazovat zpracovaná data ve formě heatmapy na webové stránce, takže je třeba používat základní webové technologie tzn. HTML, CSS a JavaScript.

Aby bylo možné pracovat s mapou a vykreslit ji je potřeba nejprve získat jednotlivé mapové dlaždice přes nějaké API. Příklady služeb, které poskytujou takové API jsou Google Maps, Mapy.cz, Mapbox anebo také OpenstreetMap. Pro tento projekt bylo zvoleno OpenstreetMap API z jednoho prostého důvodu - je to zcela zdarma a opensource.

Dalším krokem je najít vhodnou knihovnu nebo nástroj, který by mohl získané mapové dlaždice vykreslit na webové stránce. Takové nástroje jsou např.: Leaflet.js, Maplibre, Mapbox, Google Maps, OpenLayers, CesiusJS a další... Jednou z nejoblínějších a nejpoužívanějších takových knihoven je právě Leaflet.js. Výhody této knihovy jsou: jednoduchost, výkon a použitelnost, což je ideální pro tento projekt.

Nakonec ještě bude potřebný nástroj, který nám umožní vykreslit zpracovaná data ve formě heatmapy. Pro tento účel výše jmenované knihovny většinou poskytujou nástroje pro práci s heatmapou.

1.3 Databázové technologie

Důležitou funkcionalitou této aplikace je perzistentní uložení geografických dat do databáze. Databázových systémů, které nám umožní ukládat geografická data, je mnoho, a to jak relačních, tak i nerelačních.

Relační databáze jsou tradiční volbou pro mnoho aplikací díky své stabilnosti. Typickými příklady relačních databází jsou MySQL, MSSQL, PostgreSQL a Oracle.

Na druhé straně jsou zde nerelační databázové systémy, které poskytují větší flexibilitu a škálovatelnost. Mezi populární nerelační databáze patří MongoDB, Cassandra a Couchbase. Tyto systémy se často používají pro ukládání a zpracování velkých objemů dat.

Pro tento projekt bylo zvoleno MongoDB z toho důvodu, že tento systém poskytuje flexibilní schéma, které usnadňuje práci s různorodými typy geografických informací.

2 Popis využitých technologií

V této kapitole budou podrobněji popisovány technologie, které byly využity při vývoji tohoto projektu.

2.1 Backendové technologie

Backendové technologie představují skupinu nástrojů, frameworků a programovacích jazyků, které jsou využívány pro vývoj části softwaru, která je provozována na serverové straně. Tato část softwaru je zodpovědná za zpracování dat, interakci s databází, poskytování funkcí a obsluhu komunikace s frontendem a dalšími službami.

Backendové technologie často zahrnují programovací jazyky jako JavaScript (s využitím frameworku Node.js), Python, Ruby, Java, PHP a mnoho dalších, a také frameworky jako Express.js, Django, Ruby on Rails, Spring a další. Tyto technologie umožňují vývojářům vytvářet robustní a efektivní webové aplikace, které mohou zpracovávat velké množství dat a poskytovat různé funkcionality uživatelům.

2.1.1 Node.js

Node.js je open-source a multiplatformní běhové prostředí JavaScriptu. Je to oblíbený nástroj pro téměř jakýkoliv projekt. [3]

Autoři uvádí, že Node.js používá mimo prohlížeč JavaScriptový engine V8, který je jádrem prohlížeče Google Chrome. Díky tomu je Node.js velmi výkonný. [3]

Také za jedinečnou výhodu Node.js, autoři považujou to, že miliony frontendových vývojářů, kteří píší JavaScript pro prohlížeč, mohou nyní kromě kódu na straně klienta psát i kód na straně serveru, aniž by se museli učit zcela jiný jazyk. [3]

2.1.2 Express.js

Express.js je považován za minimální a flexibilní framework pro webové aplikace Node.js, který poskytuje robustní sadu funkcí pro webové a mobilní aplikace. [6]

Autoři uvádí, že vytvoření robustního rozhraní API je snadné a rychlé, protože existuje k dispozici nesčetné množství pomocných metod HTTP a middlewaru. [6]

V dokumentaci je také zmíněno, že Express poskytuje tenkou vrstvu základních funkcí webových aplikací, aniž by zakrýval funkce Node.js, které jsou známé a oblíbené. [6]

