

Gymnázium, Praha 6, Arabská 14

Programování

MATURITNÍ PRÁCE



2025

Sabina Javůrková

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

MATURITNÍ PRÁCE

Předmět: **Programování**

Téma: **Webová aplikace, která usnadňuje cestování po České republice**

Autorka: **Sabina Javůrková**

Třída: **4.E**

Školní rok: **2024/2025**

Vedoucí práce: **Mgr. Jan Lána**

Třídní učitelka: **Mgr. Blanka Hniličková**

Čestné prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitý literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne

.....

Podpis

Anotace

Cílem mé maturitní práce je naprogramovat webovou aplikaci, která lidem zjednoduší objevování České republiky. Aplikace ČundrAppka uživatelům umožní snadněji najít důležité a aktuální informace k cestování a pomůže spojovat lidi s podobnými zájmy. Její hlavní funkce jsou: inzeráty, mapa, komunitní fórum a cestovní balíčky. K realizaci projektu využiji takzvaný MERN Stack, což je kolekce softwaru pro vývoj webových aplikací a skládá se z MongoDB, Express, React a Node.

Abstract

The aim of my graduation project is to program a web application which makes it easier for people to explore The Czech Republic. Application ČundrAppka enables users to quickly find important and up-to-date information for traveling and helps bring people with the same interests together. Its main functions are: travel ads, map, community forum and travel packages. For the build of this project I will use the so-called MERN Stack which is a collection of tools for development of web applications and it is made out of MongoDB, Express. React and Node.

Zadání projektu

ČundrAppka je webová aplikace, která usnadňuje orientaci po České republice a její objevování. Na stránce bude možné si zobrazit informace k městům a památkám vygenerované aplikací, ale také osobní postřehy ostatních uživatelů. V případě, že by uživatel neměl s kým vyrazit na "čundr", bude mít možnost zveřejnit inzerát a nebo si nějaký co odpovídá jeho specifikacím zobrazit a inzerenta kontaktovat. Klíčové funkce: inzeráty, mapa, komunitní fórum a cestovní balíčky. (*oficiální zadání odevzdané na začátku školního roku*)

Obsah

1. Úvod.....	1
2. Použité technologie.....	2
2.1. MongoDB.....	3
2.2. Express.js.....	3
2.3. React.js.....	3
2.4. Node.js.....	4
2.5. TypeScript.....	4
3. Postup instalace.....	5
3.1. Požadavky.....	5
3.2. Klonování repozitáře a instalace závislostí.....	6
3.3. Vytvoření .env souborů a spuštění.....	6
4. Struktura projektu.....	8
4.1. Backend.....	8
4.1.1. app.ts.....	9
4.1.2. routes.....	9
4.1.3. controllers.....	10
4.1.4. models.....	13
4.2. Frontend.....	14
4.2.1. index.tsx.....	14
4.2.2. App.tsx.....	15
4.2.3. pages.....	16
4.2.3.1. ads.....	17
4.2.3.2. community_forum.....	18
4.2.3.3. travel_packages.....	19
4.2.3.4. users.....	20
4.2.4. components.....	21
4.2.5. hooks.....	22
4.2.6. context.....	25
4.2.7. models.....	26

5. Implementace klíčových funkcí.....	28
5.2. Inzeráty.....	28
5.2.1. Reprezentace v databázi.....	29
5.2.2. API.....	30
5.2.3. Uživatelské rozhraní.....	31
5.3. Komunitní fórum.....	34
5.3.1. Reprezentace v databázi.....	34
5.3.2. API.....	35
5.3.3. Uživatelské rozhraní.....	36
5.4. Mapa.....	38
5.4.1. API.....	39
5.5. Cestovní balíčky.....	39
5.5.1. Uživatelské rozhraní.....	39
5.6. Uživatelé.....	41
5.6.1. Reprezentace v databázi.....	41
5.6.2. API.....	42
5.6.3. sessions.....	43
5.6.4. Uživatelské rozhraní.....	44
6. Závěr.....	46
Seznam internetových zdrojů.....	47
Seznam obrázků.....	48

1. Úvod

Pro téma mojí ročníkové a maturitní práce jsem si vybrala webovou aplikaci, která má uživatelům zjednodušit cestování po České republice. Toho docílí tím, že jim umožňuje využívat 4 hlavní funkce: inzeráty, mapa, komunitní fórum a cestovní balíčky. Funkce inzeráty jsou pomocníkem pro lidi, kteří chtějí cestovat, ale nemají s kým. Složka s mapou obsahuje pohled na Českou republiku za pomocí Google Maps, tím pádem zahrnuje všechny užitečné funkce jako druh mapy a nebo Street View. Na komunitním fóru mají uživatelé možnost zveřejňovat příspěvky se svými postřehy ke konkrétním městům a tím poskytují aktuální informace ostatním. Stránka s cestovními balíčkami obsahuje turistické informace k dohromady deseti největším českým městům.

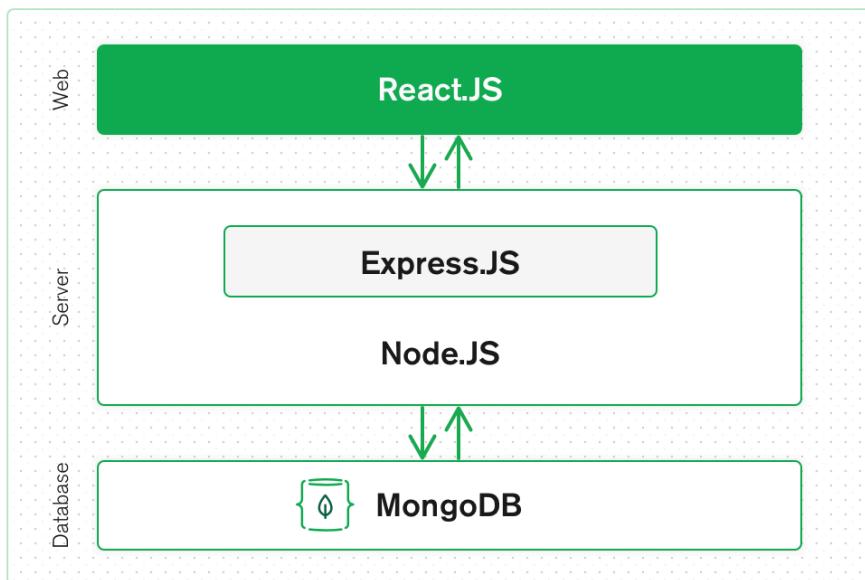
K vytvoření tohoto projektu jsem se rozhodla využít technologie z MERN Stacku, tedy databázi MongoDB, backend framework Express(.js), knihovnu React(.js) a runtime prostředí Node(.js). Všechny tyto nástroje operují na programovacím jazyce JavaScript. Nicméně, díky tomu, že JavaScript není původně backend jazykem a oproti ostatním jazykům mu chybí spousta funkcí, tak jsem se rozhodla psát v jazyce TypeScript, který je postavený na JavaScriptu a doplňuje ho o chybějící vymoženosti.

Tato dokumentace přiblíží průběh tvorby projektu a poskytne čtenářovi informace například o tom, jak ho z provoznit nebo jak chápat jeho hierarchii.

2. Použité technologie

Celý projekt je naprogramovaný za pomocí technologií z MERN Stacku, který se využívá k tvorbě webových aplikací. Jednotlivá písmena v názvu postupně představují: MongoDB, Express(.js), React(.js) a Node(.js). Díky své flexibilitě, rychlosti a rozsáhlou komunitou je velmi populární a stává se stále používanějším při vytváření dynamických webových aplikací. [1]

Protože všechny technologie v této kolekci fungují na JavaScriptu je možné mít celý projekt postavený právě na něm. A jelikož jsem chtěla mojí práci vyzdvihnout na vyšší úroveň, tak jsem místo JavaScriptu, který může být ve větších projektech nešikovný a nebezpečný (chybí mu type-checking, není object-oriented atd.) vybrala TypeScript, který je na něm postavený a doplňuje jeho mezery. [2]



Obrázek 1 - grafická reprezentace MERN kolekce

2.1. MongoDB

MongoDB je NoSQL databázový systém, který ukládá data ve formátu JSON podobných dokumentů, což umožňuje flexibilní a škálovatelnou práci s daty. Díky své architektuře podporuje horizontální škálování, replikaci a vysokou dostupnost. MongoDB je oblíbené pro moderní aplikace, které vyžadují rychlé čtení a zápis, jako jsou webové a mobilní aplikace. Podporuje také bohaté dotazovací možnosti, indexování a agregace, což usnadňuje práci s velkými objemy dat. [1]

2.2. Express.js

Express.js je minimalistický a flexibilní webový framework pro Node.js, který usnadňuje tvorbu serverových aplikací a API. Poskytuje jednoduché rozhraní pro správu směrování (routing), middleware a práci s HTTP požadavky a odpověďmi. Díky své modularitě umožňuje snadné rozšíření pomocí různých balíčků a integraci s databázemi, autentizací a dalšími službami. Express.js je oblíbenou volbou pro vývojáře díky své rychlosti, jednoduchosti a široké komunitní podpoře. [1]

2.3. React.js

React.js je populární JavaScriptová knihovna pro tvorbu uživatelských rozhraní. Umožňuje efektivní vývoj díky komponentovému přístupu a virtuálnímu DOM, což zlepšuje výkon. Podporuje jednostranný tok dat a správu stavu pomocí Redux nebo Context API. Díky široké komunitě a moderním funkcím, jako jsou Hooks, je ideální pro škálovatelné webové aplikace. [1]

2.4. Node.js

Node.js je runtime prostředí pro JavaScript postavené na V8 engine od Googlu, které umožňuje spouštění JavaScriptového kódu na straně serveru. Díky asynchronní architektuře založené na událostech a neblokujícím vstupu/výstupu je Node.js vhodné pro výkonné a škálovatelné aplikace, jako jsou webové servery, API nebo real-time aplikace. Nabízí rozsáhlý ekosystém modulů dostupných přes správce balíčků npm, což usnadňuje vývoj a integraci různých funkcionalit. [1]

2.5. TypeScript

TypeScript je nadstavba jazyku JavaScript, která přidává statické typování a další moderní funkce usnadňující vývoj robustních aplikací. Díky statickému typování umožňuje TypeScript detektovat chyby už při vývoji, což zlepšuje kvalitu kódu a snižuje počet runtime chyb. Navíc podporuje moderní syntaxi JavaScriptu a umožňuje její použití i v prostředích, která ještě nepodporují nejnovější standardy, protože se při komplikaci převádí na klasický JavaScript.

