Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Webová aplikace pro plánování turistických výletů

Bakalářská práce

Autor: Tomáš Mach
Studijní obor: Aplikovaná informatika (ai3p)

Vedoucí práce: Mgr. Hana Rohrová

Hradec Králové srpen 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 7. 8. 2023 Tomáš Mach

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Haně Rohrová za metodické vedení práce, užitečné rady a vstřícný přístup.

Anotace

Bakalářská práce se zaměřuje na vývoj ucelené webové aplikace pro plánování turistických výletů za využití běhového prostředí Node.js a moderního přístupu tzv. RESTful aplikací, kterého bylo dosaženo pomocí frameworku Express a single page aplikací za využití frameworku React. V práci je vysvětlen princip fungování těchto přístupů spolu s dalšími použitými technologiemi a příklady kódů z vyvíjené aplikace. Dále jsou zde představeny možné alternativy k použitým technologiím. Okrajově se práce také zabývá řešením uživatelských účtů, což zahrnuje implementaci bezpečnostních opatření a mechanismů pro ověření uživatelů. V neposlední řadě je představeno ovládání vyvíjené aplikace a možnosti nasazení spolu s příkladem nasazení v Docker enginu. Práce je vhodná pro všechny vývojáře, kteří mají zájem o vývoj webových aplikací s využitím moderních přístupů.

Annotation

Title: Web application for planning trips

The bachelor thesis focuses on the development of a complete web application for planning tourist trips using the Node.js runtime environment and the modern approach of RESTful applications, which was achieved using the Express framework and single page applications using the React framework. In this paper, the working principle of these approaches is explained along with other technologies used with code examples from developed application. The thesis also presents possible alternatives to the technologies used. Peripherally, the thesis also deals with user account management, which includes the implementation of security measures and user authentication mechanisms. Finally, the control of the developed application and deployment options are presented along with the example of deployment in the Docker engine. The thesis is suitable for all developers who are interested in developing web application using modern approaches.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V současné digitalizované společnosti hrají webové aplikace klíčovou roli v poskytování široké škály služeb a řešení pro uživatele po celém světě. Výhodou webových aplikací je především široká dostupnost na velkém množství platforem od počítačů, až po telefony nebo chytré televize. To umožňuje uživatelům přístup ke službám kdykoliv a odkudkoliv. S nárůstem počtu uživatelů s internetovým připojením a neustálým pokrokem technologií se stává vývoj webových aplikací stále významnějším a atraktivnějším pro programátory a vývojářské týmy.

Teoretická část této práce se zabývá technologiemi a použitými principy při vývoji webové aplikace pro plánování turistických výletů za využití moderních přístupů se zaměřením na běhové prostředí Node.js a framework Express pro vytvoření RESTful webové aplikace a frontendového přístupu single page aplikace pomocí frameworku React. Část práce je také věnována alternativním technologiím, ovládání finální aplikace včetně možností nasazení do produkčního prostředí.

Výsledkem praktické části je ucelená webová aplikace pro plánování sdílených turistických či cyklistických výletů. Ta se skládá ze dvou částí, a to samostatné aplikace pro backend v podobě REST API a samostatné aplikace pro frontend v podobě React aplikace. Výsledná aplikace umožňuje uživatelům naplánovat výlet, na který ostatní uživatelé mají možnost reagovat, a to možnostmi potvrdit účast, nebo naznačit možnou účast. Dále je umožněna uživatelům interakce s ostatními účastníky, případně pořadatelem, pomocí komentářů pod jednotlivými výlety. Výlet po skončení je možné ohodnotit a přidat k němu fotografie. Výsledné hodnocení může následně sloužit jako zpětná vazba pro pořadatele a zároveň, jako možnost umístění výletu do žebříčku nejlépe hodnocených výletů.

2 Cíl práce

Cílem této práce je vytvořit ucelenou webovou aplikaci pro plánování sdílených turistických a cyklistických výletů a v teoretické části popsat vývoj takové webové aplikace. Aplikace umožní uživatelům plánovat vlastní výlety prostřednictvím formuláře, kde zadají klíčové informace jako název, datum konání, popis a další relevantní údaje. Ostatní uživatelé pak budou moci na výlety reagovat možnostmi „zúčastním se“ či „možná se zúčastním“. K výletu budou přihlášení uživatelé mít možnost psát komentáře, to budou moct využít např. na kladení dotazů před začátkem výletu, nebo k dodatečnému slovnímu ohodnocení po skončení výletu. Mezi další funkce bude patřit například hodnocení výletů, kterých se uživatel zúčastnil pomocí skóre (hvězdiček), žebříček nejlepších uživatelů a výletů, přidávání fotografií, vyhledávání dle kritérií a další.

Aplikace bude vhodná pro milovníky turistiky a cyklistiky, kteří se rádi účastní organizovaných turistických nebo cyklistických akcí nebo by rádi takovou akci zorganizovali.

3 Metodika zpracování

Jedním z hlavních cílů bylo uživatelům nabídnout přehledný a jednoduchý nástroj, kde by každý uživatel měl možnost se dozvědět o právě organizovaných turistických či cyklistických výletech a zároveň takový výlet naplánovat pro ostatní uživatele.

Před samotným vývojem aplikace bylo zvažováno několik možných technologií, a postupů, které by byly vhodné pro vývoj použít. Pro finální aplikaci byla použita REST architektura, kdy samotná aplikace byla rozdělena na dvě části, a to backendovou a frontendovou část. Pro obě části byl zvolen jazyk JavaScript. Backendová část aplikace pak využívá populární framework Express nad běhovém prostředím Node.js a frontendová část framework pro tvorbu single page aplikací React. Výběr těchto technologií byl zvolen převážně díky vysoké popularitě těchto technologií a z části i kvůli osobní motivaci se naučit vývoj REST aplikací v kombinaci se single page aplikacemi. Při rozhodování o volbě technologie pro vývoj backendu sehrála velkou roli také jedna z výhod Node.js, a tou je sjednocení jazyků pro backendovou a frontendovou část aplikace.

Pro ukládání uživatelských dat byl využit databázový systém MySQL. Ke komunikaci s databází byl pak použit modul sequelize využívající objektového relačního mapování, to zjednodušuje práci s databází a minimalizuje použití SQL dotazů na minimum.

Pro ukládání stavu frontendové aplikace, především pro udržení přístupového tokenu přihlášeného uživatele, byl využit dodatečný modul Redux Toolkit, ten byl použit především kvůli následnému využití RTK Query, který je jeho součástí, a který zjednodušuje volání na backendovou část aplikace a umožňuje cachování výsledků do tzv. redux store.

Aplikace byla vyvíjena a testována ve vývojovém prostředí Visual Studio Code na platformě Windows 11.

4 Literární rešerše

Tato kapitola je věnována k vysvětlení základních pojmů z oblasti tvorby webových aplikací a vybraných technologií pro vývoj backendových a frontendových částí webových aplikací.

4.1 Webová aplikace

„Všechny webové aplikace jsou webovými stránkami ale všechny webové stránky nejsou webovými aplikacemi“. To v praxi znamená, že webové stránky mají pouze informační charakter, soustředí se na nefunkční obsah (jako text, obrázky apod.). Naopak webové aplikace přidávají webovým stránkám funkce, které umožní uživateli se stránkou interagovat (registrace, vyhledávání, přidávání příspěvků apod.). [1]

Webová aplikace je typ softwaru, který je typicky spuštěn na vzdáleném serveru, který je dostupný ostatním uživatelům internetu, popř. v podnikových sítích (intranetu) skrze webový prohlížeč. Výhodou této implementace je, že uživatelé nemusí aplikaci instalovat, stačí jim pouze webový prohlížeč, který je dostupný na většině zařízení od počítače, až po chytré telefony nebo televize. Další výhodou je, že k aplikaci může přistupovat velké množství uživatelů ve stejný čas. Nevýhodou pak může být nutnost připojení k internetu. Nutno ale podotknout, že internet je díky novým technologiím stále dostupnější, a to i na místech, kde to dříve nebylo možné. [2]

Webové aplikace fungují na principu klient-server a protokolu HTTP (Hypertext Transfer Protocol) a jsou programovány ve vyšších programovacích jazycích (např. PHP, JavaScript, Java, Python atd.). Klientem je typicky uživatel přistupující k aplikaci např. pomocí webového prohlížeče, ale může to být i jiná aplikace. Server je prostředí, kde je webová služba spuštěna. Webový server zpracovává požadavky klienta (např. zobrazení katalogu e-shopu), vygeneruje výsledek (např. načte produkty z databáze) a pošle odpověď klientovi (např. formou webové stránky). [3]

Webový server může ale fungovat i formou API (Application Programming Interface). V tomto případě výstup není určen uživatelům, ale dalším aplikacím. Výsledkem není graficky vypracovaná webová stránka, ale formát, kterému aplikace snadno rozumí (např. XML, JSON, Text apod.).

4.2 Databáze

Databáze je organizovaná kolekce strukturovaných dat, dnes většinou již v elektronické podobě, ale mohou být i tištěné (typicky kartotéky). Elektronická databáze je obvykle řízena SŘBD (Systém Řízení Báze Dat), jedná se o rozhraní mezi databází a uživatelem nebo aplikací. Data, SŘBD a aplikace s nimi spojená jsou spolu často nazývány, jako databázový systém. Mezi populární SŘBD se řadí např. Microsoft Access, Microsoft SQL Server, MySQL, PostgreSQL, Oracle a další. [4] SŘBD zodpovídá za:

- definici dat – prostředky pro definování a úschovu dat,
- údržbu dat – každému členu vyhrazuje záznam popisující dílčí informace o něm,
- manipulaci s daty – umožňuje vkládat, aktualizovat, mazat a filtrovat data,
- zobrazování dat – poskytnutí formy prezentace dat uživateli,
- integrita dat – metody pro zajištění správnosti dat (udržuje databázi konzistentní). [5]

4.2.1 Relační databáze

Relační databáze se staly dominantními již v 80. letech. Data v relačních databázích jsou organizována, jako soubor tabulek se sloupci a řádky. Sloupcem se rozumí atribut, který je definován názvem a množinou přístupných hodnot (datovým typem). Řádek je pak samotný záznam s konkrétními hodnotami. Relační databáze nabízí jeden z nejefektivnějších a nejflexibilnějších způsobů přístupu ke strukturovaným informacím [4]. Vztahy mezi tabulkami jsou pak definovány pomocí tzv. klíčů.

Existují dva základní druhy klíčů, a to primární a cizí klíč:

- **Primární klíč** – je unikátní, může být tvořen více atributy, nejčastěji se používá jako tzv. umělý primární klíč, který využívá sekvencí, což zaručuje zvýšení hodnoty po každém přidaném záznamu.
- **Cizí klíč** – identifikuje konkrétní záznam v jiné (cizí) tabulce. [6]

Příklad propojených relačních tabulek

Tabulka 1 – Uživatelé

ID *INTEGER(11)*	Jméno *VARCHAR(50)*	Příjmení *VARCHAR(50)*
1	Jiří	Novák
2	Zuzana	Kopecká

Zdroj: Vlastní zpracování

Tabulka 2 – Výlety

ID *INTEGER(11)*	Název *VARCHAR(30)*	Vytvořil *INTEGER(11)*
1	Výlet podél Labe	2
2	Výlet na Sněžku	1

Zdroj: Vlastní zpracování

Na příkladu je znázorněno propojení mezi tabulkami Uživatelé a Výlety. Uživatel má svůj primární klíč (atribut „ID“) a v tabulce Výlety se nachází cizí klíč (atribut „Vytvořil“), který odkazuje na primární klíč (atribut „ID“) v cizí tabulce Uživatelé. Znamená to tedy, že „Výlet podél Labe“ vytvořil uživatel s ID číslo 2 (Zuzana Kopecká) a „Výlet na Sněžku“ uživatel s ID 1 (Jiří Novák).

Komunikace s relační databází pak probíhá pomocí dotazovacího jazyka SQL (Structured Query Language). Příklady syntaxe SQL:

- SELECT atribut, (…) FROM název_tabulky (WHERE atribut=hodnota)
  - Výběr filtrování dat
- INSERT INTO název_tabulky atribut, (…) VALUES (hodnota, (…))
  - Vkládání dat
- UPDATE název_tabulky SET atribut=hodnota, (…) (WHERE atribut=hodnota)
  - Aktualizace dat

- DELETE FROM název_tabulky (WHERE atribut=hodnota)
    - Mazání dat
- a další.

4.3 Analýza vhodných technologií

Kapitola popisuje vybrané technologie, které jsou vhodné pro použití k vývoji webových aplikací.

4.3.1 Angular

Angular je komplexní framework pro tvorbu frontendových aplikací od společnosti Google. Je postaven na jazyku TypeScript, což je nadstavba jazyka JavaScript od společnosti Microsoft. Angular je určen pro designování pro různé platformy, jako např. webové aplikace, mobilní webové aplikace, nativní mobilní aplikace a nativní desktopové aplikace. Angular je nástupce starší verze AngularJS, která využívala čistý JavaScript a není s touto verzí zpětně kompatibilní. [7]

4.3.2 Vue

Stejně jako Angular se jedná o framework pro tvorbu frontendu, je vytvořen Evanem You, který dříve pracoval v Googlu. Tam dospěl k názoru, že používání Angularu je v některých případech příliš komplikované, a tak se rozhodl použít části Angularu, které se mu líbily, k vytvoření odlehčeného frameworku Vue. Ačkoliv je Vue odvozen od Angularu, je považován za více podobný Reactu. Na rozdíl od Angularu lze pro programování použít jak JavaScript, tak i TypeScript. [7]

4.3.3 PHP

PHP neboli HyperText Preprocessor je jedním z nejpopulárnějších skriptovacích jazyků pro tvorbu webových aplikací. Z počátku měl problémy týkající se výkonu, a jelikož byl tlak na tvorbu lepších a rychlejších webových aplikací, započal vývoj velkého množství frameworků. PHP kód je vkládán do HTML stránky a je zpracováván na serveru. Mezi nejpopulárnější PHP frameworky se řadí:

- **Codeigniter** – obsahuje velké množství funkcí pro zvýšení rychlosti vývoje webové stránky a poskytuje zabezpečení proti různým útokům.

- **Laravel** – jedná se o open source framework, dle tvůrce vznikl kvůli nedostatku klíčových funkcí v Codeigniter (např. autentizace uživatele).
- **CakePHP**
- **Symfony** [8]

4.3.4 Java

Jednou z možností vývoje v Javě jsou Java Server Pages (JSP). Jedná se o technologii pro vývoj dynamických webových stránek. Používá se k implementaci prezentační vrstvy neboli GUI (grafické uživatelské rozhraní). Princip je podobný jako u PHP, Javovský kód se vkládá do HTML dokumentu. Dnes se již tato technologie příliš nevyužívá a místo toho se používají frameworky. Jedním z nejpopulárnějších je open source framework Spring. Framework zajišťuje několik modulů (např. autorizace, autentizace, komunikace s databází a další), které lze použít na základě požadavků aplikace. [9]

4.3.5 Node.js

Pro pochopení, co přesně je Node.js je potřeba nejdříve pochopit, co přesně je JavaScript (neplést s Javou). JavaScript je skriptovací jazyk, původně určen pouze pro tvorbu dynamických stránek na straně klienta. Byl a dodnes je využíván v internetových prohlížečích. Ty obsahují JavaScriptový engine pro vykonávání JavaScriptového kódu. Části JavaScriptového kódu jsou zahrnuty uvnitř webové stránky (mezi tagy *<script>*). Tyto části kódu jsou zpracovávány prohlížečem při vykreslování stránky, což umožňuje stránce dynamicky přizpůsobovat její vzhled a odpovídat na interakce uživatele. [10] Různé webové prohlížeče využívají různých JavaScriptových enginů. Nejpopulárnější je engine V8, který byl vyvinut společnostní Google pro svůj prohlížeč Google Chrome, dnes je využíván i v dalších prohlížečích, které jsou postaveny na jejich open source jádru Chromium (např. Microsoft Edge).