2.2 Frontendové technologie

Frontendové technologie jsou soubor nástrojů, knihoven, frameworků a programovacích jazyků používaných pro vývoj uživatelského rozhraní webových aplikací. Tyto technologie jsou zodpovědné za prezentaci obsahu a interakci s uživatelem přímo v prohlížeči. Frontendový vývoj zahrnuje práci s HTML pro strukturování obsahu, CSS pro design a stylování, a JavaScript pro interaktivitu a dynamické chování webových stránek. Kromě těchto základních jazyků frontendový vývoj často zahrnuje také použití různých frameworků a knihoven, jako je například React, Angular, Vue pro usnadnění vývoje a správy komplexních uživatelských rozhraní. Frontendové technologie hrají klíčovou roli v tvorbě moderních webových aplikací, které jsou interaktivní a responzivní pro uživatele.

2.2.1 HTML

Jazyk HTML je nejzákladnějším stavebním prvkem webu. Definuje význam a strukturu obsahu webu a také používá ”markup” k anotaci textu, obrázků a dalšího obsahu pro zobrazení ve webovém prohlížeči. [9]

2.2.2 CSS

CSS je jazyk používaný k popisu prezentace dokumentu napsaného v jazyce HTML nebo XML. CSS popisuje, jak mají být prvky zobrazeny na obrazovce, v řeči nebo v jiných médiích. CSS patří mezi základní jazyky otevřeného webu a je standardizován ve všech webových prohlížečích podle specifikací W3C. [9]

2.2.3 JavaScript

JavaScript je lehký interpretovaný (nebo just-in-time kompilovaný) programovací jazyk s prvotřídními funkcemi. Nejznámější je jako skriptovací jazyk pro webové stránky, ale používá ho i mnoho jiných prostředí než prohlížeč, například Node.js, Apache CouchDB a Adobe Acrobat. JavaScript je prototypový, víceparadigmový, jednovláknový, dynamický jazyk podporující objektově orientované, imperativní i deklarativní (např. funkcionální programování) styly. [9]

2.3 Nerelační databázové technologie

Nerelační databázové technologie jsou systémy pro ukládání a správu dat, které se odlišují od tradičních relačních databází. Na rozdíl od relačních databází, které ukládají data do tabulek a udržují vztahy mezi různými tabulkami pomocí klíčů, nerelační databáze ukládají data v nestrukturované, nebo polostrukturované formě, často ve formátu JSON, XML nebo binárních dat. Tyto technologie jsou navrženy tak, aby byly flexibilnější a lépe přizpůsobeny různorodým datovým modelům a rychlejšímu zpracování velkých objemů dat.

Příklady nerelačních databází zahrnují dokumentové databáze, klíč-hodnota úložiště, sloupcově orientované databáze a grafové databáze. Nerelační databázové technologie jsou často využívány v moderním softwarovém vývoji, zejména v aplikacích, které pracují s velkými objemy dat a potřebují flexibilní a škálovatelné úložiště.

2.3.1 MongoDB

MongoDB (z anglického humongous, česky obrovský) je multiplatformní dokumentová databáze. Řadí se mezi NoSQL databáze a místo tradičních relačních databází využívajících tabulky používá dokumenty podobné formátu JSON (MongoDB formát nazývá BSON) a dynamické databázové schéma, které umožňuje vytváření a integraci dat pro aplikace jednodušeji a rychleji. Jedná se o open source software vydaný pod GNU Affero General Public License a Apache licencemi. [10]

2.3.2 Mongoose

Mongoose je knihovna pro objektové modelování dat (ODM) pro MongoDB a Node.js. Spravuje vztahy mezi daty, zajišťuje validaci schémat a slouží k převodu mezi objekty v kódu a reprezentací těchto objektů v MongoDB. [8]

2.4 Další nástroje a služby

V této podkapitole budou popisovány další technologie, které byly použity v tomto projektu.

2.4.1 OpenstreetMap

OpenStreetMap je projekt, jehož cílem je tvorba volně dostupných geografických dat a následně jejich vizualizace do podoby topografických map (např. silniční mapa, turistická mapa, cyklomapa a navigování v nich). Pro tvorbu geodat se jako podklad využívá záznamů z přijímačů globálního družicového polohového systému nebo jiné zpravidla digitalizované mapy, která jsou licenčně kompatibilní.