Další výhodou TypeScriptu je jeho podpora objektově orientovaného programování, jako jsou rozhraní, třídy a moduly, což usnadňuje organizaci kódu ve velkých projektech. Díky kompatibilitě s existujícími knihovnami JavaScriptu lze TypeScript snadno integrovat do stávajících projektů, přičemž mnoho populárních frameworků, jako je Angular, React nebo Vue, ho aktivně podporuje. S pomocí TypeScriptu mohou vývojáři psát čitelnější, bezpečnější a lépe udržovatelný kód. [2]

3. Postup instalace

Jelikož projekt využívá více technologií, tak je ke správné instalaci a konfiguraci potřeba pár akcí. Bez absolvování všech dole napsaných kroků aplikace nebude správně fungovat, takže pro plnohodnotný zážitek doporučuji si postup pořádně přečíst a následovat.

3.1. Požadavky

Nejprve na klonování repozitáře je potřeba mít nainstalovaný systém Git, který většina čtenářů určitě mít bude, ale pro celistvost postupu jej zmíním. Přejděte na oficiální stránky Gitu <https://git-scm.com/downloads>, vyberte verzi odpovídající Vašemu operačnímu systému a stáhněte instalační balíček. Spusťte instalační soubor a postupujte podle pokynů. Pro ověření instalace v terminálu nebo příkazovém řádku zavolejte `git --version`.

Potom je nutné mít nainstalovaný Node.js, který umožňuje spuštění JavaScriptu na straně serveru. Instalační soubor získate na oficiálních stránkách Node <https://nodejs.org/en>. Po stažení následujte pokyny a správnou instalaci můžete ověřit spuštěním příkazu `node -v` nebo `npm -v`. [3]

Jako poslední je nezbytné mít k projektu MongoDB databázi. Tu si vytvoříte buďto online přes MongoDB atlas, tak že si na stránce <https://www.mongodb.com/products/platform/atlas-database> založíte účet a vytvoříte takzvanou “cluster”. Nebo druhý způsob je si lokálně nainstalovat aplikaci z odkazu oficiálních stránek MongoDB <https://www.mongodb.com/try/download/community>. Dále následujte pokyny instalatéru a vytvořte “cluster”.

3.2. Klonování repozitáře a instalace závislostí

Jakmile máte nainstalovaný Git, tak je klonování repozitáře velmi jednoduché. Otevřete si terminál nebo příkazový řádek a za pomocí příkazu `cd` se přesuňte do složky ve které chcete mít naklonovaný projekt. Spusťte klonovací příkaz s odkazem na repozitář takto: `git clone https://github.com/gyarab/2024-4e-Javurkova-CundrAppka.git` a potom počkejte než se akce dokončí. Následně přejděte do složky projektu příkazem `cd 2024-4e-Javurkova-CundrAppka`. Takto máte repozitář s projektem zkopiřovaný k sobě.

Nyní je praktické si otevřít druhý terminál, jeden pro backend a druhý pro frontend. Nejprve začneme s frontendem, příkazem `cd client` přejdete do složky kde zavoláte `npm install`, což Vám nainstaluje všechny závislosti frontenu. Tato instalace bude chvíli trvat a tak v druhém terminálu přejděte do složky server zavoláním příkazu `cd server`, potom napište stejný příkaz jako u frontendu `npm install` pro instalaci backend závislostí.

3.3. Vytvoření .env souborů a spuštění

Posledním krokem před spuštěním aplikace je vytvoření dvou `.env` souborů (jeden pro frontend, druhý pro backend) do kterých se dávají informace, které není bezpečné sdílet nebo si je uživatel musí určit sám, takže nejsou v mé veřejném repozitáři.

Nejprve si ho vytvoříme na frontendu - ve svém kód editoru klikněte na složku `client` a v ní vytvořte soubor s názvem “`.env`”. Do tohoto souboru uložíte svůj Google API klíč a to do proměnné `REACT_APP_GOOGLE_MAPS_API_KEY` (Obr. 2). Tento klíč získáte na stránkách <https://console.cloud.google.com/>, kde si založíte projekt, a následně si necháte vygenerovat klíč. [4]

```
REACT_APP_GOOGLE_MAPS_API_KEY="muj-API-klic"
```

Obrázek 2 - ukázka *.env* na frontendu

Druhý *.env* soubor vytvořte v backend složce server. V tomto souboru budou uložené minimálně 2 věci: *MONGO_URI*, *SESSION_SECRET* a nepovinně *PORT*, na kterém poběží backend (defaultně 8000). Do proměnné *MONGO_URI* uložíte odkaz na databázi, kterou jste si vytvořili buďto v aplikaci (formát přibližně “`mongodb://localhost:27017/`”) nebo na internetu (formát přibližně “`mongodb+srv://username:password-for-user@cluster-name.rsb6wdk.mongodb.net/?retryWrites=true&w=majority&appName=cluster-name`”). Do *SESSION_SECRET* se napíše krátký náhodný string, který představuje tajný klíč za pomocí kterého aplikace šifruje data o sessions (více o sessions v kapitole 5.6.3.). V případě, že jste nastavili PORT na něco jiného než 8000 je zapotřebí upravit proměnnou *proxy* v souboru *package.json*, která je ve frontend složce. (Obr. 3)

```
PORT = 8000
MONGO_URI = "link-to-db"
SESSION_SECRET = "x#i?jf?1#$1" // random string
```

Obrázek 3 - ukázka *.env* na backendu

Po absolvování všech výše zmíněných kroků můžete aplikaci spustit. To se dělá pro backend a frontend zvlášť příkazem *npm start*. Tento příkaz zavolejte z obou terminálů - na frontendu (ve složce *client*) a backendu (ve složce *server*). Obzvláště pro frontend spouštění nějakou dobu trvá, ale po chvíli by se Vám sama měla otevřít stránka s projektem, pokud ne, přejděte v prohlížeči na adresu `http://localhost:3000/` nebo `http://127.0.0.1:3000/`.

4. Struktura projektu

Na nejvyšší úrovni projektu jsou dvě složky *client* a *server*, ty jak už jsem zmínila představují frontend a backend. Frontend je vytvořen za pomocí Reactu s TypeScriptem a backend je postaven na Node.js s frameworkm Express, který zajišťuje API. Mimo tyto složky je zde ještě soubor README.md, který se zobrazuje při otevření repozitáře na Githubu.

4.1. Backend

Při popisu backendu mluvím o obsahu složky *server*, tedy o části aplikace, která není vidět a primárně se stará o logiku a funkčnosti většiny důležitějších funkcí. Tato složka kromě dalších podsložek, které jsou rozebrány v dalších podkapitolách obsahuje konfigurační JSON soubory, díky kterým aplikace funguje tak ja má - jsou v nich například závislosti, které jsou instalovány po příkazu *npm install* nebo konfigurace Typescriptu. Potom je zde soubor *.env*, který jsme si v předchozí kapitole vytvořili a *.gitignore*, kde si autor napíše věci, které nechce při commitu posílat na Github - to v tomto případě znamená soubor *.env* a složku *node_modules*.

Složka *node_modules* obsahuje obrovské množství složek a souborů, které se stáhnou při příkazu *npm install*. Takže tento obsah představuje závislosti a závislosti závislostí, které nepotřebujeme mít v repozitáři. Další složka je *@types*, která obsahuje jenom jeden soubor, který rozšiřuje interface *SessionData* o atribut *userId* (více o sessions v kapitole 5.6.3.). Jako poslední složka je tu *src* a ta obsahuje další podsložky a veškerou logiku backendu. Jedna podsložka je například *config* ve které se definuje připojování k databázi.

4.1.1. app.ts

Jediným souborem v složce *src* je soubor *app.ts*, který představuje samotné jádro backendu. V tomto souboru se například volá funkce pro připojení k databázi, konfiguruje se soubor *.env*, registrují se endpointy (*Obr. 4*), spravují se sessions, spouští se vlastní aplikace a mnoho dalšího. Takže jednoduše řečeno se v této složce definuje, konfiguruje a volá úplně všechno co se týče backendu.

```
// registering routes
app.use('/api/ads', adRoutes)
app.use('/api/users', userRoutes)
app.use('/api/forum', forumRoutes)
```

Obrázek 4 - registrace skupin endpointů v *app.ts*

4.1.2. routes

Ve složce *routes* jsou soubory, které obsahují logiku pro jednotlivé endpointy. Každý soubor je přiřazen k jedné skupině endpointů (*Obr. 4*), které obsluhuje. Tím pádem, jelikož máme tři skupiny endpointů (ads, users, forum) máme taky tři soubory ve složce *routes*.

Takový *routes* soubor zpravidla obsahuje: import logických funkcí, které vykonávají nějaké akce a následně jejich přiřazení ke konkrétním endpointům podle toho co chci aby se stalo při zaslání requestu (žádosti) na daný endpoint (*Obr. 5*). Žádost má typ podle důvodu zaslání z frontendu - hlavní typy jsou GET (získání dat), POST (zaslání dat), PUT (zaslání a aktualizace dat) a DELETE (smazání dat). Endpoint může obsahovat parametry ke kterým mají přiřazené funkce přístup, v url se značí dvojtečkou : před názvem parametru (*Obr. 5*).

```

// import controllers
import { getAds, createAd, updateAd, saveAd, deleteAd } from '../controllers/ads.controller'

// api routes for interacting with ads
router.get('/', getAds)
router.post('/', createAd)
router.get('/:id', getAd)
router.put('/:id', updateAd)
router.delete('/:id', deleteAd)
router.post("/:userId/save-ad/:adId", saveAd)

```

Obrázek 5 - registrace konkrétních endpointů ve složce *routes*

Všechny tři soubory ve složce *routes* jsou vcelku analogické. Konkrétně v *ads.route.ts* regisuруji endpointy pro získání všech nebo konkrétního inzerátu z databáze, zveřejnění, úpravu, smazání inzerátu a uložení inzerátu (*Obr. 5*). Soubor *users.route.ts* má v sobě definici endpointů pro registraci, přihlášení, odhlášení a získání aktuálního uživatele. Ve *forum.route.ts* se regisuруje zveřejňování a mazání příspěvků a získání všech příspěvků nebo konkrétně spadajících pod město.