Vysvětlení Node.js

Node.js je platforma pro vývoj webových, mobilních, serverových, IoT a dalších aplikací v jazyce JavaScript, a to mimo webový prohlížeč. Node.js je totiž

JavaScriptové běhové prostředí (runtime environment) postavené na enginu V8. Node.js umožňuje spouštět JavaScriptový kód stejný jako ve webovém prohlížeči, je však ochuzen o některé funkce, které jsou spojeny pouze s prohlížečem (jako např. HTML DOM – Document Object Model). Node.js obsahuje několik vestavěných modulů, jako např. pro práci se soubory nebo HTTP modul pro vytvoření HTTP serveru a další. Jednou z výhod Node.js je, že lze využívat stejný programovací jazyk pro webové aplikace, jak na straně serveru, tak na straně klienta.

Node.js využívá asynchronní událostmi-řízenou architekturu (event-driven architecture) a je postaven na jednom běžícím vlákně, v kombinaci s rychlým V8 enginem má údajně menší režii než architektury založené na vláknech. Ostatní systémy využívající vlákna pro zpracování souběžných úkolu mívají často složitou režii paměti, kterou Node.js nemá. To má za výsledek vysokou efektivnost a rychlost Node.js aplikací.

Asynchronní událostmi-řízená architektura umožňuje vyřízení několika úkolů najednou, jako např. přepínání mezi několika požadavky od různých klientů. Původní tvůrce Node.js, Ryan Dahl se řídil těmito následujícími klíčovými body:

- Jedno-vláknový, událostmi řízený programovací model je jednodušší na programování a má menší složitost a režii než aplikační servery, které se spoléhají na vlákna pro zpracování více souběžných úkolů.
- Převedením blokovaných volání funkcí do asynchronního kódu, se může nakonfigurovat systém tak, aby spustil událost, až bude blokovaná událost dokončena.
- Využitím V8 enginu z prohlížeče Google Chrome má za následek, že veškerá práce na vylepšení a zlepšení výkonu, které jdou do V8, budou přínosem také pro Node.js. [11]

Node.js vs. jiné aplikační servery

Ve většině aplikačních serverů je schopnost zpracování několika souběžných požadavků implementována pomocí více vláknové architektury. V takovém systému jakýkoliv požadavek na data nebo jiné blokované volání funkce způsobí, že jedno vlákno je pozastaveno a další může být spuštěno (aplikační server neustále spouští a zastavuje vlákna pro zpracování požadavků). Každé pozastavené vlákno čeká, až bude (typicky vstupně/výstupní) operace dokončena, a přitom využívá celý zásobník volání v paměti. To přidává na složitosti aplikačního serveru a jeho režii.
Příklad čtení dat z databáze:

```
vysledek = query("SELECT * FROM tabulka");
//Kód po získání výsledku
```

V tomto případě je kód pozastaven, dokud databázová vrstva nepošle dotaz do databáze a čeká na výsledek nebo chybu. K obsloužení dalšího požadavku v průběhu čekání na výsledek musí být spuštěno další vlákno.

Node.js naopak využívá jedno vláknové běžící jádro. To znamená, že musí využívat událostmi-řízený model pro řízení souběžných úkolů, namísto čekajícího kódu na získání výsledku z blokovaného požadavku (jako např. získání dat z databáze). Jelikož využívá jedno vlákno, nepotřebuje tedy vlákna přepínat. Je tu místo toho událostní smyčka, která odesílá události obslužným funkcím, podle toho, co se právě děje.

Stejný příklad pro čtení z databáze pomocí asynchronního volání:

```
query("SELECT * FROM table", function (err, result) {
          if(err) throw err; //zpracování chyby
          //kód po získání výsledku
}
```

Zpracování funkce query zabere stejný čas jako v předchozím příkladu, ale místo blokování běžícího vlákna se vrátí do událostní smyčky, která může zpracovávat další požadavky. Node.js nakonec vyvolá událost, která zavolá funkci s výsledkem, nebo chybou. [11]

Node Package Manager (NPM)

Node.js umožňuje využívání uživatelských modulů, rozšiřující funkcionalitu. Toho je docíleno pomocí NPM (Node Package Manager), který je automaticky nainstalován s Node.js. NPM umožňuje snadnou instalaci dodatečných balíčků pomocí příkazového řádku, a to příkazem *npm install (název)*. NPM pak automaticky modul najde a stáhne. Součástí NPM je soubor *package.json*, který obsahuje základní informace o projektu (název, verzi, popis, licenci apod.) a zároveň nainstalované moduly. Pro inicializování souboru se použije příkaz *npm init*. Následně vytvořený *package.json* pak vypadá následovně [12]:

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Obr. 1 Ukázka výchozího soubrou package.json
Zdroj: Vygenerováno pomocí NPM

4.3.6 Express

Express je framework pro tvorbu webových aplikací v Node.js. Využívá základní HTTP modul obsažený přímo v Node.js a dodatečný modul connect a jeho komponenty, nazývané „middlewares“. Vývojáři mohou využít jakékoliv knihovny pro konkrétní projekt, což umožňuje vysokou flexibilitu a přizpůsobení. Pokud bychom psali webovou aplikaci jenom pomocí základních Node.js modulů, narazili bychom pravděpodobně na neustále psaní stejného kódu pro podobné úkoly, jako:

- parsování těla HTTP požadavků,
- parsování cookies,
- spravování sessions,
- organizování cest řetězením *if* podmínek na základě URL adres a HTTP metod požadavků,
- určení správných hlaviček odpovědí na základě datových typů.

Řešením těchto problémů může být právě použití frameworku Express. [13]

Jak Express funguje

Samotná instalace frameworku je jednoduchá, stačí využít NPM, který je nainstalovaný přímo s Node.js. V příkazovém řádku se poté odnavigujeme do složky nového Express projektu a zadáme příkaz *npm install express*.

Express má obvykle jeden hlavní soubor (např. *index.js*), ve kterém se provádí následující kroky:

1. zahrnutí závislostí (modulů, knihoven) třetích stran a také našich vlastních modulů (kontroléry, modely, nástroje apod.),
2. nakonfigurování nastavení Express.js (např. template engine v případě full stack aplikace),
3. definování middlewarů, jako obsluha chyb, složky statických souborů (obrázky, styly css), parsery pro cookies a další,
4. definování cest,
5. připojení k databázi,
6. spuštění aplikace.

Když je Express aplikace spuštěna, naslouchá požadavkům na konkrétním portu. Každý příchozí požadavek je zpracován na základě definovaného řetězu middlewarů a cest z vrchu dolů. To je důležité si uvědomit, protože to umožňuje řídit tok spouštění. Může být například více funkcí zpracovávající stejný požadavek a některé z těchto funkcí mohou být „uprostřed“ (odtud název middleware), jako např:

- Parsování informací z cookies, následně přejít na další krok (funkci) když je parsování dokončeno.
- Parsování parametrů z URL adresy, následně přejít na další krok, když je parsování dokončeno.
- Autorizace uživatele apod. [13]

Příklad jednoduché Express aplikace

Příklad „Ahoj světe“ pomocí Node.js a frameworku Express. Po instalaci a vytvoření spouštěcího souboru (např. index.js) pomocí *npm init* a *npm install express*:

Zahrneme express knihovnu

```
const express = require('express');
```

Vytvoříme aplikaci.

```
const app = express();
```

Vytvoříme konstantu PORT, podle toho, jaký port chceme, aby aplikace využívala. V příkladu je to port 3000.

```
const PORT = 3000;
```

Definujeme cestu (*/hello*), pro HTTP GET metodu. Druhým parametrem je tzv. request handler s alespoň dvěma parametry, a to *req* (request – požadavek) a *res* (response – odpověď).

```
app.get('/hello', (req,res) => {
    res.send("Ahoj světe");
});
```

V poslední řadě je třeba aplikaci spustit, popřípadě vypsat do console zprávu o spuštění aplikace.

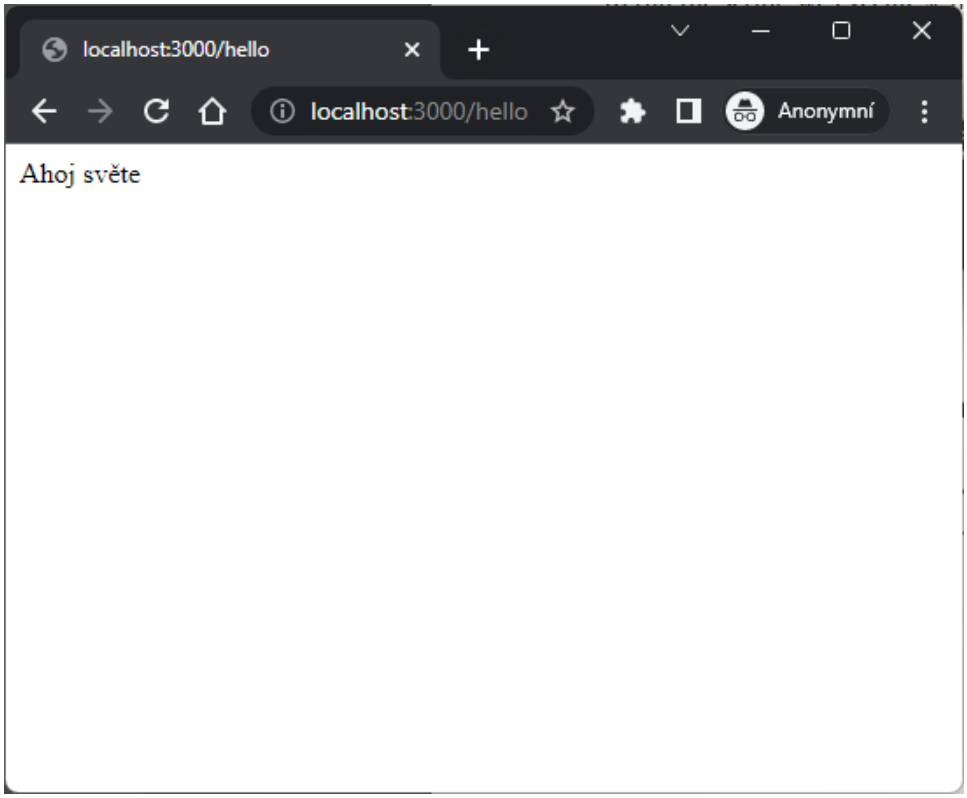
```
app.listen(PORT, (err) => {
    if(!err)
        console.log(`Server is running on http://localhost:${PORT}/`);
});
```

Samotný script pak spustíme v příkazovém řádku příkazem *node index.js* (případně jiný název pro hlavní soubor). Samotný výstup konzole pak vypadá následovně:

```
C:\Users\Tomáš\Desktop\test>node index.js
Server is running on http://localhost:3000/
```

Obr. 2 Výstup konzole
Zdroj: Vlastní zpracování

Po zadání adresy *http://localhost:3000/hello* do prohlížeče, se zobrazí výstup „Ahoj světe“.



Obr. 3 Otevřený prohlížeč na adrese http://localhost:3000/
Zdroj: Vlastní zpracování

4.3.7 React

React je open source JavaScriptová knihovna pro vytváření Single Page Aplikací (SPA). React byl vyvinut společností Facebook. Vydán byl v roce 2013 a od té doby se stal jednou z nejpopulárnějších knihoven pro tvorbu webových aplikací.

Klasické vs. SPA aplikace

Klasické webové aplikace fungují tak, že uživatel navštíví URL adresu a vyžádá si HTML soubor a všechny další přidružené soubory, jako jsou CSS a JavaScript z webového serveru. Po načtení stránky, může uživatel se stránkou interagovat. Každá další změna stránky znamená zopakování celého řetězce událostí znovu.

V moderním přístupu je pozornost upřena na klienta, kde za vykreslení celé aplikace je zodpovědný JavaScript na straně klienta. To funguje tak, že uživatel navštíví URL adresu a vyžádá si jeden malý HTML soubor a jeden přidružený soubor JavaScriptu. Po načtení může uživatel se stránkou začít interagovat. Při změně stránky není

potřeba vyžadovat žádné další soubory, použije se původně vyžádaný JavaScript k vykreslení nové stránky. [14]

Vlastnosti Reactu

Jednou z klíčových vlastností Reactu je jeho použití komponent. Komponenty jsou nezávislé části kódu, každá komponenta definuje svůj vzhled pomocí HTML, CSS a JavaScriptu, který dodává funkčnost interakcím. Komponenty pak lze použít v hierarchii komponent, k vytvoření celé aplikace. Výhodou je, že komponenty lze používat i na více místech v aplikaci. Název komponenty by měl vždy obsahovat počáteční velké písmeno, React tím rozlišuje, zda se jedná o HTML element (HTML elementy začínají malým písmenem), nebo o komponentu.

React komponenta sice vypadá, že je tvořena klasickým HTML, avšak jedná se o tzv. JSX (JavaScript XML), které umožňuje snadno kombinovat HTML a JavaScript.

```
import React from "react";

const Component = () => {
    const description = "První komponenta v Reactu";
    return (
        <main>
            <h1>Komponenta</h1>
            <p>{description}</p>
        </main>
    );
};

export default Component;
```

Obr. 4 Ukázka komponenty v Reactu
Zdroj: Vlastní zpracování

Že se nejedná o klasické HTML lze také poznat, že nelze použít některé atributy jako u klasického HTML, jako např. *class* nebo *for*. Jelikož se jedná o JavaScript, pouze zapsaný v jiné podobě, nelze použít tyto názvy, neboť jsou klíčovými slovy využívány samotným jazykem (definování třídy *class* a *for* cyklus). Proto byly tyto atributy nahrazeny atributy *className* a *htmlFor*. [14]

React využívá tzv. virtuální DOM (Document Object Model), jedná se o reálný DOM, který React udržuje v paměti. Tento virtuální DOM se používá k efektivnímu vykreslování změn v uživatelském rozhraní. Namísto toho, aby React neustále

aktualizoval celé rozhraní při každé změně, pouze aktualizuje změněné části virtuálního DOM. To funguje následovně:

1. Kdykoliv se změní stav aplikace, virtuální DOM a React vykreslí uživatelské rozhraní do virtuální reprezentace DOM.
2. React poté vypočítá rozdíl mezi oběma virtuálními reprezentacemi (předchozí a aktuální).
3. React aktualizuje pouze tu část, která je třeba aktualizovat ve skutečném DOM. [15]

4.3.8 MySQL

MySQL je open source relační databázový software, jak již název napovídá, využívá dotazovací jazyk SQL k manipulaci s daty a je jedním z nejpoužívanějších databázových softwarů na světě. Vyznačuje se svojí spolehlivostí, rychlostí a snadnému používání, a to jak u webových, tak desktopových platformách. MySQL je multiplatformní a nevyžaduje ke spuštění a běhu uživatelské rozhraní, lze ho používat v textovém režimu pomocí sady příkazů. Je však k dispozici i několik možností pro uživatelská rozhraní pro správu MySQL databází, jedním z nejpopulárnějších je webové rozhraní phpMyAdmin nebo např. desktopový program MySQL Workbench. [16]

MySQL umožňuje výběr z několika storage enginů, jedná se o moduly, které zajištují způsob ukládání a manipulace s daty v databázi. Každý má své výhody a nevýhody. Výchozím storage enginem v MySQL je InnoDB, který podporuje klíčové funkce pro moderní databáze, jako jsou transakce, zotavení z havárie, cizí klíče atd. Další možné storage enginy jsou např. MyISAM, což byl výchozí engine ve starších verzích, nepodporuje transakce ani cizí klíče, je však optimalizován pro kompresi a rychlost. Memory engine je pak vhodný pro extrémně rychlé operace s daty, je uložen v RAM a po restartu operačního systému jsou data ztracena. CSV (Comma Seperated Value) engine ukládá data v *.csv* formátu. [17]

4.4 Použité technologie

Níže je seznam technologií, které byly skutečně použity pro vývoj webové aplikace.