Projekt byl založen v roce 2004 a využívá kolektivní spolupráce spolu s koncepcí Otevřeného software. Data jsou poskytována pod licencí Open Database License. OpenStreetMap byl inspirován projekty jako je například Wikipedie, umožňuje jednoduchou editaci dat, uchovává kompletní historii provedených změn, výsledky práce jsou dostupné veřejnosti. [14]

2.4.2 Leaflet.js

Leaflet je open source knihovna JavaScriptu, která se používá k vytváření webových mapových aplikací. Poprvé byla vydána v roce 2011, podporuje většinu mobilních a desktopových platforem a podporuje HTML5 a CSS3. Mezi její uživatele patří například FourSquare, Pinterest, Flickr a USGS. [2]

2.4.3 Geoapify

Společnost Geoapify nabízí komerční služby v oblasti geoprostorových a lokalizačních technologií. Platforma Geoapify Location Platform poskytuje rozhraní a hotová řešení, která pomáhají vytvářet vlastní mapy, sestavovat trasy a analyzovat geodata. Jako základ kombinuje nejlepší existující softwarové komponenty a zdroje dat, včetně OpenStreetMap, spojuje je dohromady a navrch přidává pohodlí a jednotné API. [4]

2.4.4 Nginx

Nginx je softwarový webový server s load managment a reverzní proxy s otevřeným zdrojovým kódem. Pracuje s protokoly HTTP (i HTTPS), SMTP, POP3, IMAP a SSL. Zaměřuje se především na vysoký výkon a nízké nároky na paměť. Nginx je dostupný na Unixu, Linuxu a dalších Unix-like systémech pod BSD [13]

3 Vývoj aplikace

V této kapitole bude hloběji prozkoumán proces vývoje aplikace. Konkrétní pozornost bude věnována implementaci různých funkcionalit pomocí vhodných programovacích jazyků a technologií.

3.1 Vykreslení mapy

Základním úkolem této práce je vykreslit mapu. Pro dosažení tohoto cíle se bude využívat knihovna Leaflet.js [1], jak již bylo zmíněno. Proto se bude postupovat podle dokumentace této knihovny, kde bude nalezen správný postup.

Nejprve je potřeba zahrnout CSS a JavaScript soubory následujícím způsobem:

```
<link
    rel="stylesheet"
    href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
    integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
    crossorigin=""
/>
<script
    src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
    integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
    crossorigin="">
</script>
```

Dále je potřeba vytvořit prvek s určitým id na místě, kde bude mapa vykreslena:

```
<div id="map"></div>
```

Naposledy bude vytvořena funkce `setupMap()`, ve které bude inicializována mapa a bude nastaven pohled na příslušné zeměpisné souřadnice a také úroveň přiblížení. Dále, jak již bylo zmíněno na začátku, bude využívána služba OpenStreetMap k získání mapových dlaždic a nakonec tato vrstva bude vložena do této mapy pomocí metody `addTo()`.

```
1 const setupMap = () => {
2   const map = L.map("map").setView([50.77242005174584,
      15.072913082867514], 17);
3   L.tileLayer("https://tile.openstreetmap.org/{z}/{x}/{y}.png",
4   {
5     attribution:
6       '&copy; <a href="https://www.openstreetmap.org/copyright">
      OpenStreetMap</a> contributors',
7   }).addTo(map);
8 }
```

To je všechno, co je potřeba udělat, aby se zobrazila funkční mapa. Výstup je možné vidět na obrázku 3.1.

Obrázek 3.1: Vykreslená mapa

3.2 Nastavení a konfigurace aplikačního serveru

V následujícím kroku bude zaměřeno na nastavení a konfiguraci aplikačního serveru v prostředí Express.js [6]. Bude popsán postup instalace a konfigurace Express.js

a případně dalších závislostí nezbytných pro správné fungování aplikace a také bude popsáno propojení aplikačního serveru s MongoDB databází.