4.1.3. controllers

Stejně jako u souborů ve složce *routes* tak i v této složce každý soubor reprezentuje logické operace pro jednu konkrétní skupinu endpointů. Takže i tato složka obsahuje tři soubory s logikou pro obsluhu requestů zvlášť pro ads, users a forum.

Soubor představující controller obsahuje deklarace funkcí, které jsou po importu ve složce *routes* přiřazeny ke konkrétním endpointům. Každá funkce v takovém souboru se chová jako odpověď na request od frontendu - to znamená, že vykoná nějaké operace, které vyústí v odeslání response (odpovědi) na frontend (uživateli). V tělu funkce většinou probíhají různé kontroly správnosti uživatelského requestu.

Jak již bylo zmíněno v předešlé kapitole, tak requesty mají různé typy, response takové typy nemají, ale podle toho na jaký typ requestu odpovídají se dá zpravidla říct, jak budou vypadat. Response na GET request (například uživatel chce získat konkrétní inzerát z databáze (*Obr. 6*)) bude vždy posílat na frontend nějaké info z databáze o které bylo požádáno. Response na POST request (například zveřejnění příspěvku (*Obr. 7*)) udělá nový záznam do databáze a může a nebo nemusí posílat info na frontend (v mé projektu vždy zasílám proměnnou *success*, která dá uživateli vědět, zda se akce podařila). Podobně jako u POST requestů se bude odpovídat na PUT requesty, udělá se záznam do databáze, který však něco přepisuje, takže nepřidává nic nového a potom může poslat info na frontend. U DELETE requestů se pracuje opět docela podobně jako u POST a PUT, zapíše se do databáze, kde však ten zápis smaže nějaká existující data, odpověď na frontend není klíčová. V mé programu jsem používala všechny tyto čtyři typy requestů a u všech jsem měla response, které vraceley JSON data mezi kterými vždy byla proměnná *success* skrz kterou se frontend o úspěšnosti requestu.

```
// fetches a single ad
// @route GET /api/ads/:id
export const getAd: RequestHandler = asyncHandler(async (req: Request, res: Response) => {
  try {
    // get ID that is part of the url
    const { id } = req.params

    // if ad with such ID does not exist, send that info to frontend
    if (!mongoose.Types.ObjectId.isValid(id)) {
      res.status(404).json({ success: false, message: 'Inzerát se zadáným ID neexistuje' })
      return
    }

    // find the particular ad
    const ad = await Ad.findById({_id: id})

    if(ad){
      // success, send data to frontend
      res.json({ success: true, data: ad })
      return
    }
  } catch (error) {
    // if anything goes wrong it is server's fault - status 500 is sent to frontend
    res.status(500).json({ success: false, message: 'Při načítání inzerátu nastala chyba' })
  }
})
```

Obrázek 6 - controller funkce pro získání specifického inzerátu z databáze

```

// creates a post
// @route POST /api/forum
export const createPost = async (req: Request, res: Response): Promise<void> => {
  try {
    // extract ad data from the request body
    const { city, title, text } = req.body
    // retrieve the authenticated user's ID from the session
    const authenticatedUserId = req.session.userId
    // find particular user by id, including email, returns a promise
    const user = await User.findById(authenticatedUserId).select('+email').exec()

    // make sure that the required fields are filled out
    if (!city || !text) {
      res.status(400).json({ success: false, message: 'Město a text jsou potřeba vyplnit.' })
      return
    }

    // create a new post instance with the provided data
    const newPost = new Post({ city, title, text, full_name: `${user?.first_name} ${user?.last_name}`, user: authenticatedUserId })

    // save the post to the database
    const savedPost = await newPost.save()
    await User.findByIdAndUpdate( user?._id, { $push: { posts: newPost._id as string } }, { new: true })

    // if succesful let frontend know
    if(savedPost){
      res.status(201).json({success: true, data: savedPost})
    }
  } catch (err) {
    // if unsuccesful let frontend know
    res.status(500).json({ success: false, message: 'Nastala chyba při tvorbě příspěvku.' })
  }
}

```

Obrázek 7 - controller funkce pro vytváření příspěvků

Jak jsem již zmínila, tak složka *routes* mého projektu obsahuje tři soubory každý týkající se vlastní skupiny a každý z nich popisuje logiku u všech možných operací dané skupiny. Například soubor *ads.controller.ts* zajišťuje zveřejnění, mazání, úpravu, uložení a zobrazení jednoho (Obr.6) i více inzerátů. Soubor *forum.controller.ads* obsahuje analogické funkce (Obr. 7) a v *users.controller.ts* se zajišťuje registrace, přihlášení, odhlášení, zobrazení profilu a získání aktuálního uživatele.

4.1.4. models

Složka *models* v MERN Stack aplikací zajišťuje správnou komunikaci s databází tím, že se v ní definují schémata objektů, které se do databáze ukládají. Můj projekt má tři typy objektů (čtyři se sessions, více o sessions v kapitole 5.6.3.), které se do databáze ukládají a to jsou *Ad*, *Post* a *User*.

V takovém *models* souboru se tedy do schémata upřesní typy a nezbytnost atributů a potom se do databáze registrují. Normálně by toto stačilo, ale jelikož je moje práce psaná v typescriptu musela jsem u všech modelů definovat Interface, který zdědil vlastnosti MongoDB dokumentu.

Nejrozsáhlejší model je v mého projektu *Ad* v souboru *ads.model.ts*, který má mnoho atributů a spoustu z nich nepovinnou (Obr. 8), jelikož jsem chtěla dát uživatelům, kteří si inzerát vytváří volnost si to udělat jak chtějí.

```
// mongoose schema for ads
const adSchema = new Schema<IAd>(
  {
    title: { type: String, required: true },
    description: { type: String, required: true },
    phone: { type: String },
    destination: { type: String },
    date: { type: String },
    preferences: {
      gender: { type: String },
      minAge: { type: String },
      maxAge: { type: String },
      languages: [String],
      smokingPreference: { type: String },
    },
    full_name: { type: String, required: true },
    email: { type: String, required: true },
    user_age: { type: Number, required: true },
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  },
  { timestamps: true }
)
```

Obrázek 8 - mongoose schéma pro cestovní inzerát ve složce *models*

Moje další dva modely představují příspěvek na komunitním fóru a uživatele. U uživatele jsou všechny atributy povinné.

4.2. Frontend

Když je řeč o frontendu tohoto projektu hovoří se o obsahu složky *client*. Ta se zaslouží o vygenerování toho, co uživatel vidí když spustí aplikaci a taky za komunikaci s backendem posíláním requestů. A jelikož se nyní pohybujeme v Reactu, tak narozdíl od backendu se zde nevyužije pouze jeden jazyk TypeScript, ale i více jako JavaScript XML (psaní html v javascriptu), HTML a CSS.

Ve frontend složce se analogicky nachází nezbytné *JSON* konfigurační soubory, *.gitignore*, *node_modules* složka se závislostmi a *.env*, který jsme si vytvořili. Dále je tu složka *public* ve které jsou uloženy uživatelem dostupné soubory, což znamená HTML kód stránky, obrázek malé ikonky na kartě (favicon) a textové soubory s informacemi o českých městech.

Další podsložkou *clienta* je složka *src* (zkratka pro source), kam se ukládá veškeré logika frontendu. V té je ještě řada dodatečných složek a souborů, například složky *styles* do které se ukládají CSS soubory nebo *assets* kde jsou obrázky co napříč frontendem používám. Ostatní složky jsou komplikovanější a tak mají vlastní kapitolu. Dále je tu soubor *react-scripts*, který se stará o to aby TypeScript správně fungoval.

4.2.1. index.tsx

V React programech slouží *index.tsx* (*.jsx*) soubory jako vstupní body aplikace a nachází se zde kód, který renderuje *root* aplikace, tedy úplně všechno co na obrazovce vidíme. Kromě samotné aplikace z *App.tsx* (*.jsx*) se zde importují další moduly, kterými se aplikace může takzvaně “obalit” a k jejím funkcím je pak přístup ze všech částí webu (v méém případě *Authprovider* a *BrowserRouter*). Takto definovaný root se naváže k *<div>* značce v souboru *index.html* a aplikace se tím zobrazí. [5]

4.2.2. App.tsx

Jak je podle jména zřejmé, tak v souboru *App.tsx* (*.jsx*) se renderuje vlastní aplikace. Tím pádem, jestli je například nějaká komponenta, u které chceme aby byla nadřazená veškerému ostatnímu obsahu (v mého projektu to jsou komponenty pro navbar, potvrzení odhlášení a footer), tak se renderuje právě tady.

Kromě k vizualizaci slouží tento soubor k registraci adres webu. Toto už nejsou endpointy o kterých nemá běžný uživatel ponětí, ale přímo url adresy webové stránky, která je vidět a dá se přepisovat na horní liště. Ty se registrují za pomocí implementace *BrowserRouter* z balíčku *react-router-dom*, kterým je aplikace v *index.tsx* obalená. Adresa se definuje tak, že se nejdřív určí jak bude vypadat url do proměnné *path* a do *element* se vloží *tsx* (*jsx*) soubor ve kterém se renderuje to, co chceme na konkrétní adrese vidět (*Obr. 9*). V případě, že by měl být přístup na stránku nějakým způsobem omezen (například na stránku s uživatelským profilem mohou jen přihlášení) jsem použila mnou vytvořenou komponentu *PrivateRoute*, která před renderováním obsahu stránky zkонтroluje to, co je potřeba (více o *PrivateRoute* v kapitole *4.2.4.*).

```
{/* page for displaying all ads */}
<Route path='/inzeraty' element={<AdsPage />} />
{/* page for posting ads */}
{/* protected -> not logged-in users are redirected to '/prihlaseni' */}
<Route path='/inzeraty/zverejnit' element={<PrivateRoute redirectTo='/prihlaseni'><CreateAdPage /></PrivateRoute>} />
{/* page for viewing a particular ad */}
<Route path='/inzeraty/:id' element={<ViewAdPage />} />
{/* page for modifying a particular ad */}
{/* protected -> only owner can edit the ad, everyone else is sent back to viewing the particular ad */}
<Route path='/inzeraty/upravit/:id' element={<PrivateRoute redirectTo='/inzeraty'><UpdateAdPage /></PrivateRoute>} />
```

Obrázek 9 - registrace webových adres v souboru *App.tsx*

4.2.3. pages

Ve složce *pages* se vyskytují *tsx* soubory, každý představuje část webové aplikace, kterou si uživatel zobrazí při přechodu na konkrétní url. Tedy jsou to ty soubory, které jsou v souboru *App.tsx* přiděleny k jednotlivým adresám webu. Přesto, že má ČundrAppka čtyři hlavní funkce, tak jsou zde pouze tři podsložky jich se týkající ve kterých se shromažďují soubory po skupinách podle oblasti stránky ke kterým se vztahují (inzeráty, komunitní fórum a cestovní balíčky). Funkcionalita mapa obsazuje jenom jednu stránku, protože se jedná o pouhé zobrazení mapy, tím pádem se soubor *MapPage.tsx* vyskytuje přímo ve složce *pages*. Kromě tohoto souboru jsou tu ještě další dva: *HomePage.tsx*, která se zobrazí při spuštění aplikace (tedy je na adrese `/`) a *NotFoundPage.tsx*, která je zobrazena, že uživatel manuálně přepisuje url stránky a dostane se někam, kde nic není. Pak je tam ještě složka *users* se stránkami, které se týkají samotného uživatele.