- Node.js
- Express
- React
- MySQL
- React

5 Návrh backendu

Backend aplikace byl vyvíjen jako REST API, pomocí návrhového vzoru Model View Controller (MVC) a pomocí webového frameworku Express v Node.js. Pro definici modelu byla použita knihovna Sequelize, jedná se o ORM (Objektově Relační Mapování), který umožňuje mapování relačních databází na objekty, a tím minimalizovat nutnost psaní SQL dotazů.

5.1 REST (Representational State Transfer)

REST je jedním z architektonických stylů vhodných pro vývoj webových služeb. Na rozdíl od jiných webových služeb, jako např. SOAP, REST není závislý na žádném protokolu, tedy pokud protokol podporuje URI (Uniform Resource Identifier) schéma, pro identifikaci zdroje. V případě naší aplikace se pohybujeme ve světě webových aplikací, URI schéma nám tedy poskytuje protokol HTTP. Jak již bylo řečeno jedná se o architektonický styl, a ne o standard, co by předepisoval soubor pravidel, kterým by bylo třeba se řídit k dosažení tzv. „RESTful architektury“. To vede k častému nevyužití plného potenciálu RESTu. Proto je vhodné dodržovat některé metody, není to však vyžadováno. Některé doporučené praktiky jsou:

- používání správných HTTP metod (GET,POST, PATCH, DELETE, …),
- používání správných HTTP stavových kódu (2xx, 3xx, 4xx, 5xx),
- unikátní URI pro každý zdroj,
- specifikovat formát odpovědi,
- definování verze pro zachování zpětné kompatibility. [18]

5.1.1 Zdroje

Zdroj je hlavní stavební kámen REST architektury. Vše, co může být pojmenováno může být zdroj (uživatel, výlet, obrázek atd.). Zdroj je abstrakcí čehokoliv, co může být dále upřesněno. Zdroje definují, čeho se budou služby týkat a jaké typy informací a s nimi související akce budou přenášeny. Zdroj se skládá z:

- reprezentace – jakým způsobem jsou data reprezentována (nejčastěji JSON, případně XML a další),
- identifikátor – URL adresa pro získání specifického zdroje,
- metadata – typ obsahu, poslední čas modifikace atd.,
- control data – pro cachování. [18]

5.1.2 Reprezentace

Jak již bylo zmíněno reprezentace zdroje může být v podstatě v libovolném formátu, pro účely aplikace byl použit formát JSON (JavaScript Object Notation). Jedná se o formát pro výměnu dat, je snadno čitelný a nezávislý na programovacím jazyce. JSON byl odvozen od JavaScript objektů, které si jsou velmi podobné. Proto je velmi výhodný při používání v JavaScriptu, kde jednoduše pomocí *JSON.parse()*, převedeme JSON textový řetězec na JavaScriptový objekt, a naopak pomocí *JSON.stringify()*.

JSON se skládá z klíčů a hodnot. Každý klíč je textový řetězec v uvozovkách a každá hodnota může být číslo, řetězec, logická hodnota, objekt nebo pole. Objekty jsou uzavřeny ve složených závorkách, pole potom v hranatých závorkách a samotné klíče jsou odděleny čárkou.

```
{
   "id":1,
   "name":"Výlet po okolí Černé hory",
   "description":"Cyklistický výlet po okolí černé hory",
   "user_participation":true,
   "trip_type":{
      "type":"Cyklistika"
   }
}
```

Obr. 5 Zjednodušený příklad výstupu výletu v JSON formátu
Zdroj: Vlastní zpracování

5.1.3 Identifikátor zdroje

Identifikátor zdroje by měl poskytnout unikátní způsob identifikace v jakémkoli čase, tzn., aby v různé časy nebyly pod daným identifikátorem různé zdroje. Nad zdroji by mělo být možné vykonávat akce, typicky na základě HTTP metod:

- GET – přístup ke zdroji v režimu pouze pro čtení,
- POST – odeslání nového zdroje na server (akce vytvoření),
- PUT – aktualizace zdroje (akce aktualizování – nahrazení celého zdroje),
- PATCH – aktualizace zdroje (akce aktualizování – částečná úprava zdroje),
- DELETE – smazání zdroje (akce smazání),
- a další. [18]

Příklad identifikátorů v aplikaci:

GET /api/v1/trips – Výpis všech výletů,
GET /api/v1/users – Výpis všech uživatelů.

Příklad akce

POST /api/v1/trips – Vytvoření nového výletu, výlet je součástí těla HTTP požadavku,
DELETE /api/v1/trips/1 – Odstranění výletu id 1,
PATCH /api/v1/trips/1 – Aktualizace výletu id 1, změny jsou součástí těla HTTP požadavku.

Příklad filtrování

GET /api/v1/trips?limit=30&order=createdAt.desc – Omezení na 30 výsledků, seřazeny sestupně podle data vytvoření.

5.1.4 Stavové kódy

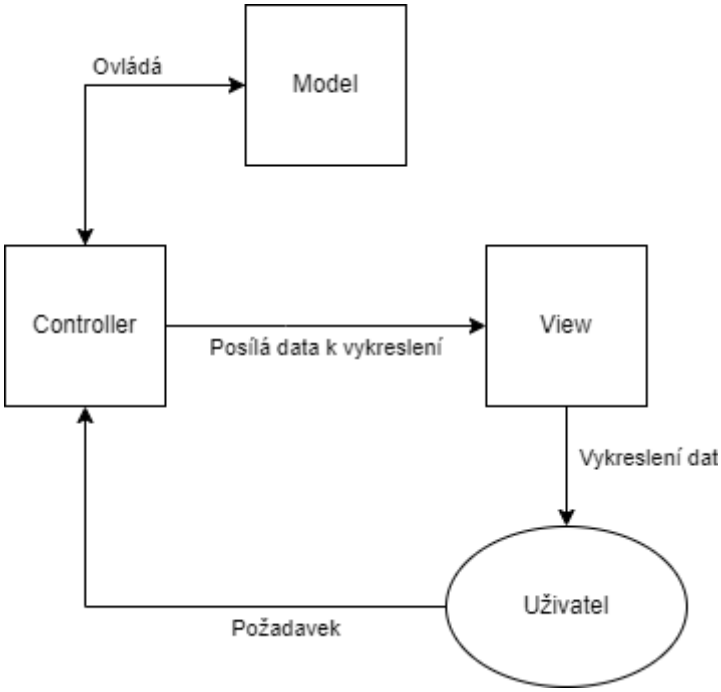
V případě protokolu HTTP, může REST těžit z využití HTTP stavových kódů, jedná se o číslo reprezentující stav výsledku požadavku. Stavové kódy pomáhají klientům určit stav odpovědi. Mezi nejpoužívanější stavové kódy se řadí např. 404 – stránka nenalezena, 400 – Bad request (špatný požadavek), 200 – Ok, 500 – Chyba serveru. Kódy jsou seskupeny do několika skupin podle významu:

- 1xx – informační,
- 2xx – úspěšné,
- 3xx – přesměrování,
- 4xx – chyba na straně klienta,
- 5xx – chyba na straně serveru. [18]

5.2 Model View Controller (MVC)

Aplikace byla vyvíjena podle návrhového vzoru MVC (Model View Controller), jedná se o způsob vývoje aplikací, kde je logika aplikace oddělena od prezentace dat. Což má za následek efektivnější vývoj a údržbu aplikace. Jedná se o jeden z nejpoužívanějších návrhových vzorů pro vývoj webových aplikací. MVC se skládá ze tří vrstev:

- **Model** – Zodpovědný za reprezentaci a (např. načítání a ukládání z a do databáze) a poskytuje je dalším částem aplikace.
- **View** – Zodpovědný za prezentaci dat uživateli, zobrazuje data z modelu a nabízí uživateli prostředky umožňující interakci s aplikací (tlačítka, formuláře apod.), má formu uživatelského rozhraní.
- **Controller** – Model sám o sobě neví, kdy a jaká data má zpracovávat a view, jaká data má zobrazit. Proto existuje controller, který zpracovává uživatelské požadavky a řídí interakci mezi modelem a view vrstvou, to znamená, že controller ovládá model pro získání dat a výsledek předá view vrstvě pro zobrazení. [18]



Obr. 6 Komunikace mezi vrstvami MVC modelu
Zdroj: Vlastní zpracování podle Fernanda Doglio

Jelikož webová aplikace pro plánování výletů byla vyvíjena jako REST API, view vrstva zde v tradičním smyslu neexistuje. Místo toho jsou data vrácena klientovi pomocí jiného formátu, v případě této aplikace ve formátu JSON objektu, který obsahuje veškerá data potřebná pro zobrazení frontendovou aplikací. Můžeme se tedy dívat na problém tak, že backendová část aplikace obsahuje vrstvy Model a Controller, a vrstva View je reprezentována jinou aplikací, která je zodpovědná za zobrazení dat z REST API, v našem případě pomocí React aplikace.

5.2.1 Příklad MVC v rámci aplikace

Příklad MVC pomocí frameworku Express a Sequelize, pozn.: struktura, model a kód byl pro příklad výrazně zestručněn.

Struktura části aplikace

```
controller
├── trip.controller.js
├── user.controller.js
├── comment.controller.js
└── ...
model
├── trip.model.js
├── user.model.js
├── comment.model.js
└── ...
routes
├── trip.router.js
├── user.router.js
├── comment.router.js
└── ...
index.js
```

Adresář routes obsahuje veškeré endpointy, přes které může klient komunikovat s controllerem, *index.js* je pak hlavní spouštěcí soubor aplikace

Příklad části modelu definovaný pomocí Sequelize,

```
module.exports = (sequelize, DataTypes) => {
    const Trip = sequelize.define("trips",{
        id: {
            type: DataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        name: {
            type: DataTypes.STRING(64),
            allowNull: false
        },
        description: {
            type: DataTypes.STRING(500),
            allowNull: true
        },
    })
    return Trip;
};
```

Obr. 7 Část modelu trip.model.js
Zdroj: Vlastní zpracování

Příklad části controlleru,

```
const db = require("../models"); //Načtení db objektu s definovanými modely
const Trip = db.trip; //Uložení modelu Trip z objeku db do samostatné konstanty

module.exports = {
    createTrip: async (req, res) => {
        /*
           validace
        */

        //Načtení názvu a popisu výletu z těla požadavku
        const { name, description } = req.body;
        //Try catch block v případě vyskytnutí chyby při komunikaci s databází
        try {
            //Vytvoření výletu a uložení do konstanty (komunikace s modelem)
            const trip = await Trip.create({ name, description });
            //Odeslání odpovědi klientovi
            res.json({ success: true, trip });
        } catch (e) {
            //Odeslání statusu 500 (Server Error) a zprávy s chybou
            res.status(500).json({ success: false, message: e.message });
        }
    },
    getTrips: async (req, res) => {/* Kód pro získání výletů */},
    updateTrip: async (req, res) => {/* Kód pro aktualizaci výletu */},
    deleteTrip: async (req, res) => {/* Kód pro smazání výletu */}
};
```

Obr. 8 Příklad části controlleru trip.controller.js
Zdroj: Vlastní zpracování

Příklad části routeru

```
const trip = require("../controllers/trip.controller"); //Načtení controlleru

module.exports = (app) => {
    //Vytvoření express routeru
    const router = require("express").Router();

    //Definování endpointu pro vytvoření výletu (metoda POST)
    router.post("/", trip.createTrip);
    /*
        Definování endpointu pro získání všech výletů nebo konkrétního výletu
        podle volitelného parametru tripId (metoda GET)
    */
    router.get("/:tripId?", trip.getTrips);

    router
        .route("/:tripId") //Definování endpointu s parametrem tripId
        .patch(trip.updateTrip) //s metodou PATCH pro aktualizaci výletu
        .delete(trip.deleteTrip); //s metodou DELETE pro smazání výletu

    /*
        Přiřazení routeru k aplikaci, na endpoint /trips
        tzn. všechny endpointy routeru budou mít předponu /trips
        (/trips, /trips/:tripId)
    */
    app.use("/trips", router);
};
```

Obr. 9 Příklad části routeru trip.router.js
Zdroj: Vlastní zpracování

6 Návrh databáze

Jak již bylo zmíněno v předchozí kapitole, pro komunikaci s databází byl použit ORM (Objektově Relační Mapování) Sequelize pro Node.js, který zajišťuje mapování relačních tabulek na JavaScriptové objekty. Sequelize nabízí podporu pro několik typů relačních databází, jako jsou PostgreSQL, Oracle, Micorosft SQL Server, MySQL a další.

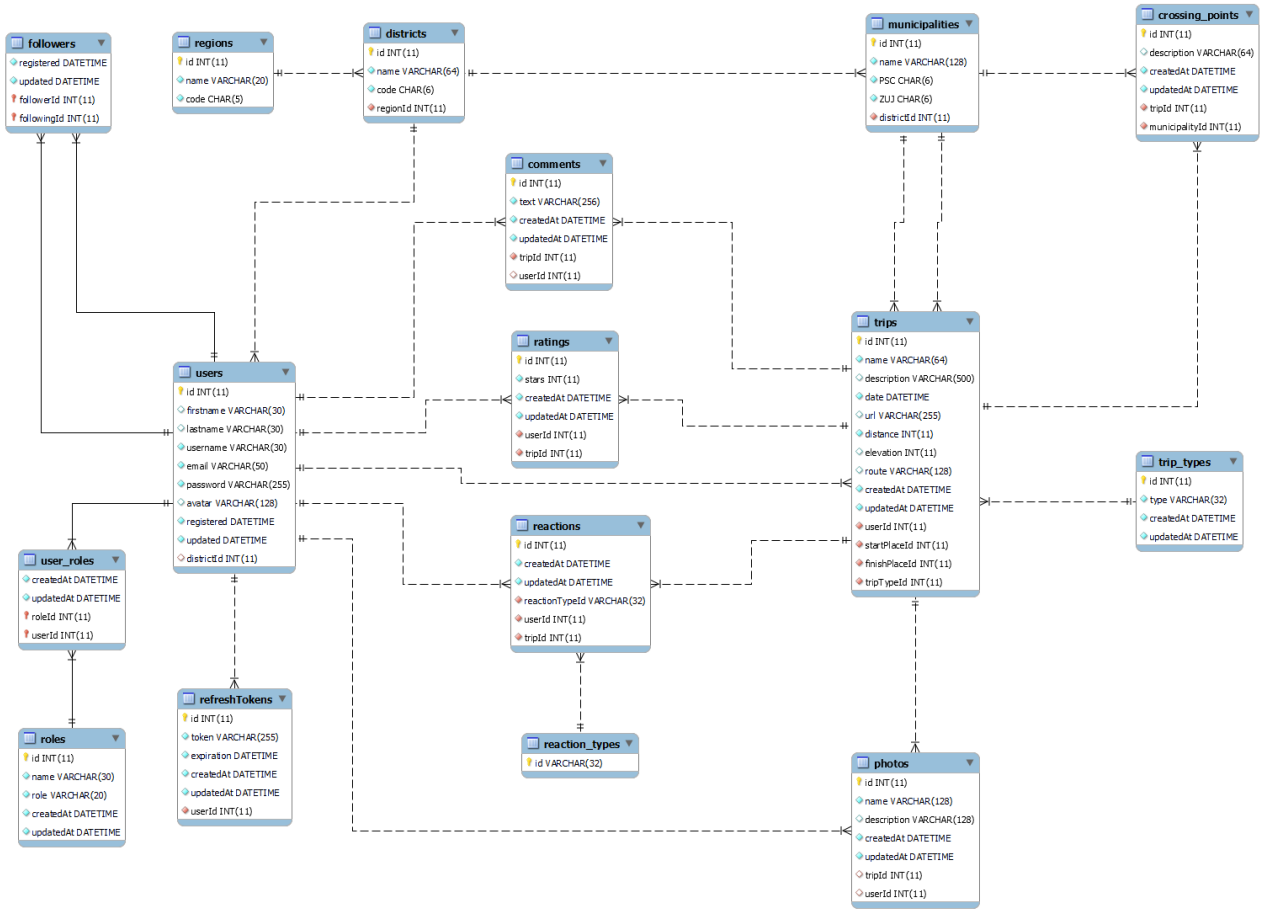
6.1 Připojení k databázi

Připojení k databázi probíhá vytvořením instance Sequelize s parametry: název databáze, název uživatele, heslo uživatele a objektem s nastavením, který obsahuje adresu databázového serveru, dialect určuje, o jaký databázový systém se jedná (v případě naší aplikace se jedná o MySQL) a další nastavení. V příkladu níže (Obr. 10) jsou hodnoty potřebné pro připojení načítány z proměnného prostředí, jehož obsah se bude lišit podle prostředí, ve kterém bude aplikace spuštěna. Pozn.: kód byl zjednodušen a neobsahuje všechny modely a vztahy. Samotné vztahy jsou pak popsány v samostatné kapitole 6.3.

```
const Sequelize = require("sequelize");

const sequelize = new Sequelize(
    process.env.DB_DATABASE, //název databáze
    process.env.DB_USER, //název uživatele
    process.env.DB_PASSWORD, //heslo uživatele
    //Objekt s nastavením
    {
        host: process.env.DB_HOST, //adresa serveru
        dialect: "mysql" //dialect
    }
);

//Definice objektu, přes který se později přistupuje k modelům případně instanci sequelize
const db = {};

//Importování modelů do objektu db
db.user = require("../models/user.model")(sequelize, Sequelize.DataTypes);
db.trip = require("../models/trip.model")(sequelize, Sequelize.DataTypes);
db.municipality = require("../models/municipality.model")(sequelize,Sequelize.DataTypes);
//... další importy modelů

//... definování vztahů mezi modely

//Přidání instance sequelize do objektu db
db.sequelize = sequelize;

//Exportování objektu db
module.exports = db;
```

Obr. 10 Připojení k databázi a import modelů
Zdroj: Vlastní zpracování

6.2 ER Diagram



Obr. 11 ER Diagram
Zdroj: Vlastní zpracování

6.3 Vztahy

Vztahy zajišťují v databázi referenční integritu (způsob zaručení konzistence dat). V databázi učené pro aplikaci jsou definovány následující vztahy:

6.3.1 Vztahy 1:N

Vztah 1:N je takový vztah, kde jednomu záznamu může náležet více záznamů (0-N) v druhé (propojené tabulce).