Nejprve je potřeba inicializovat projekt příkazem `npm init` a pro instalaci Express.js frameworku bude spuštěn příkaz `npm install express`. Dále do proměnné express bude importován modul Express.js. Tento modul je framework pro tvorbu webových aplikací v Node.js. Následně bude vytvořená nová instance aplikace Express. Tato instance představuje webovou aplikaci. Dále pomocí příkazu `app.listen()` bude spuštěn server na zadaném portu. Nakonec bude přidán middleware do aplikace, který zpracuje příchozí požadavky s obsahem typu JSON.

```
1 const express = require("express");
2 const app = express();
3 const port = 3000;
4 app.listen(port, () => {
5     console.log(`App listening on port ${port}`);
6 });
7 app.use(express.json());
```

Také pro práci s databází a nastavení komunikace webového serveru s MongoDB databází bude využívána knihovna Mongoose [11]. Podrobnější informace o této knihovně jsou uvedeny v kapitole Popis využitých technologií.

```
1 const mongoose = require("mongoose");
2 mongoose.connect(process.env.DATABASE_URL);
```

3.3 Vytváření koncového bodu pro přidání dat do databáze

V následujícím bloku kódu je vidět definované schéma pro kolekci dokumentů v MongoDB databázi, která obsahuje pole pro zeměpisné šířky (lat), délky (lon) a váhy (weight) jednotlivých bodů. Hned pod tím je definován GET endpoint '/add' pro přidání nových souřadnic do databáze. Pomocí `req.query` se získají vstupní data tj. nové souřadnice z dotazu (query) odeslaného při volání tohoto endpointu. Následně se vytvoří nový dokument v kolekci "Coordinates" pomocí modelu Coordinates, metody `create()` a nových souřadnic získaných z dotazu. Nakonec server odpoví klientovi s HTTP stavovým kódem 201 (Vytvořeno) a vytvořenými souřadnicemi v JSON formátu. V případě neúspěchu nebo jakékoliv chyby server odpoví HTTP stavovým kódem 500 (Vnitřní chyba serveru) a zprávou o chybě v JSON formátu.

```
const coordinatesSchema = new mongoose.Schema({
  lat: { type: Number, required: true },
  lon: { type: Number, required: true },
  weight: { type: Number, required: true }
});
router.get('/add', async (req, res) => {
  try {
    const newCoordinates = req.query;
    const coordinates = await Coordinates.create(newCoordinates);
    res.status(201).json(coordinates);
  } catch (error) {
    res.status(500).json({ message: error });
  }
});
```

3.4 Zpracování vstupních dat

Vstupní data, tedy nové souřadnice, které budou přidávány do databáze, mohou přijít v takové podobě, že se nacházejí mimo určitou ulici, například na chodníku nebo jinde. Je důležité tuto situaci řádně ošetřit a zpracovat data tak, aby byla správně přiřazena k nejbližší ulici. Pro tuto funkci bude využívána služba Geoapify [5], jež nabízí API pro přiřazení zeměpisných bodů k nejbližší ulici. Podrobnější informace o této službě jsou uvedeny v kapitole Popis využitých technologií.

V následujícím bloku kódu bude vytvořena asynchronní funkce `fetchMatchData()`, která přijímá dva parametry: `url` (adresa, na kterou bude odeslán požadavek) a `waypoints` (seznam bodů, které budeme chtít přiřadit k nejbližší ulici). Tato funkce slouží k získání dat ze zadaného URL pomocí metody POST. V ní je vytvořen objekt `body`, který obsahuje požadované vlastnosti. Dále se odesílá asynchronní HTTP POST požadavek a výsledek tohoto požadavku se uloží do proměnné response, z které se pomocí metody `json()` získají data a nakonec tato funkce vrátí zpracovaná data ve formě JSON objektu.

```
const fetchMatchData = async (url, waypoints) => {
  const body = { mode: 'drive', waypoints: waypoints };
  const response = await fetch(url, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(body),
  });
  const data = await response.json();
  return data;
};
```