Takový soubor představující stránku, tedy soubor typu *jsx* nebo v méém případě *tsx* obsahuje funkci, která generuje obsah konkrétní stránky. Zpravidla je v ní příkaz *return* do kterého se píše kód v jazyce JSX, tedy kombinace javascriptu a html. Mimo to je obvyklé ve funkci mít jiné akce, například získání nějakých dat z backendu, které jsou následně vloženy do *return* funkce a zobrazeny v prohlížeči.

Komunikaci s backendem skrz tyto soubory jsem naprogramovala za pomocí takzvaných React Hooků, což jsou metody, které jsou definovány ve složce *hooks* a k jejím použití stačí jenom import. Takto jsem zamezila DRY a udělala svůj projekt přehlednějším.

V případě, že soubor operuje na adresu, která obsahuje parametr, tak k němu má přístup za pomocí hook funkce *useParams()* z balíčku *react-router*.

4.2.3.1. ads

Složka *ads* obsahuje soubory, které generují obsah stránek týkajících se cestovních inzerátů. Takové stránky jsou následující: */inzeraty*, */inzeraty/zverejnit*, */inzeraty/:id*, */inzeraty/:id/upravit*, */muj-ucet/moje-inzeraty* a */muj-ucet/ulozene-inzeraty*. Ve všech těchto souborech využívám mnou vytvořené hooky pro komunikaci s backendem, někdy stačí data jednoduše získat a jindy se musí ještě logicky upravit - například při zobrazování inzerátů, jejíž jsem autor (*Obr. 10*).

```
// fetch all ads and loading state
const { ads, loading: loadingAds } = useFetchAds()
// access user data and loading state from authcontext
const { user, loading: loadingUser } = useAuth()
// states for storing search query and sort order
const [searchQuery, setSearchQuery] = useState('')
const [sortOrder, setSortOrder] = useState('newest')

// when loading, display loading circle
if (loadingAds || loadingUser) return <LoadingCircle/>

// retrieve field of user-owned ad IDs
const myAdsIds = user?.ads || []
// filter displayed ads to only contain the one's user has created
const myAds = ads.filter(ad => myAdsIds.includes(ad._id))
```

Obrázek 10 - logická úprava dat získaných z backendu v *MyAdsPage.tsx*

```
<div className="ads-container">
  {myAds.length > 0 ? (
    // render each ad
    myAds.map((ad, index) => (
      <div key={index} className="vintage-paper-box">
        <h2>{ad.title}</h2>
        <p className="ad-description">{ad.description}</p>
        <div className="ad-footer">
          <a href={`/inzeraty/${ad._id}`}>Zobrazit</a>
          <p>Vytvořeno: {new Date(ad.createdAt).toLocaleDateString('cs-CZ')}
        </div>
      </div>
    ))
  ) : (
    // if user hasn't posted any ads
    <p className='no-ads'>Nezveřejnil jsi žádné inzeráty.</p>
  )
</div>
```

Obrázek 11 - část funkce *return* v *MyAdsPage.tsx*

4.2.3.2. community_forum

Ve složce *community_forum* jsou soubory týkajících se komunitního fóra. Tyto soubory generují zobrazený obsah pro následující adresy: */komunitni-forum*, */komunitni-forum/zverejnit*, */komunitni-forum/:city*, */komunitni-forum/:city/zverejnit* a */muj-ucet/moje-prispevky*. S tím, že obsah adresy pro zveřejňování příspěvku na komunitním fóru */komunitni-forum/zverejnit* a */komunitni-forum/:city/zverejnit* renderuje neobvykle jenom jeden soubor *ForumPostingPage.tsx*, který za pomocí *useParams()* rozlišuje, zda dostal parametr *city* a podle toho upraví formulář, který uživatel při tvorbě příspěvku vyplňuje (*Obr. 12*).

```
{/* if there's no city in url user has to select */}
{!city && (
  <select
    name="city"
    className="forum-select"
    onChange={(e) => setpostData({ ...postData, city: e.target.value })}
    required
  >
    {Object.entries(cities).map(([key, value]) => (
      <option key={key} value={key}>
        {value}
      </option>
    )))
  </select>
)}
```

Obrázek 12 - část funkce *return* v *ForumPostingPage.tsx*

4.2.3.3. travel_packages

Soubory, které renderují obsah stránek a týkají se cestovních balíčků jsou uloženy ve složce *travel_packages*. Jsou tu tedy jenom dva a to pro adresy */cestovni-balicky* a */cestovni-balicky/:city*. Tím pádem má tato funkcionality programu jenom dvě adresy a to pro vykreslení menu s cestovními balíčky deseti největších českých měst a zobrazování samotných informací o konkrétním městě. Data k jednotlivým městům jsou vyčteny z veřejně dostupných textových souborů uložených ve složce *client/public/cities_info* (Obr. 13). Jelikož mají všechny tyto textové soubory stejné sekce s informacemi - web, historie, zeměpis, turistické zajímavosti, kultura, restaurace a ubytování, výlety v okolí - tak jejich text podle nich rozdělím abych ušetřila kód při zobrazování dat v *return* funkci (Obr. 14).

```
// once the page is loaded useEffect is called and reads out the contents of particular city txt file
useEffect(() => {
  const path = `/cities_info/${city}.txt`
  fetch(path)
    .then(response => {
      if (!response.ok) {
        throw new Error("Zadané město nemáme ve výběru")
      }
      return response.text()
    })
    .then(data => {
      setText(data)
      setLoading(false)
    })
    .catch(error => { throw new Error("Nastal error při načítání stránky: ", error) })
}, [city]) // re-runs when city is changed
```

Obrázek 13 - čtení textového souboru v *ViewPackagePage.tsx*

```
// split by lines and then process each line
const lines = text.split('\n')
lines.forEach(line => {
  const matchedSection = sections.find(section => line.startsWith(section))
  if (matchedSection) {
    // if we find a section, save the previous one and start new one
    if (currentSection) {
      sectionData[currentSection] = currentContent.trim()
    }
    currentSection = matchedSection
    // clean up the section header
    currentContent = line.replace(matchedSection, '').trim()
  } else {
    currentContent += `\n${line}`
  }
}

// add the last section to the result
if (currentSection) {
  sectionData[currentSection] = currentContent.trim()
}
return sectionData
```

Obrázek 14 - ukázka z funkce na formátování textu na sekce v *ViewPackagePage.tsx*

4.2.3.4. users

Uživatelská autentizace a autorizace nejsou klíčovými funkcemi ČundrAppky, ale díky její povaze jsou nepostradatelné. O tom jak vypadají stránky pro přihlášení, registraci a zobrazení uživatelského profilu se starají soubory obsáhlé ve složce *users*. V souborech pro přihlášení a registraci *LoginUserPage.tsx* a *RegisterUserPage.tsx* jsou formuláře, jejíž vyplněné informace uživatelem se po kliknutím na tlačítko pošlou společně s requestem na backend. O to se v obou případech starají hook funkce importované ze složky *hooks*. V případě registrace je uživatelovi oznámeno, když si vybere username nebo email se kterým již účet existuje (*Obr. 15*). Při přihlašování je mu oznámeno když použije email nebo username se kterým účet neexistuje a nesprávné heslo. Poslední soubor *UserProfilePage.tsx* poskytuje náhled do uživatelských informací a možnost se nechat přesměrovat na zobrazení svých inzerátů a komunitních příspěvků a uložených inzerátů.

```
// execute after submitting
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault()

  // retrieve success and message from backend
  const { success, message } = await registerUser(userData as User)
  if (success) {
    // navigate to 'prihlaseni' so the user can log-in
    navigate('/prihlaseni')
  } else {
    // if anything went wrong there will be displayed error message (for example 'username already taken')
    alert(message)
  }
}
```

Obrázek 15 - chování programu po odeslání žádosti o registraci v *RegisterUserPage.tsx*

4.2.4. components

Složka *components* obsahuje *tsx* soubory, které vykreslují jednotlivé komponenty, tedy menší části aplikace. Důvod pro vytvoření komponenty je zpravidla DRY a přehlednost projektu. Skvělým příkladem vyhnutí se zbytečnému opakování je moje komponenta *LoadingCircle.tsx*, kterou využívám takřka v každém souboru ve složce *pages* když čeká na odpověď z backendu. Další komponenty jsou *Navbar.tsx*, *Footer.tsx* a *LogoutConfirmComp*, které importuju v souboru *App.tsx* a tím ho udělám jasnější a bez zbytečného zahlcení výchozího souboru aplikace, který by měl být přehledný. Komponenta *DeleteConfirmComp* se importuje do souborů, kde se aplikace chce ujistit, zda uživatel chce opravdu něco smazat (inzerát nebo příspěvek).