1:N Vztahy mezi entitami v aplikaci:

- Každý uživatel může mít mnoho refresh tokenů (0-N).
- Každý refresh token náleží právě jednomu uživateli.

- Každý kraj má mnoho okresů (0-N).
- Každý okres náleží právě do jednoho kraje.

- Každý okres má mnoho obcí (0-N).
- Každá obec náleží právě do jednoho okresu.

- Každý uživatel může vytvořit mnoho výletů (0-N).
- Každý výlet náleží právě jednomu uživateli.

- Každá obec může být u mnoha výletů v roli startovací pozice (0-N).
- Každý výlet náleží právě jedné obci v roli startovní pozice.

- Každá obec může být u mnoha výletů v roli cílová pozice (0-N).
- Každý výlet náleží právě jedné obci v roli cílová pozice.

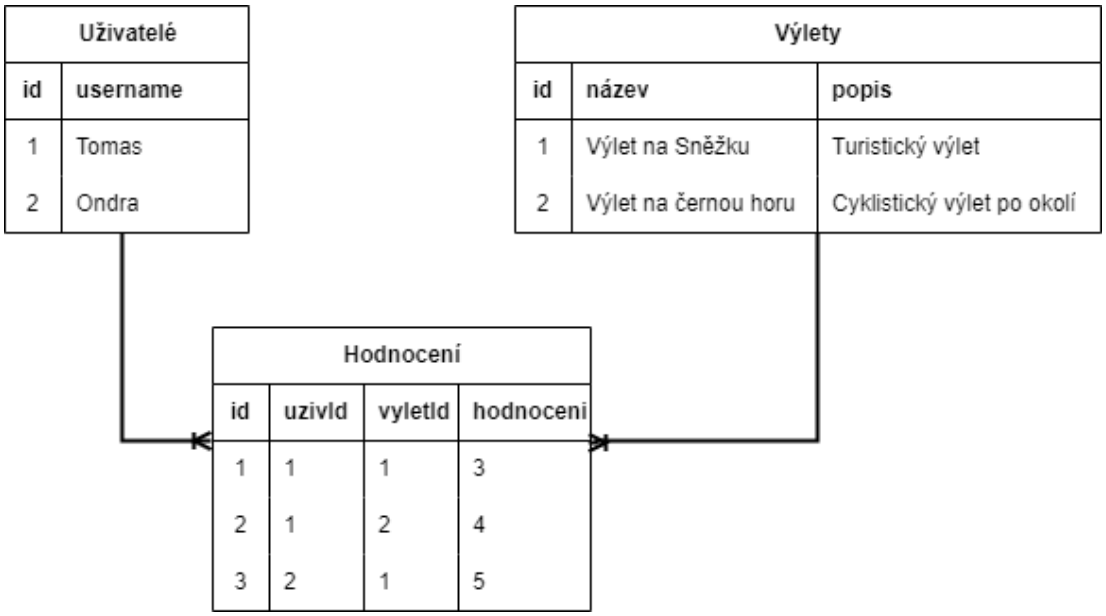
- Každý typ výletu má mnoho výletů (0-N).
- Každý výlet má právě jeden typ výletu.

- Každý typ reakce náleží mnoha reakcím (0-N).
- Každá reakce má právě jeden typ reakce.

- Každý okres může mít mnoho uživatelů (0-N).
- Každý uživatel může mít právě jeden okres (0-1).

6.3.2 Vztahy N:M

Vztah N:M je takový vztah, kde záznamu jedné tabulky, může náležet více záznamů druhé tabulky, a naopak jednomu záznamu z druhé tabulky, může náležet více záznamů z první tabulky. Tento vztah se realizuje pomocí tzv. spojovací tabulky, která vztah rozdělí na dva vztahy 1:N a M:1. Příklad níže (Obr. 12) znázorňuje vztah, kde uživatel může ohodnotit více výletů a výlet může být ohodnocen více uživateli.



Obr. 12 Znázornění N:M vztahu
Zdroj: Vlastní zpracování

N:M Vztahy mezi entitami v aplikaci:

- Každý uživatel může mít mnoho rolí (0-N).
- Každou roli může mít více uživatelů (0-N).

- Každý výlet může mít mnoho obcí v roli průchozího bodu (0-N).
- Každá obec může být obsažena u mnoha výletů v roli průchozího bodu (0-N).

- Každý uživatel může udělit hodnocení několika výletům (0-N).
- Každý výlet může být ohodnocen několika uživateli (0-N).

- Každý uživatel může napsat komentáře k několika výletům (0-N).
- Každý výlet může být okomentován několika uživateli (0-N).

- Každý uživatel může sledovat mnoho uživatelů (0-N).

- Každý uživatel může být sledován mnoha uživateli (0-N).

- Každý uživatel může reagovat na několik výletů (0-N).
- Každá výlet může mít reakce od různých uživatelů (0-N).

- Každý uživatel může nahrát fotografie k různým výletům (0-N).
- Každý výlet může mít mnoho fotografií od různých uživatelů (0-N).

6.3.3 Definice vztahů v Sequelize

Příklad níže (Obr. 13) znázorňuje definování 1:N vztahu: „uživatel může vytvořit více výletů“ a „výlet musí mít právě jednoho uživatele“ a vztah: „výlet musí mít právě jednu obec v roli startovní pozice“ a „obec může být v roli startovní pozice u více výletů“.

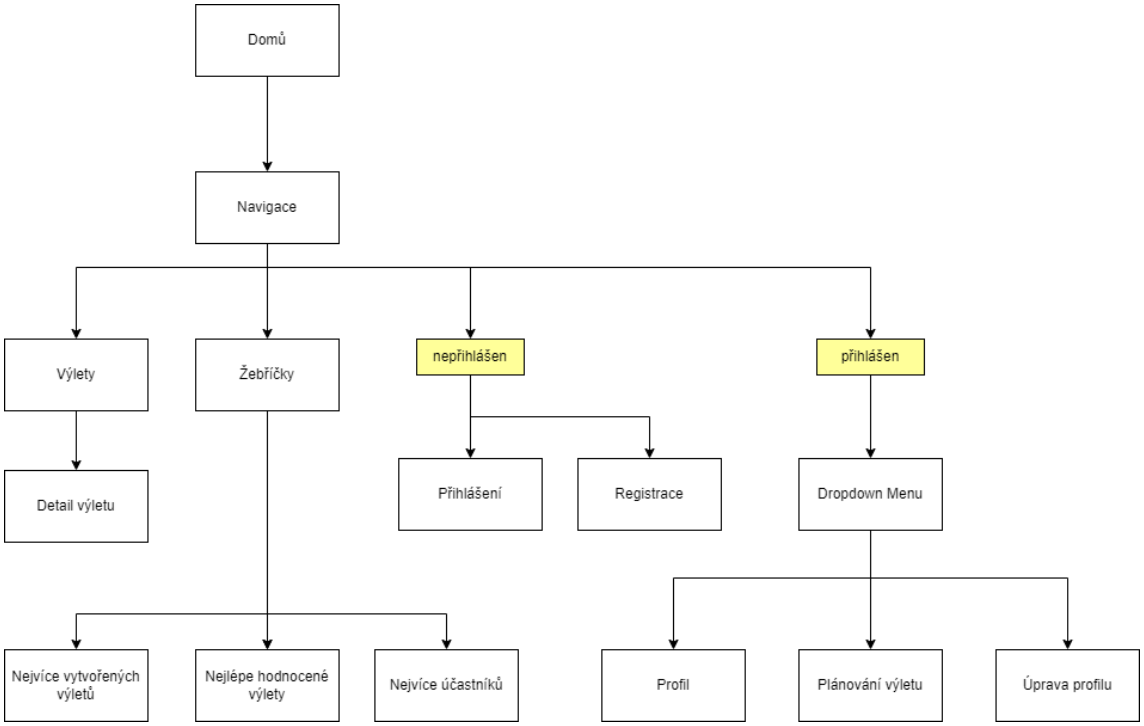
```
//Výlet může být vytvořen 1 uživatelem
db.trip.belongsTo(db.user);
//Uživatel může vytvořit 0-N výletů
db.user.hasMany(db.trip, {
    //V případě smazání uživatele, dojde ke smazání výletů daného uživatele
    onDelete: "CASCADE",
    foreignKey: { allowNull: false /*NOT NULL*/ },
});

//Obec má 1-N výletů v roli startovní pozice
db.municipality.hasMany(db.trip, {
    as: "startPlace", //role
    foreignKey: {
        name: "startPlaceId", //Název cizího klíče
        allowNull: false, //NOT NULL
    },
});

//Výlet má jednu obec v roli startovní pozice
db.trip.belongsTo(db.municipality, {
    as: "startPlace",
    foreignKey: {
        name: "startPlaceId", //Název cizího klíče
        allowNull: false, //NOT NULL
    },
});

//... další definice
```

Obr. 13 Definování vztahů pomocí Sequelize
Zdroj: Vlastní zpracování

7 Návrh frontendu

Jak již bylo naznačeno v předchozích kapitolách, frontendová část aplikace byla vyvíjena jako Single Page Aplikace (SPA) v JavaScriptovém frameworku React.js. Spolu s Reactem bylo využito několik knihoven třetích stran, pro usnadnění vývoje.

7.1 Mapa stránek



Obr. 14 Mapa stránek
Zdroj: Vlastní zpracování

7.2 Struktura projektu

Projekt byl rozdělen do 4 částí:

- **App** – obsahuje konfigurační soubory pro definování hlavního API slice pro RTK query a definování jednotného storu pro React Redux.
- **Components** – obsahuje znovupoužitelné komponenty napříč aplikací.
- **Features** – obsahuje definice redux reducerů, API slice pro jednotlivé funkce aplikace a samotné komponenty s obsahem stránky.
- **Hooks** – obsahuje vlastní hooky.

7.3 Redux Toolkit

Redux je jedna z nejpoužívanějších knihoven pro správu stavu React aplikace. K pochopení problematiky je vhodné vysvětlit jednotlivé stavy, které se v React aplikaci vyskytují.

7.3.1 Stavy v Reactu

Local state (Lokální stav) – je stav, který se využívá právě v jedné komponentě, typicky se využívá hook useState, který je součástí Reactu. Zápis useState vypadá následovně:

```
const [count, setCount] = useState(0);
```

z funkce useState destrukturalizací získáme stavovou proměnnou (v tomto případě count) a funkci pro změnu stavové proměnné (v tomto případě setCount). Parametr u funkce useState nastavuje výchozí hodnotu. Lze použít i jiné názvy, ale konvencí je vždy pro funkci pro změnu stavu používat set a název stavové proměnné. Typické použití lokálního stavu je například provázání se vstupy formuláře.

```
<input value={username} id="username" onChange={(e) => setUsername(e.target.value)}/>
```

Cross-component state (napříč komponenty) – je stav, který ovlivňuje několik komponent, v tomto případě několik komponent sdílí stav. Toho je docíleno pomocí tzv. prop drilling, kdy se stav předává z rodičovské komponenty na komponentu potomka. Typické využití je např. při otevírání a zavírání modálního okna.

App-wide-state (stav celé aplikace) – Ovlivňuje celou aplikaci, typickým příkladem je status přihlášení, kde je třeba informaci o přihlášeném uživateli držet v rámci celé aplikace. V tomto případě by využití prop drilling mělo za následek předávání stavu napříč celou hierarchií komponent, což by bylo značně nepřehledné a mohlo by zvýšit i pravděpodobnost výskytu chyb. Proto se využívají knihovny, jako je právě redux toolkit, který udržuje centrální úložiště stavů (tzv. store), ke kterému se mohou komponenty přihlásit a stav získat, případně změnit. V případě změny stavu, získají přihlášené komponenty k odběru dat oznámení o změně, a mohou tak aktualizovat své uživatelské rozhraní. [19]

7.3.2 Princip fungování

Základní princip fungování je postaven na konceptech reduktorů (reducers) a akcích (actions):

- **Reducers** – funkce zodpovědná za aktualizaci úložiště (store). Reducer přijímá dva parametry, současný stav a akci. Nutno podotknout, že reducer by měl být čistou funkcí (neměl by měnit vstupní argumenty). Komponenty pro aktualizaci storu musí vždy použít reducer. Reducer funkce musí být vždy spuštěna komponentou (např. kliknutím na tlačítko), k tomu využívá právě konceptů akcí.
- **Actions (akce)** – Komponenty odesílají akce, jedná se o JavaScriptové objekty, které obsahují povinný atribut type, který určuje typ akce a popis změny, která se má provést ve stavu aplikace a volitelně také další data (payload). Redux předá tyto akce reduceru, ten podle popisu vytvoří nový stav, kterým nahradí existující stav ve storu. Komponenty přihlášené ke storu pak získají oznámení o změně stavu a mohou aktualizovat své uživatelské rozhraní. [19]

Příklad aktualizace stavu ve store pomocí reduceru

Tento reducer je zodpovědný za inkrementaci či dekrementaci o jedničku stavu count.

```
const counterReducer = (state = { count: 0 }, action) => {
    switch (action.type) {
        case "increment":
            return { count: state.count + 1 };
        case "decrement":
            return { count: state.count - 1 };
        default:
            return state;
    }
};

export const store = configureStore({
    reducer: {
        counter: counterReducer,
    },
});
```

Obr. 15 Ukázka reduceru a vytvoření redux store
Zdroj: Vlastní zpracování

Po vytvoření redux store je třeba v hlavním souboru (typicky *index.js*) obalit komponentu App do komponenty *<Provider>* a předat vytvořený store.

```
import { store } from "./app/store";
import { Provider } from "react-redux";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <Provider store={store}>
        <App />
    </Provider>
  </React.StrictMode>
);
```

Použití v samotné komponentě vypadá pak následovně, výsledkem je tlačítko, které po kliknutí odešle akci pro inkrementaci a výsledný stav je vypsán jako text tlačítka:

```
import React from "react";
import { useDispatch, useSelector } from "react-redux";

const Test = () => {
    //získání stavu z reduceru counter z redux storu
    const counter = useSelector((state) => state.counter);
    //funkce pro odeslání akce
    const dispatch = useDispatch();

    //akce pro přičtení
    const incrementAction = () => ({ type: "increment" });

    //funkce pro obsloužení kliknutí na tlačítko
    const handeClick = () => {
        //odeslání funkce pro přičtení
        dispatch(incrementAction());
    };

    return (
        <div>
            <button onClick={handeClick}>
                {counter.count /* Vypsání aktuálního stavu */}
            </button>
        </div>
    );
};

export default Test;
```

Obr. 16 Ukázka získání a změny stavu v komponentě
Zdroj: Vlastní zpracování

7.3.3 RTK Query

Samotný redux toolkit byl použit především k udržení stavu přihlášeného uživatele v rámci aplikace a ovládání modálního okna pro registraci a přihlášení. Hlavním důvodem bylo ale využití RTK Query, což je část redux toolkitu umožňující volání API endpointů a zahrnuje další funkce, jako např. cachování do redux storu apod.