3.5 Vytváření koncového bodu pro získání dat z databáze

Dále bude vytvořen GET endpoint '/matched', který bude sloužit k získání zpracovaných dat. Aby bylo možné použít zmíněné API, tak je potřeba si vygenerovat API klíč, který následně bude uložen do proměnné `API_KEY`. Dále pomocí modelu `Coordinates` a metody `find()` se do proměnné `coordinates` uloží všechny souřadnice z databáze. Poté se pomocí metody `map()` mapujou získané souřadnice na formát očekávaný službou Geoapify. Následně se využije již definovaná funkce `fetchMatchData()` k získání přiřazených bodů k nejbližší ulici. Nakonec server odpovídá klientovi s HTTP stavovým kódem 200 (OK) a zpracovanými souřadnicemi v JSON formátu. V případě neúspěchu nebo jakékoliv chyby při zpracování požadavku, je klientovi vrácena odpověď s HTTP stavovým kódem 500 (Vnitřní chyba serveru) a zprávou o chybě v JSON formátu.

```
1 router.get('/matched', async (req, res) => {
2   const API_KEY = process.env.GEOAPIFY_API_KEY;
3   const url=`https://api.geoapify.com/v1/mapmatching?apiKey=${API_KEY}`
4   try {
5     const coordinates = await Coordinates.find();
6     const waypoints = coordinates.map((el) => {
7       return { location: [el.lon, el.lat] };
8     });
9     const matchedWaypoints = await fetchMatchData(url, waypoints);
10    res.status(200).json(matchedWaypoints);
11  } catch (error) { res.status(500).json({ message: error }); }
12 });
```

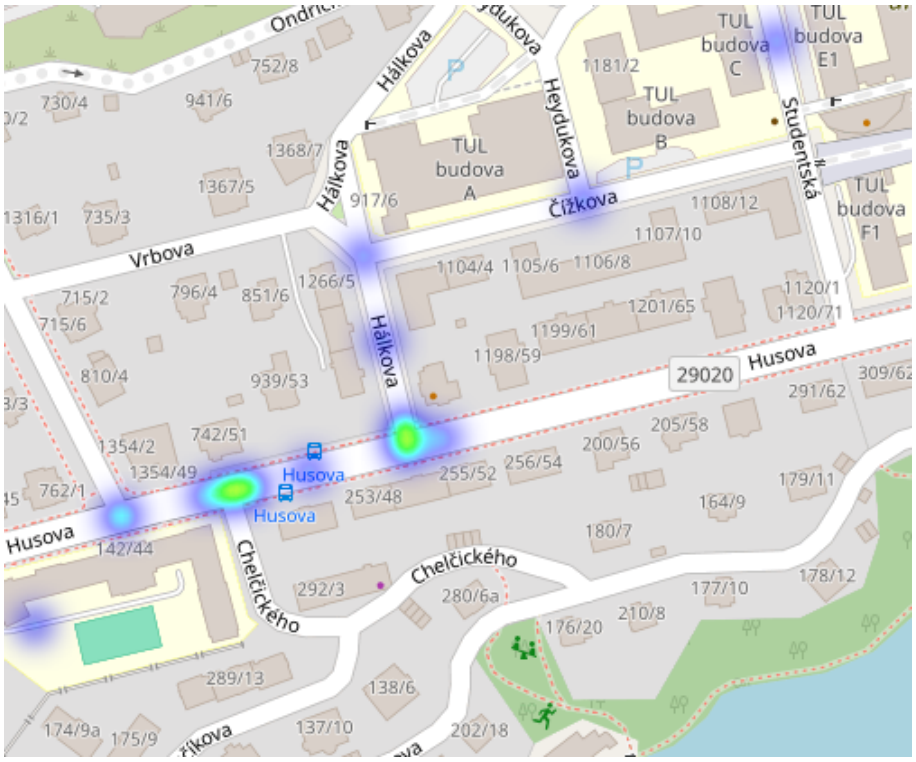
3.6 Zobrazení zpracovaných dat ve formě heatmapy

V této podkapitole bude popsán postup vytvoření heatmapy za pomoci již zmíněné knihovny Leaflet.js [1], která poskytuje funkcionality pro práci s teplotní vrstvou mapy.

Nejprve bude vytvořena funkce `setupHeatLayer()`, která bude sloužit k vytvoření a nastavení teplotní vrstvy na zadané mapě (map) pomocí dat o teplotě (heatData) a poloměru (radius). Dále je inicializováno prázdné pole `heatData`, do kterého budou postupně přidávána data pro heatmapu. Následně se bude volat funkce `fetchData()`, která má za úkol získat zpracované zeměpisné souřadnice z backendu. Postup zpracování těchto souřadnic je detailně popsán v podkapitole Zpracování vstupních dat. Jakmile jsou data k dispozici, jsou zpracována a uložena do pole heatData ve vhodném formátu pro heatmapu. Následující blok kódu zajišťuje, že funkce `setupHeatLayer()`, která slouží k vytvoření teplotní vrstvy bude volána pouze tehdy, když jsou dostupná data pro heatmapu. To zabraňuje chybám, pokud data ještě nebyla načtena.