Jak již bylo zmíněno v kapitole 4.2.2., tak tato složka obsahuje i soubor *PrivateRoute.tsx*, jenž je importovaný v *App.tsx* a “ochraňuje” určité části webu před neautorizovaným přístupem. Tato komponenta je tedy používaná v případě, když se uživatel snaží dostat na stránku s limitovaným přístupem (například */muj-ucet*, */registrace* nebo */inzeraty/upravit/:id*). Jako jeden z parametrů, který od *App.tsx* dostane je *redirectTo* a to upřesní jaký typ logické kontroly je potřeba provést (uživatelská autorizace nebo ověření vlastnictví inzerátu). Na Obrázku 16 je vidět část z přiřazení kontroly ve *switch* podle obsahu proměnné *redirectTo*.

```

// example usage: trying to access 'login' when logged-in
case '/muj-ucet': {
  // if user is logged-in redirect to /muj-ucet
  if (user) {
    return <Navigate to='/muj-ucet' />
  }
  // if user isn't logged-in let him access the protected page
  break
}

// example usage: trying to edit an ad that is not user's
case '/inzeraty': {
  // retrieve an array of user-owned ads IDs
  const userAds = user?.ads || []
  // is the ID from url part of this array?
  const isMine = userAds.includes(id as string)
  // if ad is not user's redirect to the page that just views the specific ad
  if (!isMine) {
    return <Navigate to={`/inzeraty/${id}`} />
  }
  // if user isn't logged-in let him access the protected page
  break
}

```

Obrázek 16 - část kódu pro autorizaci uživatelských dat v *PrivateRoute.tsx*

4.2.5. hooks

React Hooks jsou speciální funkce, které usnadňují práci s Reactem – pomáhají například spravovat stav (paměť) komponenty nebo reagovat na změny v aplikaci. Dříve bylo potřeba používat složité třídy, ale dnes díky hookům můžeme vše dělat přímo v jednoduchých funkcích. Nejčastěji používané hooky jsou *useState* (pro uchování a změnu dat) a *useEffect* (pro reakce na změny, třeba načtení dat z internetu). Kromě těchto vestavěných hooků si můžeme vytvořit i vlastní, a to jsem ve svém projektu udělala právě ve složce *hooks*. Důvod k vytvoření vlastních hooků je když chceme opakovaně používat určitou logiku v různých částech aplikace - v mé případě vlastní hook pro práci s API. Díky hookům je kód kratší, přehlednější a snadněji udržovatelný. [6]

Tato složka má tři další podsložky pojmenované podle toho, které části aplikace se věnují - *ads*, *forum* a *users*. V každé této složce je několik *ts* souborů ve kterých definuju posílání jednotlivých requestů na backend. Soubor se vždy jmenuje podle účelu requestu - například *useDeleteAd.ts* se stará o smazání cestovního inzerátu.

Všechny tyto hooky fungují tak, že pošlou request v korektní formě na konkrétní API adresu na které backend poslouchá za stejným účelem se kterým je request odeslán. Potom backend provede jisté akce a frontendu nazpátek v response pošle status úspěšnosti akce a jiná data o která si frontend mohl požádat.

Všechny moje hooky nemají jednotnou architekturu, používala jsem jednu pro requesty typu GET (*Obr. 17*) a druhou pro typy POST, DELETE a PUT (*Obr. 18*). Toto jsem udělala proto, že u requestů typu GET se očekává response s jistými objekty o které si frontend žádá a taky se tyto hooky volají podmíněně. To znamená, že využívám vestavěné React hooky *useState* a *useEffect*. Ostatní hooky, které posílají requesty jiného typu než GET jsem naprogramovala jako asynchronní funkci, která je hookem exportovaná a uživatel si ji sám volá. Avšak jedna věc je pro všechny soubory ve složce *hooks* jednotná a to to, že backend na všechny endpointy jako součástí response vždy pošle boolean proměnnou *success* a její hodnota určuje, zda byla požadovaná akce úspěšná.

Jak jsem již zmínila, jsou zde tři podložky pro cestovní inzeráty, komunitní fórum a uživatelé. Jelikož jsou soubory v nich ve většině případech analogické, nebudu jednotlivé složky rozdělovat do kapitol. Funkcionalita mapa a cestovní balíčky nemají své hooky, jelikož u nich není zapotřebí komunikovat s backendem - mapy jsou přes API a balíčky z textových souborů v *public* složce.

```

// hook for fetching an ad
const useFetchSingleAd = (id: string) => {
  // states for storing fetched ad and loading situation
  const [ad, setAd] = useState<Ad | null>(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    // make request on an endpoint containing ad's ID where backend is listening
    fetch(`api/ads/${id}`, { method: 'GET' })
      .then(res => {
        // if something went wrong stop the loading and return
        if (!res.ok) {
          setLoading(false)
          return
        }
        // if response is ok, return it for further logic
        return res.json()
      })
      // returned data consists of attributes success and fetched ad
      .then((data: { success: boolean, data: Ad }) => {
        // if fetch was successful save ad into state
        if (data.success) {
          setAd(data.data)
        }
        setLoading(false)
      })
      // if error occurred (for example wrong ID) set loading to false (frontend handles rest)
      .catch(() => {
        setLoading(false)
      })
  }, [id])

  // hook returns fetched ad and loading state
  return { ad, loading }
}

```

Obrázek 17 - hook funkce posílající API request typu GET v *useFetchSingleAd.ts*

```

// hook for creating a post
const useCreatePost = () => {
  // function that is later imported where needed
  const createPost = async (newPost: Post) => {
    try {
      // make request on an endpoint where backend is listening
      const response = await fetch('/api/forum', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        // body containing an ad for creation
        body: JSON.stringify(newPost),
      })

      // retrieve a response containing success status
      const data = await response.json()
      // return the outcome
      return { success: data.success }
    } catch {
      // if failed, return unsuccess
      return { success: false }
    }
  }

  // hook returns the function declared above
  return { createPost }
}

```

Obrázek 18 - hook funkce posílající API request typu POST v *useCreatePost.ts*

4.2.6. context

Do složky *context* se v MERN aplikacích dávají soubory poskytující informace a data, ke kterým chceme mít přístup globálně ze všech částí webu. Vémém případě mám ve složce pouze soubor *AuthContext.tsx*, ale můžou zde být i další například *AlertContext.tsx* pro jednotný alert nebo *ThemeContext.tsx* pro barevný motiv aplikace. Takovými contexty se potom v souboru *index.tsx* (*.jsx*) "obalí" komponent aplikace *<App />*.

Jak jsem již zmínila, tak v této složce mám jenom jeden soubor *AuthContext.tsx* a ten celé aplikaci globálně zprostředkovává uživatelův aktuální stav autorizace. To znamená, že v případě, že potřebuji vědět, zda je uživatel přihlášený, případně jeho informace, tak si data přečtu z tohoto contextu (Obr. 19). Těchto informací dosáhne importováním hooků z *hooks/users*.

```
// access user and loading state from auth context
const { user, loading } = useAuth()
```

Obrázek 19 - získání uživatelských dat za pomocí *AuthContext.tsx*

```
export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
  // import necessary hooks
  const { user, loading, fetchUser } = useFetchUser()
  const { logoutUser } = useLogoutUser()

  const logout = async () => {
    await logoutUser()
    await fetchUser() // clear user state after logout
  }

  // wrapped around children so it provides info to all of them
  return (
    <AuthContext.Provider value={{ user, loading, fetchUser, logout }}>
      {children}
    </AuthContext.Provider>
  )
}
```

Obrázek 20 - ukázka kódu z *AuthContext.tsx*

4.2.7. models

Složka *models* je poslední podsložkou *src* a jak už jméno napovídá se ní vyskytují soubory ve kterých jsou definované modely objektů, které ve svém projektu používat - to znamená *Ad* (cestovní inzerát), *Post* (příspěvek na fóru) a *User* (uživatel). Této složky je potřeba kvůli tomu, že píši ve TypeScriptu, kde je type-checking, tím pádem musím mít pevně dané interfaces objektů se kterými pracuji.

Jelikož se všemi modely z této složky také pracuju s databází u všech je definován atribut *_id*. Také se u každého z nich vyskytuje alespoň jeden nepovinný atribut (značí se ? za názvem) - to znamená, že si můžu vytvořit objekt daného typu bez toho abych musela vyplňovat tento nepovinný údaj, všechny ostatní jsou ale při tvoření objektu povinné.

Intefaces objektů se kterými pracuji v databázi by na frontendu měly mít stejné atributy. Ještě je v pořádku, když jich je méně (způsobí jen to, že se na frontendu k údaji nedostanu), ale nikdy by v modelech na frontendu neměl být definovaný takový atribut, který není v backend modelech. Jedinou výjimkou je *_id* a to proto, že na backendu představuje model dokument v MongoDB a tomu je automaticky údaj *_id* přiřazen, tím pádem se nemusí deklarovat naopak, by to působilo zmatky.

```
export interface Ad {
  _id: string
  title: string
  description: string
  phone?: string // optional
  destination?: string // optional
  date?: string // optional
  preferences?: { // optional
    gender?: string // optional
    minAge?: string // optional
    maxAge?: string // optional
    languages?: string[] // optional
    smokingPreference?: string // optional
  }
  createdAt: string
  updatedAt: string
  user: string
  full_name: string
  email: string
  user_age: number
}
```

Obrázek 21 - deklarace interfacu pro inzerát v *ad.ts*

```
export interface User {
  _id: string
  username: string
  first_name: string
  last_name: string
  birthday: Date,
  age?: number, // optional since it is calculated at backend
  email: string
  password: string
  ads?: string[] // optional
  saved_ads?: string[] // optional
  posts?: string[] // optional
}
```

Obrázek 22 - deklarace interfacu pro uživatele v *user.ts*

5. Implementace klíčových funkcí

Při vymýšlení zadání mého projektu jsem se snažila dát dohromady takové funkce, které by byly nejvíce užitečné a zároveň jsem je dokázala s omezeným časem a schopnostmi naprogramovat. Bylo mi jasné, že chci spojovat lidi, takže inzeráty byly první funkce projektu, kterou jsem si napsala na seznam. Také mi přišlo užitečné, kdyby uživatelé měli přístup k aktuálním informacím o dění ve městech do kterých se potenciálně chtějí vydat - tak vzniklo komunitní fórum. Dále mi přišlo vcelku logické zahrnout mapu a turistické balíčky s informacemi, když se jedná o cestovatelský web.

K implementaci mi postačily technologie pro které jsem se původně rozhodla, takže to "jediné" co mi stálo v cestě bylo naučit se programovat s MERN Stackem a TypeScript.