Při vývoji aplikace bylo použito jednotné API odkazující na backendovou část aplikace pomocí createApi, v createApi se definuje tzv. baseQuery, která specifikuje např. základní tzv. baseUrl, která je identická pro všechny následně definované endpointy. Z důvodu přehlednosti nebyly samotné endpointy v aplikaci definovány přímo v rámci createApi, ale byly vkládány (injektovány) v samostatných souborech (tzv. slicech). [20]

Níže (Obr. 17) je znázorněno, jak je definováno API v samotné aplikaci, bylo zde ale vypuštěno řešení reautentizace, ta je znázorněna v kapitole 8.2.1(Obr. 21Obr. 20 Rozkódovaný JSON token).

```
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

const baseQuery = fetchBaseQuery({
    //Definování základní cesty k api (načteno z proměnného prostředí)
    baseUrl: process.env.REACT_APP_API_URL,
    //Zahrnutí cookies a autentizační hlavičky v požadavcích
    credentials: "include",
    prepareHeaders: (headers, { getState }) => {
        //Získání tokenu z redux store
        const token = getState().auth.token;
        if (token) {
            //pokud token existuje nastaví se do hlavičky
            headers.set("authorization", "Bearer " + token);
        }
        return headers;
    },
});

export const apiSlice = createApi({
    baseQuery: baseQuery,
    endpoints: (builder) => ({}),
});
```

Obr. 17 Ukázka definování API pomocí RTK Query
Zdroj: Vlastní zpracování

V aplikaci byly využity 2 typy endpointů, a to query a mutation. Query je určen pro dotazování se na data z API endpointu (metoda GET) a mutation pro změnu (mutaci) dat na API endpointu (metody POST, PUT, DELETE atd.). [20] Jak již bylo zmíněno samotné endpointy byly injektovány z jiných souborů. Níže (Obr. 18) je znázorněn apiSlice, který je zodpovědný za autorizaci, a tudíž obsahuje definované všechny endpointy týkající se registrace a přihlášení.

```
import { apiSlice } from "../../app/api/apiSlice";

export const authApiSlice = apiSlice.injectEndpoints({
    endpoints: (builder) => ({
        login: builder.mutation({ //mutation
            query: (credentials) => ({ //parametr obsahuje předaná data (username, heslo)
                url: "/auth/login", //Cesta (baseQuery + url)
                method: "POST", //Metoda
                body: { ...credentials }, //zahrnutí předaných dat do těla požadavku
            }),
        }),
        register: builder.mutation({
            query: (credentials) => ({
                url: "/auth/register",
                method: "POST",
                body: { ...credentials },
            }),
        }),
    }),
});

//Vyexportování vygenerovaných hooků pro volání endpointů
export const {useLoginMutation, useRegisterMutation,} = authApiSlice;
```

Obr. 18 Příklad definování endpointů
Zdroj: Vlastní zpracování

Použití v samotných komponentech pak vypadá následovně:

```
import { useLoginMutation } from "./authApiSlice";
import { useDispatch } from "react-redux";
import { setCredentials} from "./authSlice";

const LoginModal = () => {
    //získání metody login z importované useLoginMutation
    const [login] = useLoginMutation();
    const dispatch = useDispatch();

    const handleSubmit = async (values, { resetForm }) => {
        try {
            //odeslání požadavku na přihlášení a uložení odpovědi do userData
            const userData = await login(values).unwrap();

            //Uložení tokenu a uživatelských informací z odpovědi do redux storu
            dispatch(
                setCredentials({
                    token: userData.accessToken,
                    user: userData.user
                })
            );
            resetForm({ values: "" });
        } catch (e) {
            //Zachycení chyby
            setErrMsg(e.message);
        }
    };
    return; //Zde by se vracel samotný formulář
}
```

Obr. 19 Příklad použití v komponentě LoginModal
Zdroj: Vlastní zpracování

7.4 Formik

Další významnou knihovnou použitou při vývoji frontendové části byla knihovna Formik, ta se zaměřuje na usnadnění práce s formuláři. Umožňuje např. snadnou validaci podle definovaného schématu.

7.5 Bootstrap

Bootstrap je CSS framework, který nabízí předdefinované vzhledy pro velké množství komponent, jako jsou tlačítka, formuláře, navigační panely, modalová okna, seznamy a další. Především ale umožňuje snadné rozvržení obsahu na stránce (layoutu), díky jeho silnému grid systému, kdy celá stránka je rozdělena do 12 částí a jednotlivé sekce obsahu lze různě roztahovat přes těchto 12 částí. Např. hlavní obsah se může rozpínat přes 8 částí a vedlejší panel přes zbylé 4 části. To lze měnit i na základě velikosti zobrazovacího zařízení a to pomocí tzv. breakpointů (xs, sm, md, lg, xl, xxl), což zajištuje kompletně responsivní webové stránky. [21]

8 Uživatelské účty

Kapitola pojednává o tom, jak se v aplikaci přistupuje k uživatelským účtům, jaké obsahuje role, bezpečnostní požadavky a způsob komunikace mezi frontendovou a backendovou částí aplikace.

8.1 Přístup k uživatelům

V aplikaci je rozlišen uživatel dvěma základními způsoby, a to, zda je přihlášen, nebo nepřihlášen. Nepřihlášený uživatel má omezené možnosti, jako např. nemožnost reagovat na výlety, psát komentáře, přidávat fotografie, hodnotit výlety atd. Přihlášený uživatel může mít přiděleny 3 role:

- **User** – tuto roli získá každý uživatel automaticky při registraci.
- **Admin** – má možnost upravovat a mazat výlety, komentáře, fotografie jiných uživatelů. Může také upravovat uživatelské profily, ale nemůže nastavovat role.
- **Hl. Admin** – role hlavní admin umožňuje přidělovat uživatelům uživatelské role.

Předpokládá se, že první registrovaný uživatel je hlavní administrátor, tudíž se mu nastaví obě administrátorské role automaticky. Pro registraci účtu je vyžadován e-mail a přístupové heslo, které musí splňovat podmínky:

- musí mít alespoň 8 znaků,
- musí obsahovat alespoň jedno velké písmeno a jednu číslici.

Všechna hesla jsou ukládána v databázi v hashované podobě, a to pomocí hashovací funkce Bcrypt.

8.2 JSON Web Token (JWT)

Jedním z principů RESTU je bezstavovost. JWT je pak způsob bez stavového řešení autentizace (ověření uživatele), to znamená, že JWT není uložen na serveru, jako v případě např. session base přihlašování. JWT token si udržuje klient (např. v cookies, či local storage nebo nejlépe ve stavu aplikace). JWT se pak posílá v hlavičce s každým požadavkem na REST API. [22]

JWT kóduje data do JSON objektu a skládá se ze tří částí:

- **Hlavička** – obsahuje informaci o algoritmu a typu tokenu.
- **Payload (data)** – obsahuje samotná data, která jsou v tokenu uložena (např. ID uživatele, role atd.) a informaci o datu vytvoření a expirace (vypršení) tokenu.
- **Podpis** – obsahuje zakódovanou část hlavičky, dat a „secret“, což může být libovolný řetězec znaků, který by měl být bezpečně uložen na serveru, aby nedošlo k odcizení. [22]

Jednotlivé části tokenu se oddělují tečkou. Podoba JWT pak vypadá následovně:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjE0LCJ1c2VybmFtZSI6IlRlc3QiLCJlbWFpbCI6InRlc3RAdGVzdC5jeiIsInJvbGVzIjpbIlVTRVIiLCJBRE1JTiIsIkhMX0FETUlOIl0sImlhdCI6MTY4OTg0Mjk1NiwiZXhwIjoxNjg5ODQzODU2fQ.u_uqlsx0MKDleALhi6cj530mdQQXDr89ko0lyoPeUN0
```

Rozkódovaná podoba tohoto tokenu pak vypadá následovně:

```
//Hlavička
{
  "alg": "HS256",
  "typ": "JWT"
}
//Payload (data)
{
  "userId": 14,
  "username": "Test",
  "email": "test@test.cz",
  "roles": [],
  "iat": 1689842956,
  "exp": 1689843856
}
//Podpis
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Obr. 20 Rozkódovaný JSON token
Zdroj: Dekódováný JWT token pomocí jwt.io

8.2.1 Access a refresh Token

Access token (přístupový token) je token, který se posílá s každým požadavkem a používá se pro autentizaci uživatele. Z bezpečnostních důvodu by měl mít access token omezenou dobu platnosti (v řádech minut). K obnovení přístupu se pak používají refresh tokeny (ty mají obvykle platnost delší – v řádech až dní). [23]

V případě této aplikace, bylo využito jak access, tak refresh tokenů. Access token je uložen ve stavu aplikace a refresh token v HTTP only cookies, to znamená, že nemůže být získán pomocí JavaScriptu.

Princip fungování v aplikaci:

1. Uživatel navštíví webovou stránku a má v cookies uložen refresh token.
2. Refresh token se pošle s požadavkem na API.
3. API ověří platnost a vrátí access token.
4. Klient uloží access token do stavu aplikace.
5. Klient posílá access token s každým požadavkem.
6. API ověří jeho platnost a pošle odpověď.
7. V případě vypršení platností, API vrátí odpověď s kódem 401.

8. Klient pošle požadavek na obnovení access tokenu (spolu s refresh tokenem).
9. API ověří platnost refresh tokenu a vrátí nový access token.
10. Klient zopakuje předchozí požadavek spolu s novým access tokenem.
11. V případě vypršení refresh tokenu bude uživatel odhlášen.

```
import { Mutex } from "async-mutex";
import { logOut, setCredentials } from "../../features/auth/authSlice";

//Zajišťuje ochranu před posíláním mnoha neautorizovaným požadavkům
const mutex = new Mutex();

const baseQueryWithReauth = async (args, api, extraOptions) => {
    await mutex.waitForUnlock();
    //získá výsledek ze zadaného požadavku
    let result = await baseQuery(args, api, extraOptions);

    //Pokud vrátí error se statusem 401 (Neautorizován)
    if (result?.error?.status === 401) {
        //pokud není mutex zamknutý
        if (!mutex.isLocked()) {
            //zamkne mutex
            const release = await mutex.acquire();
            try {
                //posílá požadavek na obnovu access tokenu
                const refreshResult = await baseQuery(
                    { url: "/auth/refreshToken", method: "POST" },
                    api, extraOptions
                );
                if (refreshResult.data) {
                    //pokud se vrátí výsledek aktualizuje access token
                    //a informaci o uživateli v redux store
                    const { accessToken, user } = refreshResult.data;
                    api.dispatch(setCredentials({ token: accessToken, user }));
                } else {
                    //pokud požadavek nevrátí žádná data, odhlásí uživatele
                    api.dispatch(logOut());
                }
                //Zopakuje původní požadavek s novým access tokenem nebo bez tokenu
                result = await baseQuery(args, api, extraOptions);
            } finally {
                //uvolní mutex po dokončení požadavku
                release();
            }
        } else {
            //pokud je mutex zamčený čeká na odemčení
            await mutex.waitForUnlock();
            result = await baseQuery(args, api, extraOptions);
        }
    }
    return result;
};

export const apiSlice = createApi({
    baseQuery: baseQueryWithReauth,
    endpoints: (builder) => ({}),
});
```

Obr. 21 Ukázka kódu reautentizace ve frontendové části aplikace
Zdroj: Vlastní zpracování

8.2.2 Refresh token rotation

Refresh token rotation je způsob ochrany proti zneužití refresh tokenu. V případě odcizení refresh tokenu by měl útočník možnost nekonečného generování nových access tokenů. Refresh token rotation garantuje při každém vygenerování nového access tokenu vygenerování i nového refresh tokenu. Tím se zamezí generování nových access tokenů pomocí jednoho refresh tokenu. [23]

9 Ovládání aplikace

V této kapitole bude popsáno ovládání aplikace, doplněno o ukázky v podobě obrázků ze samotné aplikace a vysvětlením, kde se dané prvky na stránce nachází.

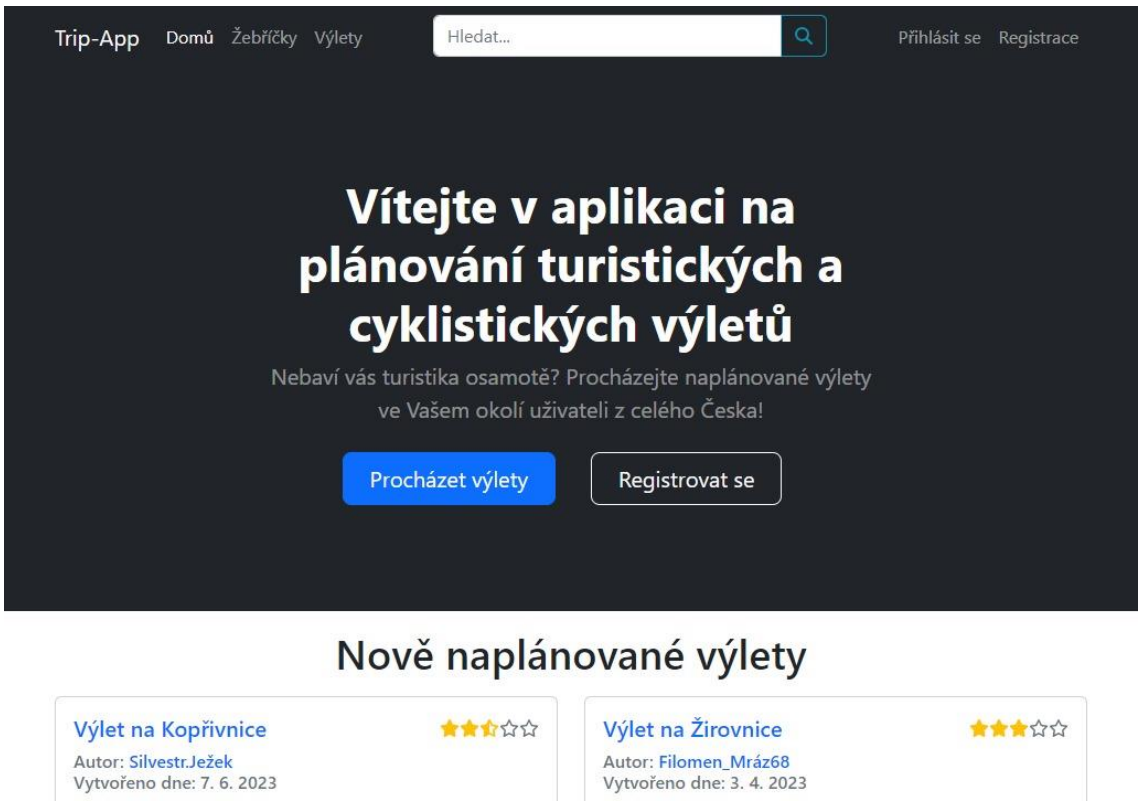
9.1 Navigační panel

Navigační panel je společný pro všechny stránky, má dvě varianty, a to pro přihlášeného a nepřihlášeného uživatele. Nepřihlášený uživatel má na pravé straně možnost přihlášení či registrace, zatímco přihlášený uživatel vidí rozbalovací nabídku se svým uživatelským jménem, kde se nachází další možnosti, jako odkaz na svůj profil, naplánování nového výletu, úpravu profilu a odhlášení. Každý uživatel (i nepřihlášený) má pak přístup k domovské stránce, žebříčkům, procházení výletů a vyhledávání výletů či uživatelů podle názvu nebo jména. Rozdíl mezi oběma variantami je vidět na obrázku níže (Obr. 22).