Zobrazenou teplotní vrstvu je možné si prohlédnout na obrázku 1.2.

```
const setupHeatLayer = (map, heatData, radius) => {
  const heat = L.heatLayer(heatData, { radius: radius }).addTo(map);
};
const heatData = [];
const matchedData = fetchData(
  "http://localhost:3000/api/coordinates/matched"
).then((data) => {
  const { waypoints } = data.features[0].properties;
    waypoints.forEach((waypoint) => {
      const { location } = waypoint;
      const lon = location[0]; const lat = location[1];
      const intensity = 1.5;
      heatData.push([lat, lon, intensity]);
    })});
if (heatData) { setupHeatLayer(map, heatData, 10) }
```



Obrázek 3.2: Zobrazená data ve formě heatmapy

3.7 Zabezpečení vytvořené API

K zabezpečení API byla využita technologie JSON Web Token (JWT), která poskytuje bezpečný a efektivní způsob autentizace a autorizace uživatelů. JWT umožňuje výměnu ověřených a důvěryhodných informací mezi klientem a serverem.

3.7.1 Co je JSON Web Token?

JSON Web Token (JWT) je otevřený standard (RFC 7519), který definuje kompaktní a samostatný způsob bezpečného přenosu informací mezi stranami ve formě objektu JSON. Tyto informace lze ověřovat a důvěřovat jim, protože jsou digitálně podepsané. JWT lze podepsat pomocí tajného klíče (algoritmem HMAC) nebo páru veřejného a soukromého klíče pomocí RSA nebo ECDSA. [7]

3.7.2 Struktura tokenu

JWT se skládá ze tří částí:

Header (Hlavička)

Hlavička se obvykle skládá ze dvou částí: typu tokenu, což je JWT, a použitého podepisovacího algoritmu, například HMAC SHA256 nebo RSA. Například:

```
1 {
2   "alg": "HS256",
3   "typ": "JWT"
4 }
```

[7]

Payload

Druhou částí tokenu je payload, který obsahuje nároky. Nároky jsou prohlášení o entitě (typicky uživateli) a další údaje. Existují tři typy deklarací: registrované, veřejné a soukromé nároky. [7]

Registrované nároky Jedná se o soubor předdefinovaných tvrzení, která nejsou povinná, ale doporučená, aby poskytla soubor užitečných, interoperabilních tvrzení. Některé z nich jsou: iss (issuer), exp (expiration time), sub (subject), aud (audience) a další. [7]

Veřejné nároky Ty mohou být definovány dle libosti těmi, kdo používají JWT. Aby se však předešlo kolizím, měly by být definovány v registru IANA JSON Web Token Registry nebo by měly být definovány jako URI, který obsahuje jmenný prostor odolný proti kolizím. [7]

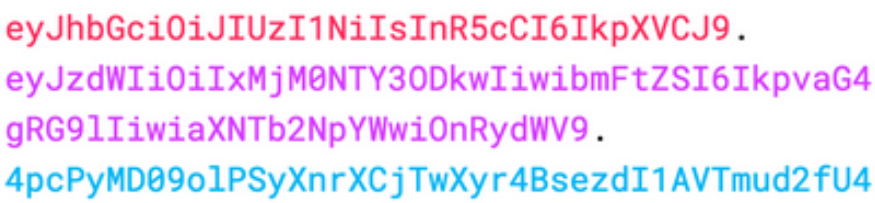
Soukromé nároky Jedná se o vlastní tvrzení vytvořená za účelem sdílení informací mezi stranami, které se dohodnou na jejich používání, a nejedná se ani o registrovaná, ani o veřejná tvrzení. [7]

Signatura

Aby byl token bezpečný, je podepsán. Podpis se vytváří následovně:

1. Vezme se zakódovaná hlavička.
2. Vezme se zakódované tělo.
3. Tyto dvě části se spojí tečkou.
4. Spojený řetězec se zahashuje pomocí algoritmu specifikovaného v hlavičce a tajného klíče nebo privátního klíče.

[7]



Obrázek 3.3: Příklad výsledného JWT tokenu [7]

3.7.3 Implementace autentifikace pomocí JWT

Vytváření koncového bodu pro registrace uživatele

Následující kód definuje POST endpoint, který očekává, že tělo požadavku obsahuje email a heslo uživatele. Pomocí `User.findOne({ email: email })` se kontroluje, zda uživatel s daným emailem již existuje v databázi. Pokud uživatel s daným emailem existuje, vrátí se HTTP status 409 (Conflict) a zpráva "Email already exists". Pokud email není obsazen, generuje se "salt" pomocí `bcrypt.genSalt()`, což je náhodná hodnota používaná k ochraně hesla. Heslo je následně zašifrováno pomocí `bcrypt.hash(password, salt)`, což zajišťuje, že skutečné heslo není uložené v databázi v čitelné podobě, ale jako bezpečný hash. Nový uživatel je vytvořen v databázi pomocí `User.create(req.body)`, kde `req.body` obsahuje všechny potřebné informace včetně zašifrovaného hesla. Pokud je uživatel úspěšně vytvořen, server vrátí HTTP status 201 (Created).

```
1  router.route('/').post(async (req, res) => {
2      try {
3        const { email, password } = req.body;
4        const foundedUserByEmail = await User.findOne({ email: email });
5        if (foundedUserByEmail) {
6          return res.status(409).json({ message: 'Email already exists.'
       });
7        }
8        const salt = await bcrypt.genSalt();
9        const hashedPassword = await bcrypt.hash(password, salt);
10       req.body.password = hashedPassword;
11       const user = await User.create(req.body);
12       res.status(201).json(user);
13     } catch (error) {res.status(500).json({ message: error });}
14   });
```

Vytváření koncového bodu pro autentifikaci uživatele

Následující blok kódu definuje POST endpoint, který očekává, že tělo požadavku obsahuje email a heslo uživatele. Pomocí `User.findOne({ email: email })` se kontroluje, zda uživatel s daným emailem existuje v databázi. Pokud uživatel s daným emailem neexistuje, server vrátí HTTP status 404 (Not Found). Pokud uživatel existuje, pomocí `bcrypt.compare(password, user.password)` se porovná zadané heslo s hashovaným heslem uloženým v databázi. Pokud heslo souhlasí, pokračuje se dál. Pokud heslo nesouhlasí, server vrátí HTTP status 401 (Unauthorized). Pokud jsou přihlašovací údaje správné, vytvoří se objekt, který obsahuje informace o uživateli (v tomto případě pouze email). Pomocí `jwt.sign(myUser, process.env.ACCESS_TOKEN_SECRET)` se vygeneruje JWT token. `process.env.ACCESS_TOKEN_SECRET` obsahuje tajný klíč použitý k podepsání tokenu. Server následně vrátí vygenerovaný token klientovi s HTTP statusem 200 (OK).

```
1 router.route('/auth').post(async (req, res) => {
2   const { email, password } = req.body;
3   const user = await User.findOne({ email: email });
4   if (!user) {return res.status(404)
5     .json({ message: `User with email ${email} doesn't exists.` });}
6   try {
7     if (await bcrypt.compare(password, user.password)) {
8       console.log(process.env.ACCESS_TOKEN_SECRET);
9       const myUser = { name: email };
10      const accessToken = jwt.sign(myUser, process.env.
      ACCESS_TOKEN_SECRET);
11      res.status(200).json({ accessToken: accessToken });
12    } else {res.status(401).json({ message: 'Wrong credetials' });}
13  } catch (error) {res.status(500).json({ message: error });}
14 });
```

3.8 Nasazení aplikace na serveru

V této kapitole bude podrobněji popsán proces nasazení aplikace na serveru. Server, na který bude aplikace nasazena, běží na Ubuntu a jako webový server je použit NGINX. Pro automatické spouštění a správu procesů byl využit správce procesů - PM2.