5.2. Inzeráty

Idea inzerátů v ČundrAppce je taková, že má vypomoci lidem, kteří by rádi cestovali, ale nemají s kým. Na webu mají možnost buďto zveřejnit inzerát a více si specifikovat kam a s kým by rádi putovali nebo si za pomocí rozsáhlých filtrů najít ten perfektní inzerát od někoho jiného a následně inzerenta kontaktovat. Kromě zveřejňování a zobrazování inzerátů má uživatel možnost ty svoje i po zveřejnění upravovat a mazat a ty které mu připadají sympatické si k sobě uložit na profil. Při prohledávání inzerátů je taky zobrazené datum kdy byl vytvořen a naposledy aktualizován aby si uživatel ujistil, že je stále platný a zda má cenu inzerenta kontaktovat. Na celém webu je samozřejmě ošetřena bezpečnost, takže úprava inzerátu je možná jedině pro vlastníka a zveřejňování je znemožněno nepřihlášeným.

5.2.1. Reprezentace v databázi

Reprezentace inzerátu v databázi je zajištěna v souboru *ads.model.ts*, kde se nejen definuje, ale i registruje do MongoDB databáze. Každý objekt tohoto typu *Ad* má určité povinné atributy ze kterých uživatel vyplňuje pouze nadpis a popis. Ty ostatní nezbytné se vyplní sami při tvorbě - to znamená uživatel jeho věk a celé jméno. Mezi nepovinné patří například destinace, doba výletu nebo různé osobní preference na společníka. U každého nově vytvořeného inzerátu se k němu do databáze přidají dva atributy "timestamps", které představují datum a čas vytvoření a poslední úpravy. (*Obr. 8*) Jelikož se jedná o databázi MongoDB, tak má každý dokument databázi atribut *_id* podle kterého je mezi sebou identifikujeme.

```
// mongoose schema for ads
const adSchema = new Schema<IAd>(
  {
    title: { type: String, required: true },
    description: { type: String, required: true },
    phone: { type: String },
    destination: { type: String },
    date: { type: String },
    preferences: {
      gender: { type: String },
      minAge: { type: String },
      maxAge: { type: String },
      languages: [String],
      smokingPreference: { type: String },
    },
    full_name: { type: String, required: true },
    email: { type: String, required: true },
    user_age: { type: Number, required: true },
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  },
  { timestamps: true }
```

Obrázek 8 - mongoose schéma pro cestovní inzerát ve složce *models*

5.2.2. API

V souboru *ads.route.ts* je pro inzeráty rezervované dohromady šest endpointů na kterých backend poslouchá. K těmto adresám jsou přiřazeny funkce, které se exekutují jakmile na url poslán request a provede určitou logiku. Jedná se logiku, která komunikuje s databází, to znamená, že buďto z ní získává nějaké info nebo do ní zapisuje. V případě inzerátu jsou možné následující akce s databází: získání všech inzerátu, tvorba inzerátu, získání konkrétního inzerátu, úprava inzerátu, smazání inzerátu a uložení inzerátu (*Obr. 5.*).

```
// import controllers
import { getAds, createAd, getAd, updateAd, saveAd, deleteAd } from '../controllers/ads.controller'

// api routes for interacting with ads
router.get('/', getAds)
router.post('/', createAd)
router.get('/:id', getAd)
router.put('/:id', updateAd)
router.delete('/:id', deleteAd)
router.post("/:userId/save-ad/:adId", saveAd)
```

Obrázek 5 - registrace konkrétních endpointů ve složce *routes*

Funkce, které jsou k jednotlivým endpointům přiřazeny se importují ze souboru *ads.controller.ts* a každá funkce provede jistou logiku a vrací response ve které se uvádí, zda-li byl její účel úspěšně splněn a popřípadě další data.

5.2.3. Uživatelské rozhraní

Jelikož má celá webová aplikace cestovatelský a lehce staromódní nádech chtěla jsem vyobrazit jednotlivé inzeráty jako kdyby to byly útržky zazloutlého papíru. Každopádně tohoto jsem nedokázala docílit úplně podle mých představ, ale i přesto je vzhled inzerátů velmi sympatický.

Na domovské stránce pro inzeráty se vyobrazí všechny inzeráty z databáze seřazené od nejnovějšího (*Obr. 23*), ale uživatel má možnost si pořadí změnit nebo vyplnit filtry aby se mu ukazovaly jenom inzeráty, které odpovídají jeho osobě (*Obr. 24*).



Obrázek 23 - vykreslení nerozkliknutých inzerátů

A screenshot of a search/filter interface. At the top, there is a search bar with "česko", a date selector set to "duben 2025", and a clear button. Below that, it says "Tvoje informace:" (Your information). Under "Tvoje informace:", there is a section for languages: "Mluvíš: Česky Španělsky Anglicky Rusky Italsky Německy Francouzsky". Below the language section are dropdown menus for gender ("Žena") and name ("Nekuřák") with the number "20" next to them.

Obrázek 24 - příklad vyfiltrování inzerátů

Po kliknutí na tlačítko zobrazit se uživateli přehledně vypíší všechny informace o inzerátu, ale i o inzerentovi. Uživatel má možnost si jakýkoliv inzerát uložit k sobě na profil a později se k nim vrátit. Pokud je vlastníkem inzerátu může ho upravovat (*Obr. 25*).



Obrázek 25 - rozkliknutý inzerát

Při tvorbě inzerátu je zapotřebí vyplnit pole pro název a popis, potom jsou všechna ostatní políčka dobrovolná. Toto dodává uživateli volnost si zveřejnit jakýkoliv inzerát chce podle jeho potřeby. Vzhled vytváření inzerátu je zcela stejný jako při úpravě již existujícího inzerátu s jediným rozdílem, že při úpravě se pole uživateli před-vyplní (*Obr. 26*).

Vytvoř si svůj inzerát

Název

Popis

Telefon

Destinace

Jakými jazyky mluvíš?

Čeština Španělština Angličtina Ruština
 Italština Němčina Francouzština

Preferované vlastnosti kamaráda na cestování:

Pohlaví

Minimální věk

Maximální věk

Kuřáctví

Zveřejnit inzerát

Zpátky

Obrázek 26 - tvorba inzerátu

Když si uživatel zobrazuje svoje uložené nebo vlastní inzeráty, tak vizuální rozhraní vypadá podobně jako při zobrazování všech inzerátů najednou. Jediný rozdíl je v tom, že místo filtrů je zde vyhledávací okénko do kterého uživatel zadá heslo díky kterému najde specifický inzerát rychleji.

5.3. Komunitní fórum

Účelem komunitního fóra je poskytování aktuálních informací z první ruky od lidí po celé republike. Fórum je rozdělené podle deseti největších českých měst konkrétně Praha, Brno, Ostrava, Plzeň, Liberec, Olomouc, České Budějovice, Hradec Králové, Zlín a Pardubice. Každé z těchto jednotlivých měst je možné si rozkliknout a pročíst si příspěvky od uživatelů. Prohlížení fóra je dostupné pro všechny, ale zveřejňovat může jen přihlášený uživatel. Ten si potom ve svém profilu může zobrazit všechny své příspěvky a popřípadě je smazat.

5.3.1. Reprezentace v databázi

Samotné komunitní fórum není objektem v databázi, za to ale jeho příspěvky jsou. Tento MongoDB dokument se v mému projektu nazývá *Post* a je deklarován a zaregistrován do databáze v souboru *forum-post.model.ts*. Takový objekt typu *Post* má od uživatele určitě vyplněný atribut *city*, který označuje ke kterému městu příspěvek patří, *text*, což je jeho obsah a nepovinně atribut *title*. Dále má tento objekt atributy “timestamps” (*createdAt*, *updatedAt*) a *full_name* s *user*, které mu vyplní sám backend dle uživatele při tvorbě příspěvku (Obr. 27). Jelikož se jedná o databázi MongoDB, tak má každý dokument databázi atribut *_id* podle kterého je mezi sebou identifikujeme.

```
// mongoose schema for posts
const postSchema = new Schema<IPost>(
{
    city: { type: String, required: true },
    title: { type: String },
    text: { type: String, required: true },
    full_name: { type: String, required: true },
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
},
{ timestamps: true }
)
```

Obrázek 27 - definice mongoose schéma a registrace příspěvku do databáze

5.3.2. API

Aby bylo možné získávat informace o příspěvcích z databáze od backendu je zapotřebí mít definované endpointy na kterých server poslouchá. Vém projektu jsou pro komunitní fórum rezervovány čtyři adresy a to konkrétně pro: tvorbu příspěvku, smazání příspěvku, získání všech příspěvků a získání příspěvků pod konkrétním městem (*Obr. 28*).

```
// import controllers
import { getCityPosts, createPost, getAllPosts, deletePost } from '../controllers/forum.controller'

// api routes for forum and interacting with posts
router.post('/', createPost)
router.delete('/:id', deletePost)
router.get('/posts', getAllPosts)
router.get('/posts/:city', getCityPosts)
```

Obrázek 28 - registrace endpointů pro komunitní fórum

Funkce, které jsou k jednotlivým endpointům přiřazeny se importují ze souboru *forum.controller.ts* a každá funkce provede jistou logiku a vrací response ve které se uvádí, zda-li byl její účel úspěšně splněn a popřípadě další data.

5.3.3. Uživatelské rozhraní

Když jsem si navrhovala jak bych chtěla aby vypadalo komunitní fórum ČundrAppky několikrát se mi vybavil Twitter. Chtěla jsem aby příspěvky byly jednoduché a aby jejich jediným úkolem bylo předat informaci bez jakéhokoliv vizuálního okouzlení.

Než uživatel přejde na konkrétní fórum nějakého města, tak je se mu zobrazí menu deseti různých možností podle toho, jaké město ho zajímá. Tlačítko pro přidání příspěvku se zobrazí pouze přihlášeným uživatelům. Ostatní vybídne k přihlášení (*Obr. 29*).



Obrázek 29 - vstupní bod komunitního fóra

Při rozkliknutí konkrétního města se vyrendrují všechny příspěvky, které pod město náleží seřazené od nejnovějšího. Je možné si je seřadit od nejstaršího nebo mezi nimi hledat kdyby uživatele zajímalo něco konkrétního. Analogicky tu je tlačítko pro přidání příspěvku se stejnou logiku pro viditelnost (*Obr. 30*).



Obrázek 30 - komunitní fórum Prahy

Na tvorbu příspěvku je tedy uživatel odkázán buďto z domovské stránky fóra a nebo z fóra pod konkrétním městem. Toto hraje velkou roli, protože stránky na které je z těchto dvou tlačítek odkázán nejsou stejné. V případě, že přijde z domovské stránky, tak je kromě titulku a textu příspěvku vyzván k vyplnění města pod které bude příspěvek spadat. Když si tvorbu příspěvku otevře z například pražského fóra tento atribut příspěvku bude pro uživatele sám předvyplněn (*Obr. 31*).