Obr. 22 Navigační panely
Zdroj: Vlastní zpracování

9.2 Domovská stránka

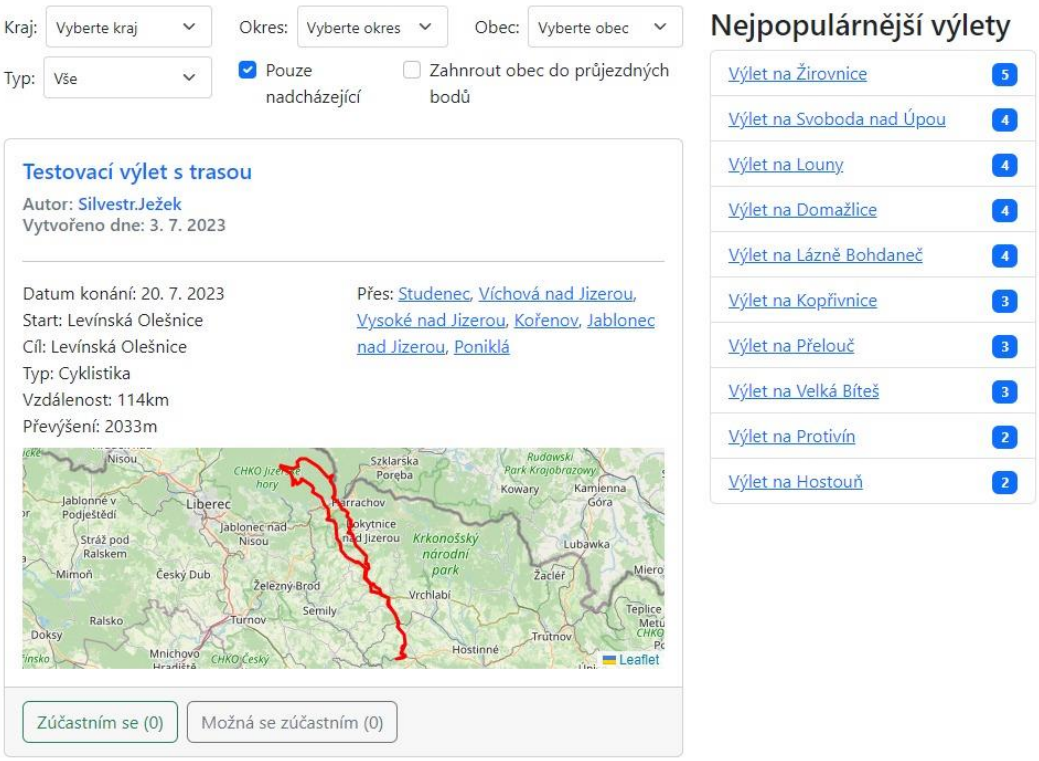
Na domovské stránce se nachází úvodní přivítání, a to včetně rychlého přístupu k procházení výletů, případné registraci. V případě přihlášeného uživatele je tlačítko registrace nahrazeno tlačítkem „Naplánovat výlet“. Níže na stránce se pak nachází nejnovější naplánované výlety.



Obr. 23 Domovská stránka
Zdroj: Vlastní zpracování

9.3 Procházení výletů

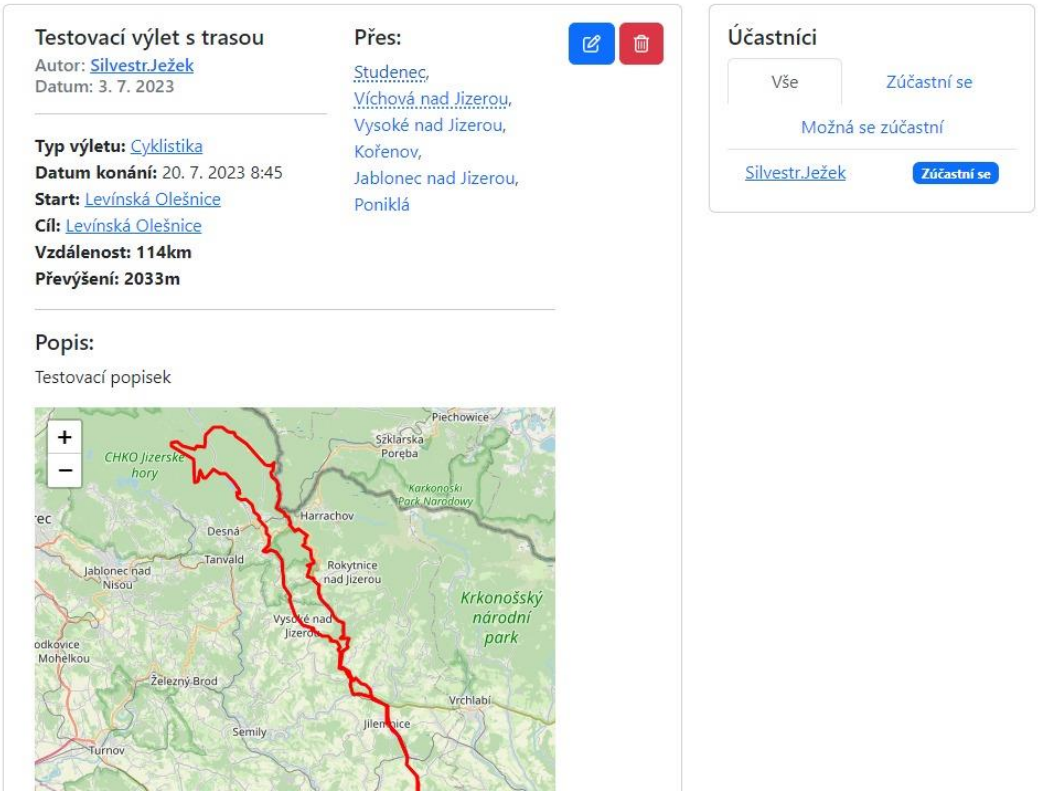
Na stránce procházení výletů je možné podrobnější vyhledávání výletů, lze zde výlety filtrovat podle kraje, okresu, či konkrétní obce. Obec pak lze zahrnout do průjezdných bodů, což umožňuje vyhledat výlety, které prochází danou obcí. Dále lze filtrovat výlety podle typu (turistika, cyklistika) či zobrazit pouze nadcházející výlety. Na pravé straně se pak nachází seznam nejpopulárnějších výletů (podle počtu zúčastněných). Jednotlivé výlety pak lze rozkliknout pro podrobnější informace.



Obr. 24 Procházení výletů
Zdroj: Vlastní zpracování

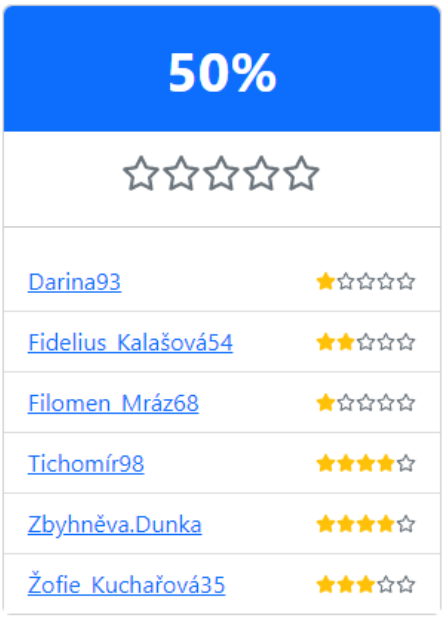
9.4 Detail výletu

Detail výletu zobrazuje veškeré dostupné informace o výletu, jako místo konání, cíl, vzdálenost, průjezdné body, mapu a podobně. Na průchozí body s tečkovanou čarou je možno najet myší pro zobrazení popisu daného bodu. Tvůrce nebo administrátor má možnost výlet upravit či smazat (ikonky v horní části), na pravé straně se pak nachází seznam účastníků, které je možné dále rozdělit na ty, které mají účast potvrzenou, nebo na ty, které se ještě pro účast rozhodují.



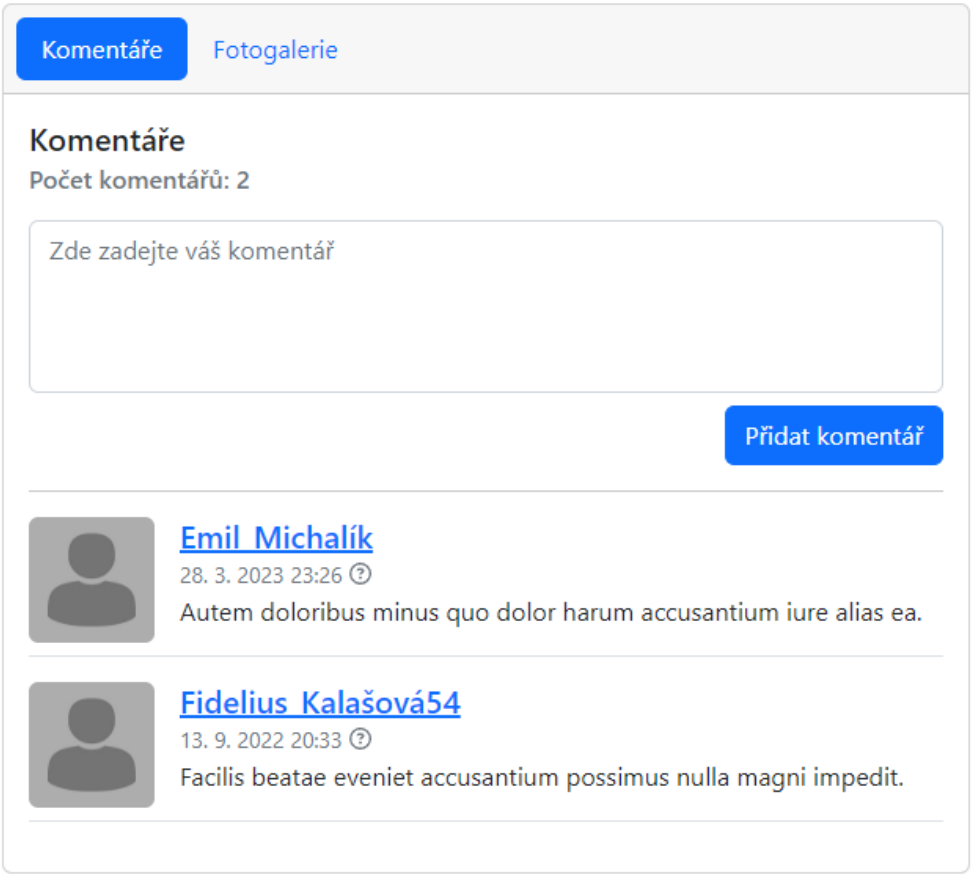
Obr. 25 Detail výletu
Zdroj: Vlastní zpracování

V případě již skončeného výletu, je možné výlet ohodnotit. V tomto případě se panel s hodnocením objeví na pravé straně nad účastníky.



Obr. 26 Hodnocení výletu
Zdroj: Vlastní zpracování

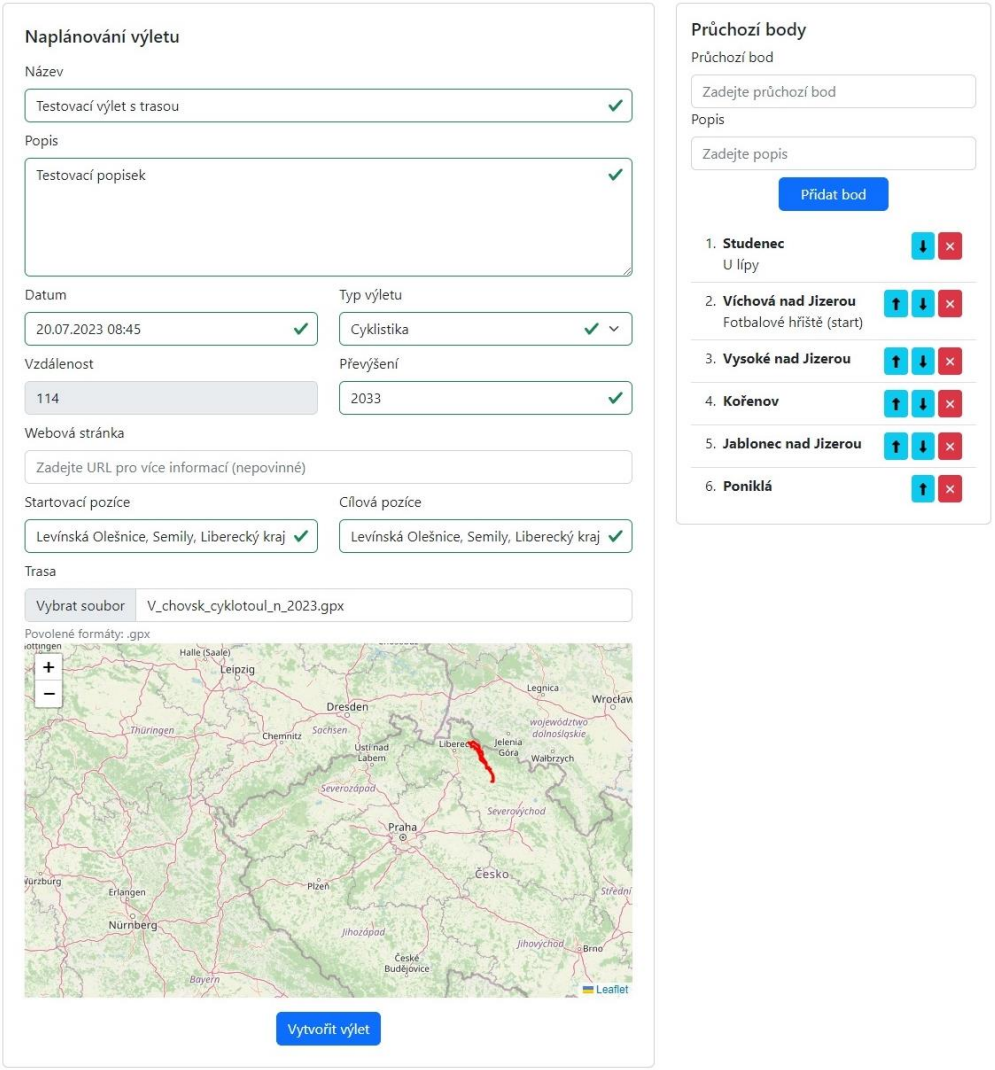
V dolní části se pak nachází prostor pro komentáře a fotogalerii. Podobně jako u hodnocení, je možné přidávat fotografie až po skončení výletu.