3.8.1 Příprava a instalace potřebných nástrojů

Nejdříve bylo nutné připravit server s operačním systémem Ubuntu. Toto zahrnovalo aktualizaci systému a instalaci Node.js a pm2.

```
sudo apt update && sudo apt upgrade
sudo apt install nodejs
sudo npm install -g pm2
```

3.8.2 Nasazení aplikace

Aplikace byla nahrána na server pomocí Git. Po nahrání aplikace bylo nutné nainstalovat její závislosti.

```
git clone https://github.com/lilianluca/bp-heatmap
npm install
```

Aplikace byla poté spuštěna pomocí PM2

```
pm2 start server.js --name "heatmap"
```

3.8.3 Konfigurace NGINX

Pro zajištění reverzního proxy a distribuce provozu byl nakonfigurován NGINX. Konfigurační soubor pro aplikaci v NGINX vypadá následovně:

```
server {
        listen 81;
        listen [::]:81;
        root /home/lilian/bp-heatmap/client/build;
        index index.html index.htm index.nginx-debian.html;
        server_name ulice.nti.tul.cz;

        location /lilian {
            alias /home/lilian/bp-heatmap/client/build;
            try_files $uri $uri/ /index.html;
        }

        location /api {
            proxy_pass http://localhost:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }
}
```

Konfigurační soubor byl uložen v /etc/nginx/sites-available/heatmap a poté byl vytvořen symbolický odkaz do složky sites-enabled.

```
sudo ln -s /etc/nginx/sites-available/heatmap /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

Závěr

V této práci byla vyvinuta webová aplikace pro automatizovaný dopravní průzkum ulic, která umožňuje zpracování a vizualizaci dat o vzdálenostech mezi cyklisty a ostatními vozidly v městském provozu. Aplikace využívá moderní technologie jako Node.js s frameworkem Express.js na serverové straně a MongoDB pro ukládání dat. Na straně klienta byly použity HTML, CSS a JavaScript spolu s knihovnou Leaflet.js pro tvorbu heat mapy.

Implementace aplikace byla prováděna s důrazem na škálovatelnost a bezpečnost, což je zásadní pro budoucí zpracování velkého množství dat. Nasazení aplikace na linuxový server s webovým serverem Nginx a správou procesů pomocí PM2 zajišťuje spolehlivý a efektivní provoz.

Další rozvoj aplikace by mohl zahrnovat integraci pokročilejších analytických nástrojů, rozšíření funkcionality o sběr dalších typů dopravních dat a zlepšení uživatelského rozhraní pro lepší interakci uživatelů s aplikací. Tímto způsobem by aplikace mohla ještě více přispět k analýze a zlepšení bezpečnosti v městském prostředí.

Projekt je nahraný na GitHubu a je dostupný na následující adrese: https://github.com/lilianluca/bp-heatmap.git

Použitá literatura

[1] AGAFONKIN, Volodymyr. *Leaflet.js* [online]. 2010. [cit. 2024-04-06]. Dostupné z: https://leafletjs.com/.

[2] AGAFONKIN, Volodymyr. *Leaflet.js* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Leaflet_(software).

[3] DAHL, Ryan. *Node.js* [online]. 2009. [cit. 2024-04-15]. Dostupné z: https://nodejs.org/en.

[4] *Geoapify* [online]. [cit. 2024-04-21]. Dostupné z: https://wiki.openstreetmap.org/wiki/Geoapify.

[5] *Geoapify* [online]. [cit. 2024-04-08]. Dostupné z: https://www.geoapify.com/.

[6] HOLOWAYCHUK, TJ. *Express.js* [online]. 2010. [cit. 2024-04-08]. Dostupné z: https://expressjs.com/.

[7] *JSON Web Token* [online]. b. r. [cit. 2024-05-14]. Dostupné z: https://jwt.io/.

[8] KARNIK, Nick. *Úvod do Mongoose pro MongoDB* [online]. [cit. 2024-04-21]. Dostupné z: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/.

[9] *MDN Web Docs* [online]. 2005. [cit. 2024-04-21]. Dostupné z: https://developer.mozilla.org/en-US/.

[10] *MongoDB* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/MongoDB.

[11] *Mongoose* [online]. [cit. 2024-04-08]. Dostupné z: https://mongoosejs.com/.

[12] *Nejpoužívanější webové technologie v roce 2023* [online]. 2023. [cit. 2024-04-05]. Dostupné z: https://survey.stackoverflow.co/2023/.

[13] *Nginx* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/Nginx.

[14] *OpenStreetMap* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-04-21]. Dostupné z: https://cs.wikipedia.org/wiki/OpenStreetMap.

[15] *Web Technology Surveys* [online]. b. r. [cit. 2024-05-23]. Dostupné z: https://w3techs.com/technologies/overview/programming_language.