A form titled 'Zveřejnění příspěvku' (Publication of post). It has three input fields: a dropdown menu for 'Místo' (Location) set to 'Praha', a text input for 'Titulek' (Title), and a large text area for 'Text příspěvku...' (Post text). At the bottom are two buttons: a dark brown 'Zveřejnit' (Publish) button and a light brown 'Zpátky' (Back) button.

Obrázek 31 - tvorba příspěvku na komunitní fórum

Poslední akce spojená s komunitním fórem je zobrazování svých vlastních příspěvků. K tomu se uživatel dostane přes zobrazení svého profilu. Vykreslení těchto příspěvků je z většiny stejné jako u otevření nějakého konkrétního fóra s jediným rozdílem, že u každého příspěvku je tlačítko "smazat" a místo jména uživatele je zde název města pod kterým byl příspěvek zveřejněn.

5.4. Mapa

Funkce mapy mi přišla logická na zahrnutí do ČundrAppky, jelikož se jedná o webovou stránku zaměřenou na cestování. Při otevření stránky se mapa vycentruje na Českou republiku. Jelikož je implementovaná za pomocí Google Maps, tak dědí všechny její funkce, to znamená, že uživatel může využít například Street View (*Obr. 32*).



Obrázek 32 - zobrazení mapy na frontendu

5.4.1. API

Pro vyobrazení mapy projekt využívá službu Google Maps API, které vyžaduje platný API klíč (ten se musí bezpečně uložit - soubor *.env*). V případě, že vývojář klíč nezahrne do *.env* a nebo je neplatný, tak se mapa stále zobrazí, ale s watermarkem. Na backendu pro tuto funkci nemám vyhrazené adresy, protože není vůbec potřeba jakkoliv interagovat s databází.

```
<LoadScript googleMapsApiKey={apiKey} loadingElement={<LoadingCircle />}>
| <GoogleMap mapContainerClassName="map-box" center={center} zoom={7} />
</LoadScript>
```

Obrázek 33 - využití API klíče pro generování mapy na frontendu

5.5. Cestovní balíčky

Funkce cestovní balíčky shrnuje informace k deseti největším městům Prahy (k těm stejným deseti jako jsou u komunitního fóra). Každý balíček obsahuje ke každému městu následující informace: web, historie, zeměpis, turistické zajímavosti, kultura, restaurace, doprava a ubytování, výlety v okolí. Přístup k těmto mají i nepřihlášení uživatelé a jsou čerpány z textových souborů které jsou taky globálně dostupné.

5.5.1. Uživatelské rozhraní

Analogicky jako u komunitního fóra, když uživatel otevře stránku pro cestovní balíčky, tak si musí nejdříve vybrat o jaké město má zájem (*Obr. 34*).



Obrázek 34 - vstupní bod pro cestovní balíčky

Po vybrání města se zobrazí obsah textového souboru ve kterém se vyskytuje mnoho užitečných turistických informací. Kromě všeobecných informací cestovní balíček zprostředkovává odkaz na komunitní fórum toho města, jehož balíček si uživatel zobrazuje (*Obr. 35*).

PRAHA

[Přejít na KOMUNITNÍ FÓRUM](#)

WEB

WWW.PRAHA.EU

HISTORIE

Praha, hlavní město České republiky, byla založena v 9. století a v průběhu historie se stala sídlem českých panovníků. Největší rozkvět zažila za vlády Karla IV., kdy vzniklo Karlovo univerzita, Karlovy mosty a Nové Město. Po vzniku Československa v roce 1918 se stala hlavním městem nového státu a dnes je ekonomickým, politickým i kulturním centrem země s více než 1,3 milionu obyvatel.

ZEMĚPIS

Praha se nachází ve středních Čechách na řece Vltavě. Je obklopena kopci, z nichž nejznámější jsou Petřín, Vyšehrad a Letná. Díky své poloze byla vždy strategickým dopravním uzlem.

TURISTICKÉ ZAJÍMAVOSTI

Pražský hrad - největší hradní komplex na světě a sídlo prezidenta Karlova most - gotická památká spojující Malou Stranu a Staré Město Staroměstské náměstí a Orloj - srdce města s jedním z nejstarších orlojů Vyšehrad - pevnost s krásnými výhledy na město Židovská čtvrť (Josefov) - historická čtvrť s unikátními synagogami

KULTURA

Festival Pražské jaro - prestižní hudební festival klasické hudby Signal Festival - mezinárodní festival světelného umění Národní divadlo, Stavovské divadlo, Rudolfinum - významná divadla a koncertní sály

RESTAURACE, DOPRAVA A UBYTOVÁNÍ

Restaurace: U Fleků, Café Louvre, Etska Doprava: Hlavní nádraží, Letiště Václava Havla Ubytování: Four Seasons, levné hostely pro turisty

VÝLETY V OKOLÍ

Karlštejn - gotický hrad Karla IV. Kutná Hora - chrám sv. Barbory a Košnice Český ráj - přírodní rezervace s písčkovcovými skalami

ZPÁTKY

Obrázek 35 - cestovní balíček pro Prahu

5.6. Uživatelé

I když jsem mezi své primární funkce aplikace funkci uživatele neřadila, tak jsou ale rozhodně součástí klíčových. Bez existence uživatelů by totiž nemohly vzniknout žádné inzeráty a příspěvky a to jsou jádra mého projektu. Bylo by možné aby tyto funkce byly anonymní, avšak to by zrušilo spoustu dobrých funkcí jako je například úprava vlastních inzerátů nebo ukládání k sobě na profil těch zajímavých. Uživatel může vykonávat následující uživatelské funkce: přihlášení, registrace, odhlášení a zobrazení vlastního profilu.

5.6.1. Reprezentace v databázi

Uživatelský model se do databáze MongoDB registruje jako dokument v souboru *users.model.ts*. Objektům tohoto typu se potom říká *User* objekty. Kromě obecných uživatelských atributů jako je uživatelské jméno, email a heslo má *User* atributy týkající se uživatele osobně a to jsou křestní jméno, příjmení, datum narození a věk. Potom jsou zde tři nepovinné atributy kde všechny představují pole s IDs uživatelových inzerátů, uložených inzerátů a uživatelových příspěvků (*Obr. 36*). Jelikož se jedná o databázi MongoDB, tak má každý dokument databázi atribut *_id* podle kterého je mezi sebou identifikujeme.

```

// mongoose schema for users
const userSchema = new Schema<IUser>({
  username: { type: String, required: true, unique: true },
  first_name: { type: String, required: true },
  last_name: { type: String, required: true },
  birthday: { type: Date, required: true },
  age: { type: Number, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  ads: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Ad' }],
  saved_ads: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Saved_ad' }],
  posts: [{ type: mongoose.Schema.Types.ObjectId, ref: 'posts' }]
})

// registers a User model in database
const User = mongoose.model<IUser>('User', userSchema)

```

Obrázek 36 - definice mongoose schéma a registrace uživatele do databáze

5.6.2. API

Abych mohla komunikovat s databází ohledně uživatelů je zapotřebí mít pro tuto komunikaci zaregistrované endpointy na serveru. Ty si pro uživatele definuji v souboru *users.route.ts*. Ke každé takto registrované adrese se přiřadí funkce importovaná ze souboru *users.controller.ts* a ta je volána při přijmutí requestu z frontendu na tento endpoint. Tato funkce vykoná nějaké logické operace s databází - získání nebo odeslání dat. V případě uživatelů jsou akce s databází k dispozici následující: registrace, přihlášení, odhlášení a získání uživatelského autentizačního statusu (Obr. 37)

```

// import controllers
import { getUser, registerUser, loginUser, logoutUser } from '../controllers/users.controller'

// api routes user authentication
router.post('/register', registerUser)
router.post('/login', loginUser)
router.post('/logout', logoutUser)
router.get('/', getUser)

```

Obrázek 37 - registrace endpointů pro uživatele

5.6.3. sessions

K tomu aby mohli uživatelé zůstat přihlášení se v *Express* projektech většinou používají sessions a nebo JWT tokeny. V mé práci jsem si vybrala sessions, protože poskytují jednoduchou a bezpečnou možnost, jak uchovávat informace o uživatelském stavu přímo na serveru, místo aby se tyto údaje posílaly mezi klientem a serverem při každé požadavce. Sessions fungují tak, že server vygeneruje unikátní identifikátor (session ID), který je uložen do cookie na straně klienta. Tento identifikátor následně umožňuje serveru při každém dalším požadavku načíst data související s konkrétním uživatelem z databáze, aniž by uživatel musel znova procházet autentifikací. [7]

V hlavním souboru backendu *app.ts* se sessions konfigurují a definují se určité atributy (*Obr. 38*). Jelikož je můj projekt psaný v TypeScriptu bylo nutné si specifikovat, že každá proměnná typu *SessionData* (uložené v cookies klienta, které jsou vyčteny při requestu) obsahuje atribut *userId* (*Obr. 39*). To se dělá ve složce *@types*, která byla zmíněna v kapitole 4.1..

```
// session management
app.use(
  session({
    secret: process.env.SESSION_SECRET as string,
    resave: false,
    saveUninitialized: false,
    cookie: {
      maxAge: 60 * 60 * 1000 // lasts an hour
    },
    rolling: true,
    store: Mongostore.create({
      mongoUrl: process.env.MONGO_URI
    })
  })
)
```

Obrázek 38 - konfigurace sessions v *app.ts*

```
// in the package express-session I'm expanding interface SessionData by an attribute userId
declare module 'express-session' {
  interface SessionData {
    userId: mongoose.Types.ObjectId
  }
}
```

Obrázek 39 - rozšíření interface *SessionData* o atribut *userId* v *@types*

```
// get the userId saved in user's session  
const authenticatedUserId = req.session.userId
```

Obrázek 40 - získání uživatelského ID z cookies requestu v *users.controller.ts*

5.6.4. Uživatelské rozhraní

Stejně jako u komunitního fóra, tak ani tady jsem se nesnažila vzhledem uživatelských stránek nějak oslnit (to spíše třeba u inzerátů). Chtěla jsem aby ladily ke staromódní a autentické tématice, ale určitě jsem neměla zapotřebí rozsáhlnejší animace.