Obr. 27 Komentáře k výletu
Zdroj: Vlastní zpracování

9.5 Plánování výletu

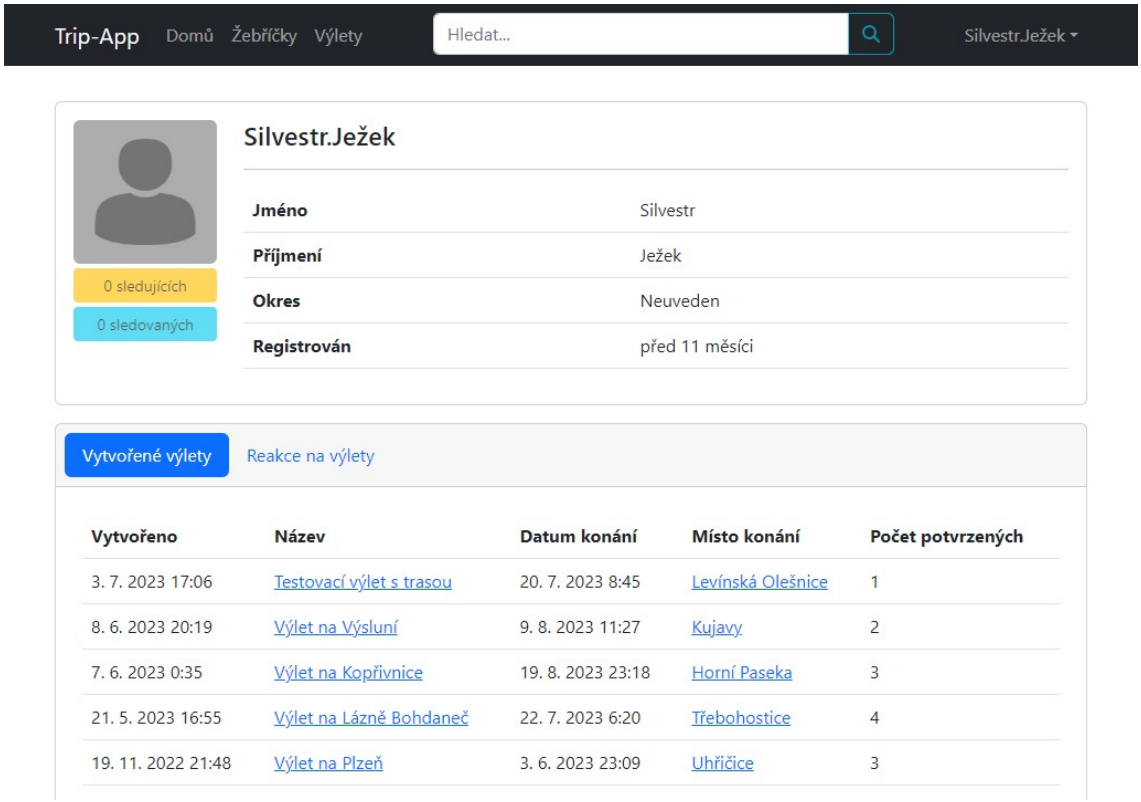
Plánování výletu se skládá z vyplnění povinných údajů, jako je název, popis, datum, typ, vzdálenost, startovní a cílové místo a nepovinných údajů, jako webová stránka, převýšení, trasa, a až 15 průchozích bodů. Vzdálenost a převýšení se vyplní automaticky v případě nahrání trasy (GPX souboru). Průchozí body pak lze v případě potřeby mazat, či měnit jejich pořadí.



Obr. 28 Plánování výletu
Zdroj: Vlastní zpracování

9.6 Uživatelský profil

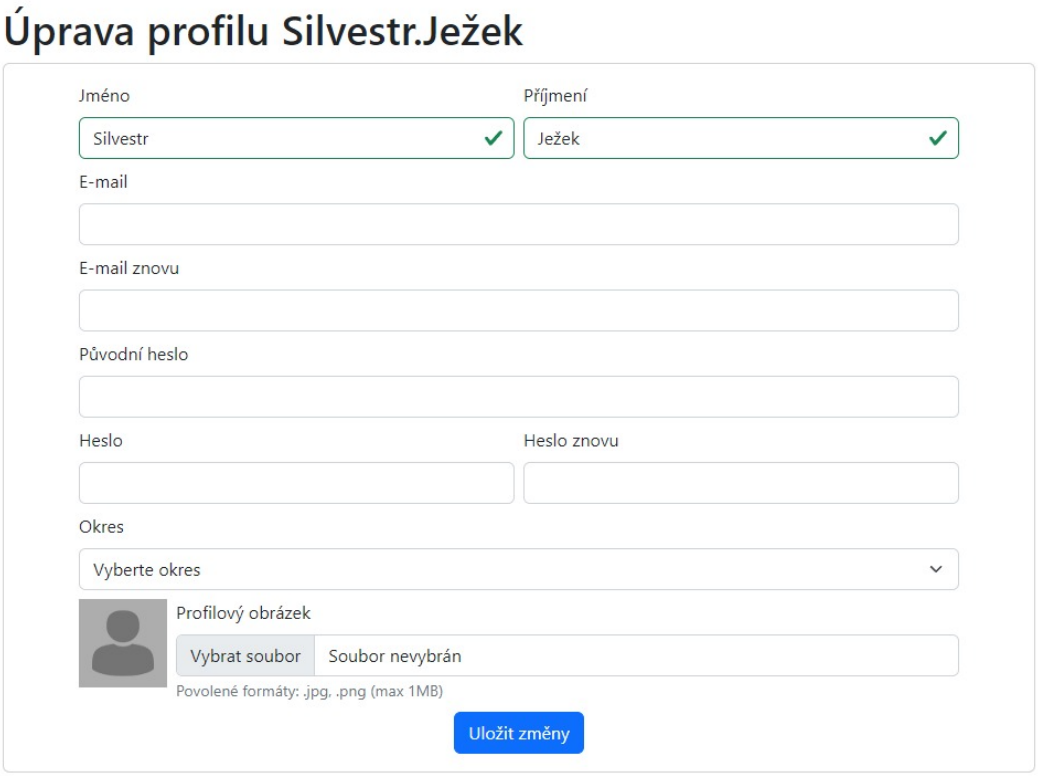
Uživatelský profil obsahuje informace o uživateli, jako profilový obrázek, jméno, příjmení, okres bydliště (tyto údaje musí sám uživatel zveřejnit v úpravě profilu) a datum registrace. Pod profilovým obrázkem je možné vidět počet sledujících a sledovaných uživatelů, po rozkliknutí se pak zobrazí seznam sledujících, nebo sledovaných uživatelů. Cizí uživatelský profil je pak možno sledovat. V případě, že si profil prohlíží administrátor, či majitel účtu, je zobrazen i registrační e-mail. Níže na stránce se pak nachází seznam vytvořených výletů a seznam výletů, na které uživatel reagoval (zda se zúčastní, nebo možná se zúčastní).



Obr. 29 Uživatelský profil
Zdroj: Vlastní zpracování

9.7 Úprava profilu

V úpravě profilu si uživatel může nastavit, případně změnit jméno, příjmení, e-mail, heslo, okres bydliště a profilový obrázek. Pro změnu hesla je třeba zadat zároveň i aktuální heslo.



Obr. 30 Úprava profilu
Zdroj: Vlastní zpracování

10 Nasazení aplikace

Před samotným nasazením je třeba serverovou a klientskou část připravit na prostředí, kde obě části aplikace budou spuštěny. Především se jedná o upravení adres pro endpointy. Či vypnutí nástrojů pro vývojáře.

10.1 Backend

U backendové části aplikace je potřeba připravit prostředí, kde bude aplikace spuštěna, pomocí proměnného prostředí, to musí zahrnovat informace o připojení k databázi, „secrety“ pro access a refresh tokeny, upřesnění adresy, na které bude backend spuštěn (slouží pro správně vrácení adresy např. pro obrázky) a adresa, kde je spuštěna frontendová část aplikace (požadavky z ostatních adres budou blokovány). Proměnné prostředí se skládá z párů klíč=hodnota. Většina služeb umožňující nasazení Express aplikace má k dispozici nástroj pro úpravu proměnného prostředí, případně lze využít *.env* soubor v kořenovém adresáři s Express aplikací.

Níže (Obr. 31) je příklad .env souboru:

```
DB_HOST="adresa mysql databáze"
DB_USER="uživatel k databázi "
DB_PASSWORD="heslo k databázi"
DB_DATABASE="název databáze "

ACCESS_TOKEN_SECRET=secret pro generování access tokenů
REFRESH_TOKEN_SECRET=secret pro generování refresh tokenů
BASE_URL="http://localhost:3500/api/v1/" #základní adresa k API
FRONTEND_URL="http://localhost:3000" #adresa kde je spuštěn frontend
```

Obr. 31 Příklad .env souboru
Zdroj: Vlastní zpracování

10.2 Frontend

U frontendové části je důležité změnit adresu k API, kde bude API spuštěno. To zahrnuje vytvoření záznamu v proměnném prostředí a klíčem REACT_APP_API_URL např:

```
REACT_APP_API_URL="http://localhost:3500/api/v1" #Adresa k API
```

Dále je vhodné použít modul *@fvilers/disable-react-devtools*. Jak již název napovídá, tento modul umožňuje vypnutí nástrojů pro vývojáře. Z tohoto balíčku je pak nutné ve spouštěcím souboru (*index.js*) importovat a vyvolat funkci *disableReactDevTools()*. Lze přidat ještě podmínku, aby došlo k vypnutí vývojových nástrojů pouze při produkci:

```
if(process.env.NODE_ENV === "production") disableReactDevTools();
```

10.3 Možnosti nasazení

Existuje mnoho služeb, které umožňují nasazení React a Express aplikací. Mezi populární služby pro hostování aplikací se řadí např:

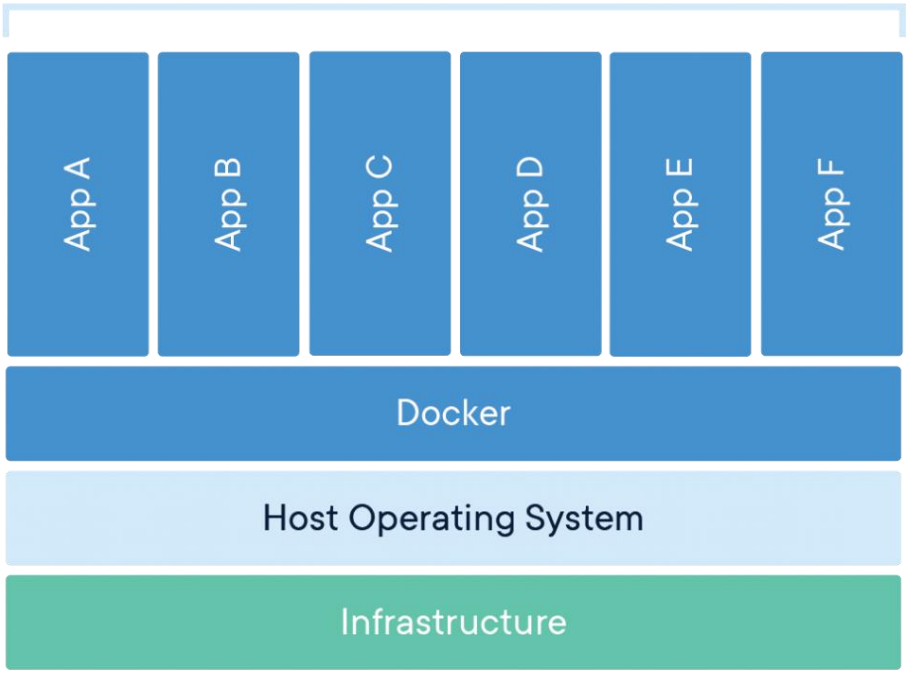
- Netlify – pro hostování statických webových aplikací, včetně React,
- Vercel – podobně jako Netlify, slouží pro hostování statických webových aplikací,
- Onrender – umožňuje hostování jak statických webových aplikací, tak i webových služeb (včetně React a Express),
- Heroku – pro hostování webových služeb (včetně Express, React, MySQL atd.),
- Docker – pro služby umožňující hostování docker kontejnerů,

- Amazon Web Services – nabízí hostování webových služeb (včetně React, Express, MySQL atd.),
- Microsoft Azure – Podobně jako Amazon Web Services.

Pro naší aplikaci je třeba hostování jak Express, tak React aplikace ale i MySQL databáze.

10.4 Docker

Docker je platforma pro tzv. kontejnerizaci aplikací. Jedná se o proces izolace aplikace a jejího prostředí s cílem usnadnit jejich provoz v různých prostředích. Docker umožňuje zabalit aplikaci a její závislosti do standardizovaného kontejneru, který je následně provozován uvnitř docker enginu ve virtualizovaném prostředí. Docker engine umožňuje kontejnerizovaným aplikacím pokaždé stejný provoz na jakékoliv infrastruktuře a je řešením tzv. „dependency hell“, kdy na různých počítačích, či platformách můžou být různé verze závislostí a eliminuje tím situace, kdy aplikace na některém počítači funguje, a na některém ne. [24].



Obr. 32 Schéma kontejnerizovaných aplikací
Zdroj: docker.com

10.4.1 Nasazení aplikace v Dockeru

V následujícím příkladu si ukážeme, jak sestavit docker image pro frontendovou a backendovou část aplikace a spustit jak databázový systém, tak i frontendovou část spolu s webovým serverem nginx a backendovou částí aplikace jako docker kontejnery v docker enginu.

V první řadě je důležité mít nainstalovaný docker engine, podrobný návod pro svoji platformu naleznete na stránkách dockeru (docker.com). Na adrese https://gitlab.com/tomas15420/trip-app jsou dostupné zdrojové kódy k aplikaci včetně inicializačního souboru pro databázi, základní konfigurace pro webový server nginx a konfigurace potřebné pro vytvoření docker imagů a spuštění docker kontejnerů. Struktura archivu je následující:

```
archiv/
├─ client/
│  ├─ Dockerfile
│  ├─ Zdrojový kód...
├─ server/
│  ├─ Dockerfile
│  ├─ Zdrojový kód...
├─ database/
│  ├─ init.sql
├─ nginx/
│  ├─ default.conf
├─ docker-compose.yml
```

Významnými soubory jsou tzv. „Dockerfily“, které se nachází jak u frontendové (client), tak i backendové (server) části aplikace. *Dockerfile* popisuje, jakým způsobem se má sestavit docker image, který pak může být spuštěn v docker kontejneru. V adresáři database se nachází inicializační script pro vytvoření databáze a v nginx adresáři se nachází výchozí konfigurace pro webový server nginx. docker-compose.yml pak umožňuje spustit více služeb (kontejnerů) najednou.

Server

```
FROM node:18.16.0-alpine
WORKDIR /app
COPY ["package.json", "package-lock.json", "./"]
RUN npm install --omit=dev
COPY . .
EXPOSE 3500
CMD ["npm", "start"]
```

Obr. 33 Dockerfile pro server
Zdroj: Vlastní zpracování

```
FROM node:18.16.0-alpine
```

Použití node jako základního image pro sestavovaný image.

```
WORKDIR /app
```

Specifikování pracovního adresáře uvnitř kontejneru.

```
COPY ["package.json", "package-lock.json", "./"]
```

Zkopírování package.json a package-lock.json do pracovního adresáře (/app).

```
RUN npm install --omit=dev
```

Nainstalování závislostí (node_modules) a vynechání vývojářský závislostí uvnitř kontejneru.

```
COPY . .
```

Zkopíruje všechny ostatní soubory a složky do pracovního adresáře uvnitř kontejneru (/app).

```
EXPOSE 3500
```

Specifikování portu, na kterém bude aplikace uvnitř kontejneru naslouchat.

```
CMD ["npm", "start"]
```

Nastavuje výchozí příkaz po spuštění kontejneru na *npm start*.

Client

```
FROM node:18.16.0-alpine as builder
ARG API_URL
ENV REACT_APP_API_URL=$API_URL
WORKDIR /app
COPY ["package.json", "package-lock.json", "./"]
RUN npm install -omit=dev
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/build ./usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Obr. 34 Dockerfile pro client
Zdroj: Vlastní zpracování

U clienta se využívá tzv. multi-stage buildu, kde v první fázi se sestaví React aplikace pomocí *npm run build* a sestavená React aplikace se pak zkopíruje do druhé fáze, která se skládá ze základního image webového serveru nginx.

```
ARG API_URL
ENV REACT_APP_API_URL=$API_URL
```

ARG definuje argument API_URL bez výchozí hodnoty, tento argument je možné pak předávat jako parametr při sestavení, ENV pak nastaví v proměnném prostředí REACT_APP_API_URL na hodnotu, předanou v parametru u API_URL. Použití ARG je nutné z důvodu nutnosti nastavení proměnné REACT_APP_API_URL při první fázi sestavení, a ne při samotném běhu kontejneru.