Přístupnost stránek pro přihlášení (*Obr. 41*) a registraci (*Obr. 42*) je omezena (více o chráněných adresách v kapitole 4.2.4.) a tak když se na ně snažím dostat jako přihlášený, přesměruje mě to na stránku s mým profilem (*Obr. 43*). Naopak když jako nepřihlášený přepíšu manuálně adresu webu na to aby se mi zobrazil můj účet, tak mě to přesměruje na přihlášení.



Obrázek 41 - přihlášení uživatele

Registrace

Křestní jméno

Příjmení

Tvoje narozeniny

dd.mm.rrrr

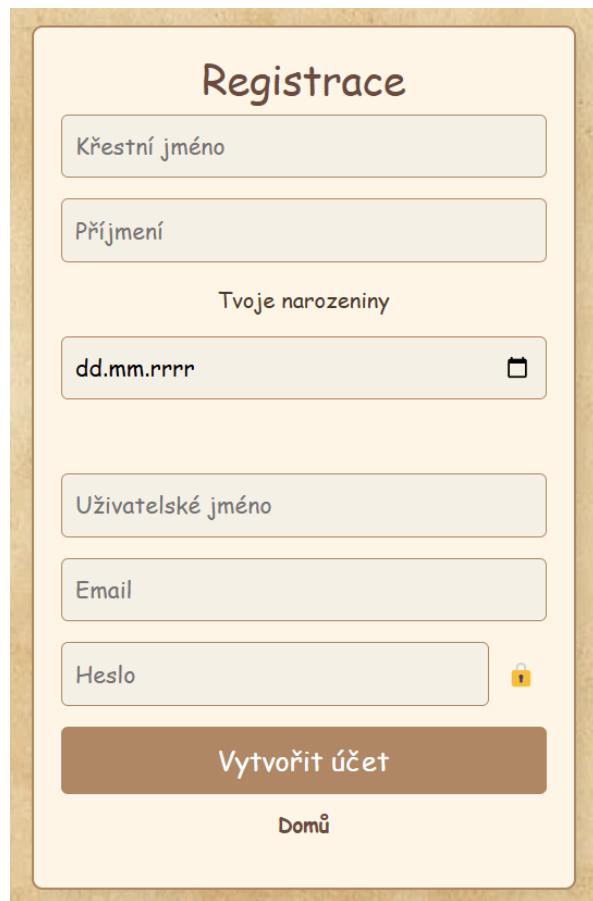
Uživatelské jméno

Email

Heslo

Vytvořit účet

Domů

A registration form titled "Registrace". It contains fields for first name, last name, date of birth (dd.mm.rrrr), username, email, and password. A "Vytvořit účet" (Create account) button is at the bottom, and a "Domů" (Home) link is at the bottom right.

Obrázek 42 - registrace uživatele

Sabina Javůrková

Username: sabi

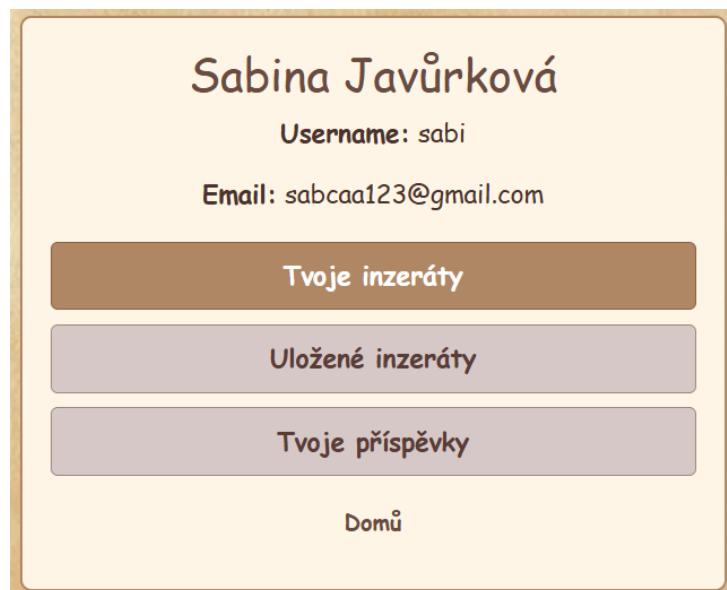
Email: sabcaa123@gmail.com

Tvoje inzeráty

Uložené inzeráty

Tvoje příspěvky

Domů

A user profile interface for "Sabina Javůrková". It shows her username (sabi), email (sabcaa123@gmail.com), and three main navigation buttons: "Tvoje inzeráty" (Your ads), "Uložené inzeráty" (Saved ads), and "Tvoje příspěvky" (Your posts). A "Domů" (Home) link is at the bottom right.

Obrázek 43 - účet uživatele

6. Závěr

Cílem mé ročníkové a maturitní práce bylo naprogramovat webovou aplikaci, která uživatelům ulehčí objevování a cestování po České republice. Klíčovou funkcí byly uživatelské inzeráty, které představují spojku mezi lidmi, kteří by rádi cestovali, ale nemají s kým. Další užitečnou funkcí mělo být komunitní fórum na kterém si mezi sebou uživatelé sdílí osobní postřehy z různých měst a zprostředkovávají tím aktuální informace ostatním. Kromě těchto dvou větších funkcí je zde i interaktivní mapa České republiky a cestovní balíčky s turistickými informacemi.

K realizaci práce jsem využila technologie z MERN kolekce, která se skládá z databáze MongoDB, backend frameworku Express, knihovny React a runtime prostředí Node. Jako programovací jazyk jsem si nevybrala JavaScript ve kterém je většina MERN Stack aplikací, ale TypeScript kvůli jeho bezpečnosti a spolehlivosti.

Snažila jsem se splnit původní zadání a dovoluji si říct, že se mi to podařilo velmi úspěšně. Místy jsem šla i nad rámec - například možnost ukládání inzerátů do uživatelského profilu. Takže s prací jsem nad mírou spokojena.

Do budoucna bych chtěla rozšířit aplikaci tak, že by bylo možné v ní měnit jazyky, abych vylepšila její inkluzivitu a dosáhla na vícero uživatelů. Líbilo by se mi přidat e-mailové ověření při registraci a nebo možnost přihlášení přes Google účet. A v neposlední řadě plánuji aplikaci deploynout na internet aby mohla ČundrAppka opravdu lidem pomoci vytvářet zážitky.

Seznam internetových zdrojů

Obrázek 1 - grafická reprezentace MERN kolekce:

<https://www.mongodb.com/resources/languages/mern-stack>

[1] GeeksforGeeks. MERN Stack. [GeeksforGeeks](#), navštíveno 29. března 2025.

[2] Hygraph. TypeScript vs JavaScript. [Hygraph](#), navštíveno 29. března 2025.

[3] Medium. Beginner Node.js, Express.js, and MongoDB Installation Tutorial for Windows. [Medium](#), navštíveno 29. března 2025.

[4] Google Developers. Get API Key. [Google Developers](#), navštíveno 29. března 2025.

[5] Bacancy Technology. React: index.html and index.js in a Create React App Application. [Bacancy Technology](#), navštíveno 30. března 2025.

[6] GeeksforGeeks. ReactJS Hooks. [GeeksforGeeks](#), navštíveno 30. března 2025.

[7] Stytch. JWTs vs Sessions: Which is Right for You?. [Stytch](#), navštíveno 30. března 2025.

Seznam obrázků

Všechny následující obrázky jsou autorské a jedná se o ukázky kódu:

Obrázek 2 - ukázka .env na frontendu

Obrázek 3 - ukázka .env na backendu

Obrázek 4 - registrace skupin endpointu v app.ts

Obrázek 5 - registrace konkrétních endpointů ve složce routes

Obrázek 6 - controller funkce pro získání specifického inzerátu z databáze

Obrázek 7 - controller funkce pro vytváření příspěvků

Obrázek 8 - mongoose schéma pro cestovní inzerát ve složce models

Obrázek 9 - registrace webových adres v souboru App.tsx

Obrázek 10 - logická úprava dat získaných z backendu v MyAdsPage.tsx

Obrázek 11 - část funkce return v MyAdsPage.tsx

Obrázek 12 - část funkce return v ForumPostingPage.tsx

Obrázek 13 - čtení textového souboru v ViewPackagePage.tsx

Obrázek 14 -ukázka z funkce na formátování textu na sekce v ViewPackagePage.tsx

Obrázek 15 - chování programu po odeslání žádosti o registraci v RegisterUserPage.tsx

Obrázek 16 - část kódu pro autorizaci uživatelských dat v PrivateRoute.tsx

Obrázek 17 - hook funkce posílající API request typu GET v useFetchSingleAd.ts

Obrázek 18 - hook funkce posílající API request typu POST v useCreatePost.ts

Obrázek 19 - získání uživatelských dat za pomocí AuthContext.tsx

Obrázek 20 - ukázka kódu z AuthContext.tsx

Obrázek 21 - deklarace interfacu pro inzerát v ad.ts

Obrázek 22 - deklarace interfacu pro uživatele v user.ts

Obrázek 23 - vykreslení nerozkliknutých inzerátů

Obrázek 24 - příklad vyfiltrování inzerátů

Obrázek 25 - rozkliknutý inzerát

Obrázek 26 - tvorba inzerátu

Obrázek 27 - definice mongoose schéma a registrace příspěvku do databáze

Obrázek 28 - registrace endpointů pro komunitní fórum

Obrázek 29 - vstupní bod komunitního fóra

Obrázek 30 - komunitní fórum Prahy

Obrázek 31 - tvorba příspěvku na komunitní fórum

Obrázek 32 - zobrazení mapy na frontendu

Obrázek 33 - využití API klíče pro generování mapy na frontendu

Obrázek 34 - vstupní bod pro cestovní balíčky

Obrázek 35 - cestovní balíček pro Prahu

Obrázek 36 - definice mongoose schéma a registrace uživatele do databáze

Obrázek 37 - registrace endpointů pro uživatele

Obrázek 38 - konfigurace sessions v app.ts

Obrázek 39 - rozšíření interface SessionData o atribut userId v @types

Obrázek 40 - získání uživatelského ID z cookies requestu v users.controller.ts

Obrázek 41 - přihlášení uživatele

Obrázek 42 - registrace uživatele

Obrázek 43 - účet uživatele