```
COPY --from=builder /app/build ./usr/share/nginx/html
```

Zkopíruje z první fáze (označenou jako builder) obsah složky build do složky */usr/share/nginx/html* uvnitř kontejneru.

NGINX

```
server {
    listen 80; #naslouchání na portu 80
    server_name localhost; #nastavení názvu serveru

    root /usr/share/nginx/html; #kořenový adresář pro soubory webové stránky

    #Soubor, který bude použit jako výchozí, pokud není specifikovaná konkrétní stránka
    index index.html;

    #pokud adresa začíná /
    location / {
        #snaží se najít soubor, který odpovídá adrese,
        #pokud neexistuje odkáže ho na index.html (kvůli SPA aplikacím)
        try_files $uri /index.html;
    }

    #pokud adresa začíná /api
    location /api {
        #presměrování na adresu kontejneru backend a port 3500
        proxy_pass http://backend:3500;
    }
}
```

Obr. 35 Výchozí konfigurace NGINX
Zdroj: Vlastní zpracování

Docker compose

docker-compose.yml je soubor, jehož účelem je konfiguraci více služeb (kontejnerů) pro Docker. Umožňuje ovládání kontejnerů jako jednoho celku. Možné příkazy jsou např. docker compose up (spuštění celku), down (zastavení celku).

Na obrázku níže (Obr. 36) je konfigurace docker-compose nakonfigurována pro spuštění na systému debian11 na localhostu.

```
version: '3'
services:
  #kontejner s mysql
  db:
    container_name: "mysql" #název kontejneru
    image: mysql #image kontejneru (stáhne se z hub.docker.com)
    restart: unless-stopped #bude restartován dokud nebude explicitně vypnut uživatelem
    #Nastavení proměnného prostředí uvnitř kontejneru
    environment:
      - MYSQL_ROOT_PASSWORD=heslo_roota
      - MYSQL_USER=db_user #uživatel pro připojení k databázi z aplikace
      - MYSQL_PASSWORD=db_password #heslo uživatele
      - MYSQL_DATABASE=trip-app #název databáze (neměnit)
    volumes:
      #namapování složky database z archivu do adresáře v kontejneru
      #pro načtení incializačního skriptu
      - ./database:/docker-entrypoint-initdb.d
  backend:
    container_name: "trip-app-api"
    restart: unless-stopped
    #sestavení kontejneru
    build:
      context: ./server #z adresáře server
      dockerfile: Dockerfile
    volumes:
      #mapování z důvodu zachování dat při smazání nebo znovusestavení kontejneru
      #namapování hostitelské složky ./trip-api/images na složku /app/images v kontejneru
      - ./trip-api/images:/app/images
      #namapování hostitelské složky ./trip-api/files na složku /app/files v kontejneru
      - ./trip-api/files:/app/files
    environment:
      - DB_HOST=db #adresa kontejneru db (může zůstat takto)
      - DB_PASSWORD=db_password #definovano výše
      - DB_USER=db_user #definovano vyse
      - DB_DATABASE=trip-app #název databáze (neměnit)

      - ACCESS_TOKEN_SECRET=tajne_heslo
      - REFRESH_TOKEN_SECRET=tajne_heslo2

      - BASE_URL=http://localhost/api/v1/ #adresa api
      - FRONTEND_URL=http://localhost #adresa frontendu (ostatní adresy budou blokovány)
    depends_on:
      - db #závisí na kontejneru db
  frontend:
    container_name: "trip-app-client"
    restart: unless-stopped
    build:
      context: ./client
      dockerfile: Dockerfile
      args:
        #specifikování argumentu API_URL, bude předán v Dockerfile
        - API_URL=http://localhost/api/v1/ #adresa k api
    ports:
      - "80:80"
    volumes:
      #namapování výchozí konfigurace do kontejneru
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
```

Obr. 36 Vysvětlení docker compose konfigurace
Zdroj: Vlastní zpracování

Zjednodušený návod na spuštění aplikace

1. Nainstalování docker enginu (dostupný na adrese: docker.com),
2. Stáhnutí zdrojových a konfiguračních souborů z https://gitlab.com/tomas15420/trip-app,
3. Otevření terminálu v adresáři se zdrojovými soubory a *docker-compose.yml*
4. Spuštění příkazu *docker compose up,*
5. Aplikace je nyní spuštěna na adrese http://localhost.

11 Shrnutí výsledků

Cílem této práce bylo vyvinout webovou aplikaci umožňující plánování turistických výletů a v teoretické části seznámit čtenáře s použitými technologiemi a metodami.

Aplikace umožňuje registrovaným uživatelům plánovat a spravovat vlastní výlety. Důležitou součástí je možnost interagovat s pořadatelem či ostatními zájemci o účast pomocí komentářů pod jednotlivými výlety. Užitečnou funkcionalitou je také možnost reagování na výlet pomocí možností „zúčastním se“, nebo „možná se zúčastním“, to dává pořadateli orientační představu o účasti na jeho naplánovaném výletu. Další významnou funkcionalitou je filtrování výletů dle různých kritérií, které umožňují uživatelům najít výlet podle lokality (kraje, okresu a obce) nebo typu (cyklistika nebo turistika). Obec pak lze zahrnou i do průjezdného bodu, tzn. že má uživatel možnost najít výlet, který např. prochází jeho bydlištěm a připojit se k výletu v průběhu. Po skončení výletu je možné výlet ohodnotit pro zpětnou vazbu pořadateli a přidat fotografie pořízené během výletu. V neposlední řadě je součástí aplikace žebříček nejlépe hodnocených výletů, výletů s největším počtem účastníků a seznam nejlepších uživatelů z hlediska vytvořených výletů.

Do aplikace byly nad rámec zadání přidány některé funkce, jako je možnost přidání trasy k výletu. Toto umožňuje nahrát trasu při vytváření nebo úpravě výletu ve formátu .gpx, taková trasa se dá exportovat z různých služeb, jako např. mapy.cz nebo ze zaznamenaných tras z populárních fitness aplikací, jako např. Strava. Další funkcí navíc je sledování uživatelů, kdy je možné uživatele sledovat podobně, jako na sociálních sítích.

Samotná práce hodně pomohla k osobnímu rozvoji v oblasti vývoje webových aplikací a naučení se nových metod, způsobů vývoje a zlepšení v programovacím jazyce JavaScript.

12 Závěry a doporučení

Cílem této práce bylo vytvořit webovou aplikaci pro plánování turistických či cyklistických výletů. Pro splnění tohoto cíle bylo potřeba nastudovat různé technologie a postupy pro vývoj webových aplikací a zvolit, které z nich použít.

Vzhledem k počáteční neznalosti některých použitých technologií bylo nutné použité technologie nastudovat. K tomu byl hodně nápomocný kurz od Dava Graye zaměřený na vývoj webových aplikací za pomoci frameworku Express a React: (https://www.youtube.com/@DaveGrayTeachesCode).

Během vývoje byl kladen větší důraz na funkční stránku aplikace než na její vzhled. Vzhledem k důležitosti uživatelského rozhraní pro uživatele, by bylo vhodné v budoucnu věnovat více pozornosti samotné prezentaci aplikace a tím zvýšit atraktivitu pro uživatele.

V aktuálním stavu aplikace lze uživatele sledovat, přestat sledovat a zobrazit seznam sledujících a sledovaných uživatelů. Avšak samotné sledování uživatelů nemá žádný funkční benefit, proto by bylo vhodné v budoucím vývoji aplikace dodat např. personalizovanou stránku pro každého uživatele, kde by byl přehled vytvořených výletů sledovanými uživateli, či výletů, kterých se sledovaný uživatel účastní nebo přidal k nějakému výletu komentář apod.

V souhrnu lze říct, že vývoj webové aplikace byl úspěšný a výsledkem je funkční aplikace, která je schopná plnit svůj účel. Avšak pro maximalizaci potenciálu, je v budoucnu vhodné věnovat se dalšímu vývoji a zlepšení aplikace. To zahrnuje nejenom dokončení funkcionality pro sledování uživatelů, ale i zdokonalení uživatelského rozhrání a případně přidání dalších funkcí, jako např. soukromé zprávy, editor tras přímo v aplikaci a další.

13 Seznam použité literatury

[1] LEVINSON, Deborah a Todd BELTON. BUILD YOUR FIRST WEB APP: Learn to Build Web Applications from Scratch. 1. New York: Sterling Publishing, 2017, 280 s. ISBN 978-1-4549-2634-4.

[2] VOLLE, Adam. Web application. Encyclopedia Brittanica [online]. Chicago: Encyclopædia Britannica, 2022 [cit. 2023-01-13]. Dostupné z: https://www.britannica.com/topic/Web-application

[3] What Is a Web Application? (With Benefits and Jobs). Indeed Career Guide [online]. 2020 [cit. 2023-01-13]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application

[4] What Is a Database?. Oracle [online]. Austin: Oracle, 2021 [cit. 2023-01-18]. Dostupné z: https://www.oracle.com/database/what-is-database/

[5] ŠKODA, Jan. Databázové systémy. Inovace VOV [online]. Praha: ČVUT, 2019 [cit. 2023-01-18]. Dostupné z: https://www.vovcr.cz/odz/tech/393/page03.html

[6] HRDINOVÁ, Jana. Informační systém cestovní kanceláře [online]. Pardubice, 2011 [cit. 2022-08-18]. Dostupné z: https://dk.upce.cz/handle/10195/39087 . Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky. Vedoucí práce Iva Ruličková.

[7] SAKS, Elar. JavaScript frameworks: Angular vs React vs Vue [online]. Helsinki, 2019 [cit. 2023-01-26]. Dostupné z: https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf. Bakalářská práce. Haaga-Helia University of Applied Sciences.

[8] A Survey on different Framework of PHP. International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS) [online]. June 2017, 6(6), 155-158 [cit. 2023-01-26]. ISSN 2278-2540. Dostupné z: https://www.ijltemas.in/DigitalLibrary/Vol.6Issue6/155-158.pdf

[9] GUPTA, Saksham a Shallu BASHAMBU, 2020. Implementation of Java Frameworks and APIs for Web Applications. International Journal of Scientific Research in Science, Engineering and Technology [online]. 7(2), 427–432 [cit. 2021-03-26]. Dostupné z: https://ijsrset.com/paper/6324.pdf

[10] WIRFS-BROCK, Allen a Brendan EICH. JavaScript: the first 20 years. Proceedings of the ACM on Programming Languages [online]. 2020, 4(HOPL), 1-189 [cit. 2023-01-30]. ISSN 2475-1421. Dostupné z: doi:10.1145/3386327

[11] HERRON, David. Node.js Web Development: Server-side web development made easy with Node 14 using practical examples [online]. 5. Birmingham: Packt Publishing, 2020 [cit. 2023-01-30]. ISBN 978-1-83898-757-2. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=4_30DwAAQBAJ

[12] SYED, Basarat. Beginning Node.js: Unleash the pover of node.js and create highly scalable websites [online]. 1. Berkely: Apress Berkeley, 2014 [cit. 2023-01-30]. ISBN 978-1-4842-0187-9. Dostupné z: https://books.google.cz/books?id=AlknCgAAQBAJ

[13] MARDAN, Azat. Express.js Guide: The Comprehensive Book on Express.js [online]. 28.5.2014. Victoria: Leanpub, 2014 [cit. 2023-01-31]. ISBN 1494269279. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=5eGRAwAAQBAJ

[14] WIERUCH, Robin. The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js. 2022. Independently Published, 2018. ISBN 9781720043997.

[15] FEDOSEJEV, Artemij. React.js Essentials: A fast-paced guide to designing and building scalable and maintainable web apps with React.js [online]. 1. Birmingham: Packt Publishing, 2015 [cit. 2023-03-30]. ISBN 1782174621. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=Rhl1CgAAQBAJ

[16] RAWAT, Bhupest, Suryari PURNAMA a Mulyati MULYATI. MySQL Database Management System (DBMS) On FTP Site LAPAN Bandung. International Journal of Cyber and IT Service Management [online]. 2021, 1(2), 173-179 [cit. 2023-03-31]. ISSN 2808-554X. Dostupné z: doi:10.34306/ijcitsm.v1i2.47

[17] MEHTA, Chintan, Ankit BHAVSAR, Hetal OZA a Subhash SHAH. MySQL 8 Administrator's Guide: Effective guide to administering high-performance MySQL 8 solutions [online]. 1. Birmingham: Pack Publishing, 2018 [cit. 2023-03-31]. ISBN 1788393848. Dostupné z: https://books.google.cz/books?id=EJdMDwAAQBAJ&

[18] DOGLIO, Fernando. REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development [online]. 2. New York: Apress, 2018 [cit. 2023-04-03]. ISBN 978-1-4842-3715-1. Dostupné z: https://www.google.cz/books/edition/REST_API_Development_with_Node_js/PsNlDwAAQBAJ

[19] PATIL, Krutika. Redux State Management System: A Comprehensive Review. International Journal of Trend in Scientific Research and Development [online]. Prosinec 2022, 6(7), 1021-1022 [cit. 2023-06-29]. ISSN 2456-6470. Dostupné z: https://www.ijtsrd.com/papers/ijtsrd52530.pdf

[20] ABRAMOV, Dan. Redux Toolkit: The official, opinionated, batteries-included toolset for efficient Redux development. Redux Toolkit [online]. [cit. 2023-07-01]. Dostupné z: https://redux-toolkit.js.org/

[21] OTTO, Mark a Jacob THORNTON. Bootstrap: The most popular HTML, CSS, and JS library in the world. Bootstrap [online]. [cit. 2023-07-01]. Dostupné z: https://getbootstrap.com/

[22] ADAM, Stenly Ibrahim, Jimmy MOEDJAHEDY a Jeremiah MARAMIS. RESTful Web Service Implementation on Unklab Information System Using JSON Web Token (JWT). In: 2020 2nd International Conference on Cybernetics and Intelligent System [online]. Airmadidi: Klabat University, 2020 [cit. 2023-07-20]. Dostupné z: doi:10.1109/ICORIS50180.2020.9320801

[23] ARIAS, Dan a Sam BELLEN. What Are Refresh Tokens and How to Use Them Securely: Learn about refresh tokens and how they help developers balance security and usability in their applications. Auth0 blog: by Okta [online]. Bellevue: Auth0, 2021 [cit. 2023-07-20]. Dostupné z: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

[24] What is a Container? Docker [online]. Palo Alto: Docker, 2023 [cit. 2023-07-31]. Dostupné z: https://www.docker.com/resources/what-container/

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2021/2022

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Tomáš Mach**
Osobní číslo:	**I2000387**
Adresa:	Levínská Olešnice 103, Levínská Olešnice, 51401 Jilemnice, Česká republika
Téma práce:	Webová aplikace pro plánování turistických výletů
Téma práce anglicky:	Web application for planning trips
Jazyk práce:	Čeština
Vedoucí práce:	Mgr. Hana Rohrová Katedra informačních technologií

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit webovou aplikaci plánovače zejména cyklo ale i pěších turistických výletů.
Uživatel si pomocí aplikace naplánuje výlet (start, cíl, body trasy – obce, popis, datum atd.) a ostatní uživatelé budou moci na výlet reagovat (klást dotazy, potvrdit účast, atd.).
Výlety půjde dále vyhledávat (podle názvu, bodů trasy), hodnotit a přidávat fotografie zúčastněnými uživateli.
Dále aplikace bude obsahovat žebříček nejlépe hodnocených výletů a uživatelů.
Osnova:
1. Rešerše
2. Použité technologie
3. Návrh backendu
4. Návrh databáze
5. Návrh frontendu
6. Uživatelské účty
7. Ovládání aplikace
8. Nasazení aplikace

Seznam doporučené literatury:

Podpis studenta: Mach Datum: 5.3.2022

Podpis vedoucího práce: Datum: