**STŘEDNÍ PRŮMYSLOVÁ ŠKOLA BRNO, PURKYŇOVA,
PŘÍSPĚVKOVÁ ORGANIZACE**

INTERAKTIVNÍ MAPA ŠKOLY

DOMINIK HUML
V4C

Profilová část maturitní zkoušky
MATURITNÍ PRÁCE

Brno 2022

Prohlášení

Prohlašuji, že jsem maturitní práci na téma *Interaktivní mapa školy* vypracoval pod vedením vedoucího maturitní práce samostatně za použití v práci uvedených pramenů a literatury.

Beru na vědomí, že zpráva o řešení maturitní práce a základní dokumentace k aplikaci bude uložena v elektronické podobě na intranetu SPŠ Brno, Purkyňova, příspěvková organizace.

Beru na vědomí, že bude má maturitní práce včetně zdrojových kódů uložena v knihovně SPŠ Brno, Purkyňova, příspěvková organizace, dostupná k prezenčnímu nahlédnutí. Škola zajistí, že nebude pro nikoho možné pořizovat kopie jakékoliv části práce.

Beru na vědomí, že SPŠ Brno, Purkyňova, příspěvková organizace, má právo celou moji práci použít k výukovým účelům a po mém souhlasu nevýdělečně moji práci užít ke své vnitřní potřebě.

Beru na vědomí, že pokud je součástí mojí práce jakýkoliv softwarový produkt, považují se za součást práce i zdrojové kódy, které jsou předmětem maturitní práce, případně soubory, ze kterých se práce skládá. Součástí práce není cizí ani vlastní software, který je pouze využíván za přesně definovaných podmínek, a není podstatou maturitní práce.

Dominik Huml

V Brně 22. dubna 2022 Podpis

Poděkování

Tímto bych rád poděkoval své vedoucí maturitní práce RNDr. Lence Hruškové za veškerý čas, trpělivost, ochotu a veškeré úsilí věnované mému dílu.

Anotace

Maturitní práce *Interaktivní mapa školy* se zabývá vytvořením interaktivního plánku pro navigaci po škole. Slouží jak pro studenty, tak i pro rodiče kteří jdou například na třídní schůzky. V aplikaci je možné prohlížet si plánek školy, vybírat si patra, zobrazit si informace o jednotlivých učebnách a učitelích v nich. Dále je také možné vyhledávat učebny a učitele. Pro administrátora je v aplikaci také přístup k editaci informací o učitelích a místnostech.

Klíčová slova

API, Python, TypeScript, Django, React, SVG

Vedoucí práce: *RNDr. Lenka Hrušková*

Obsah

1 Použité zkratky

HTML - HyperText Markup Language
CSS - Cascading Style Sheets
JSON - JavaScript Object Notation
API - Application Programming Interface
HTTP - Hypertext Transfer Protocol
URL - Uniform Resource Locator
ORM - Object-relational mapping
XML - Extensible Markup Language
SVG - Scalable Vector Graphics
E2E - End To End

2 Teoretický úvod

2.1 Úvod

Naše střední škola je poměrně rozsáhlá, a pro nové studenty nebo pro jejich rodiče může být problém se v ní orientovat. Tato aplikace má sloužit jako pomůcka pro nalezení kabinetu učitele nebo třídy kde mají zrovna výuku.

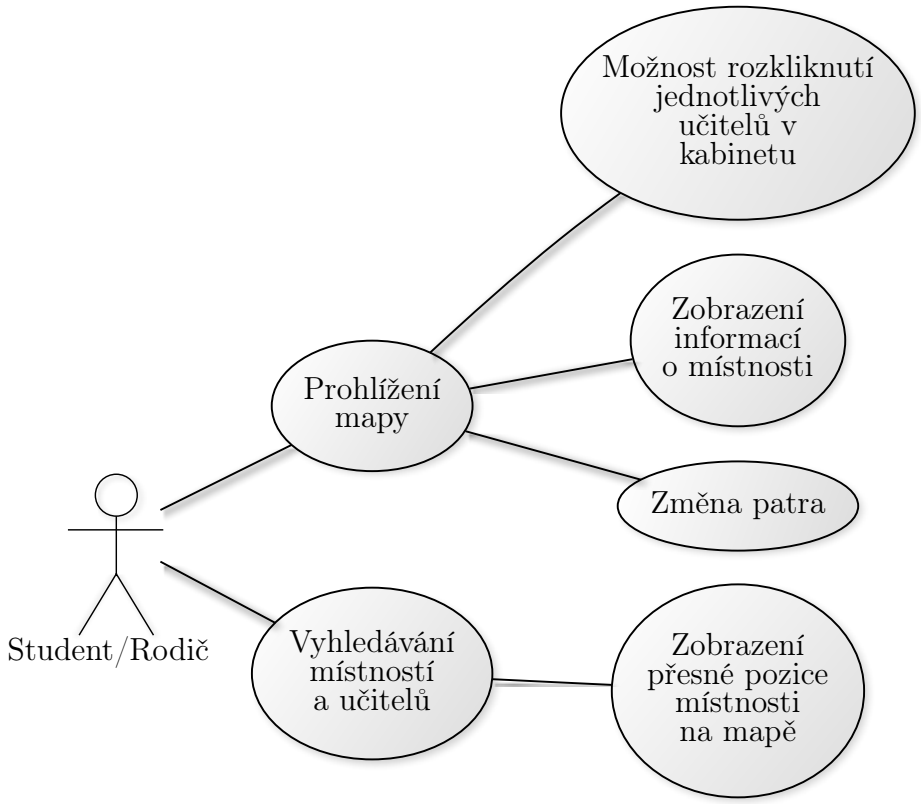
2.2 Cíl práce

Cíle, které jsem si stanovil:

- uživatelské rozhraní práce bude jednoduché a přívětivé
- mapa bude snadno editovatelná pro budoucí použití
- mapa bude přehledná, bude možno rozlišit učebny od kabinetu apod.
- pro administrátora bude dostupný panel pro jednoduchou editaci dat
- při interakci budou zobrazeny relevantní informace

2.3 Diagramy případů užití

Diagram užití zachycuje možné uživatele a všechny akce, které může daný uživatel provádět. Tento diagram zobrazuje akce, které normální uživatel může udělat.



Obrázek 1: UML diagram případů použití

3 Rozbor řešení

3.1 Backend

3.1.1 Použité technologie

Na serverovou část aplikace používám jazyk Python a webový framework Django. Backend slouží jako REST API, která je prostředník mezi databází a klientem.

3.1.2 REST API

REST API je specifický, velmi oblíbený druh API. Vyniká zejména svou jednoduchostí a čitelností. API zajišťuje funkci „rozhraní“ pro předávání dat mezi dvěma či více aplikacemi. REST k tomuto obecnému mechanizmu dodává sadu doporučení a omezení, které když API dodržuje, tak se může nazývat REST API. Příkladem může být „bezstavovost“, při které by klientská aplikace měla vždy posílat veškeré nezbytné informace pro zpracování požadavku. REST API obsahuje dílčí endpointy, které jsou dostupné na definovaných URL adresách, které provolává klientská aplikace přes HTTP protokol. Pro tělo požadavku se nejčastěji používá komunikační formát JSON [1].

3.1.3 Python

Python je vysokoúrovňový interpretovaný skriptovací programovací jazyk. Python podporuje objektově orientované programování, imperativní, procedurální a v omezené míře i funkcionální programování [2].

S Pythonem mám hodně zkušeností jak ze školy, tak i z mých osobních projektů, a tudíž byl jasnou volbou pro moji maturitní práci.

3.1.4 Django

Django je webový framework napsaný v Pythonu, který se zaměřuje na zrychlení vývoje aplikací, a čistý a praktický design. Stará se o většinu věcí, které souvisejí s vývojem webových stránek, takže se lze zaměřit na psaní aplikace bez potřeby znovu objevovat kolo. Je zdarma a open source [3].

Django jsem si vybral jelikož jsem s tímto webovým frameworkem pracoval již v minulosti, takže ho znám celkem dobře. Django má také hodně nástrojů již zabudovaných v sobě, takže je programátor nemusí implementovat. Nástroje které jsem ve maturitní práci využil jsou např. podpora databáze a ORM, serializace, administrační panel, nastavení jednotlivých URL adres, migrace databáze.

3.2 Databáze

Jako databázi jsem si vybral objektově-relační databázový systém PostgreSQL [4], který je zdarma a open source. Pro práci s databází jsem využíval program DBeaver [5].

3.3 Frontend

3.3.1 TypeScript

TypeScript je silně typovaný programovací jazyk. Jedná se o nádstavbu JavaScriptu. Vybral jsem si ho právě kvůli tomu, že je silně typovaný. Díky tomu se hodí i na větší projekty, kde se kontrola typů hodí. Syntax je téměř totožná s JavaScriptem. Výsledný projekt se při spuštění ztranspiluje do JavaScriptu [6].

3.3.2 React

React je open source knihovna pro tvorbu uživatelských rozhraní. Hlavní výhodou oproti čistému JavaScritpu jsou komponenty. Komponent je v funkce nebo třída, která v sobě může mít logiku, a vrací HTML kód. Komponentům můžeme předávat parametry které se nazývají *props*. Komponenty můžeme používat na různých místech v kódu, je to podobné jako kdybychom volali funkci [7].

3.4 Mapa

3.4.1 SVG

Je to značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML. Formát SVG je základním otevřeným formátem pro vektorovou grafiku na webových stránkách. Výhoda vektorové grafiky je, že na rozdíl od rastrové nezáleží na rozlišení a zachovává si pořád stejnou kvalitu. Kvůli tomu byla jasnou volbou pro moji mapu. Každé patro mám jako samostatný SVG soubor a načítám je dle potřeby [8].

3.4.2 Inkscape

K vytvoření map jsem použil program Inkscape [9]. Je to software na vytváření vektorové grafiky. Výhodou je, že je zdarma a open source. Zvažoval jsem také program Adobe Illustrator, ale v něm není možné editovat ID jednotlivých elementů v SVG souboru.

3.5 Lokální vývoj aplikace

Pro frontend jsou použil nástroj npm, přes který lze zapnout lokální frontend server a který se stará o restartování serveru při změně kódu. Pro backend jsem použil nástroj od Djanga, který také spustí lokální server. Tento způsob se hodí pouze na vývoj, není určen do produkčního prostředí.

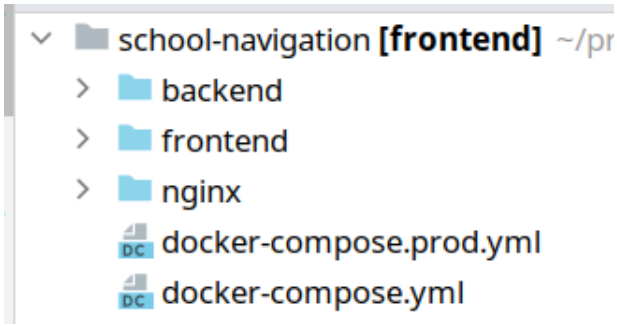
3.6 Nasazení aplikace do produkčního prostředí

Aplikace jsem nasazoval do produkčního prostředí na mém serveru s operačním systémem Ubuntu. Hostuji zde jak svoji databázi, tak webový server. K bezpečnému běhu aplikace používám webový server a zároveň reverzní proxy Nginx. Dále jsem využil technologii Docker, díky které může být má aplikace spuštěna na libovolném serveru během několika minut.

4 Experimentální část

4.1 Struktura projektu

Projekt obsahuje složky `backend`, `frontend` a `nginx`. Ve složce `nginx` se nachází konfigurace pro webový server Nginx. Soubory `docker-compose.yml` a `docker-compose.prod.yml` slouží k spojení více Dockerfilu, a následnému spuštění celé aplikace.



Obrázek 2: Struktura projektu

4.1.1 Backend

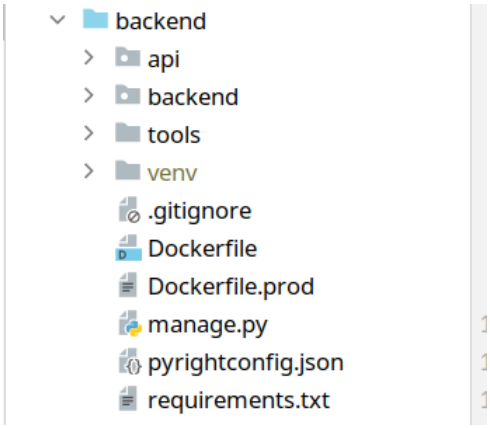
Backend zachovává strukturu kterou vytvoří Django. Popis složek:

- `backend` - nachází se zde soubory důležité pro celý projekt, např. nastavení
- `api` - obsahuje logiku backendu, jsou zde definovány views, modely, serializéry, admin stránka a endpointy
- `tools` - zde se nachází mé vlastní scripty, pro hromadné vkládání dat do databáze, a pro získávání dat ze stránek školy
- `venv` - virtuální prostředí pro python, instalují se zde dodatečné knihovny

Popis souborů:

- `Dockerfile` - soubor který obsahuje příkazy k vytvoření Docker image, je zde verze pro vývoj a pro produkci
- `requirements.txt` - seznam knihoven, které je třeba nainstalovat přes pip

- `manage.py` - nástroj od Djanga, můžeme přes něj např. spouštět lokální server nebo vytvářet administrátorské účty



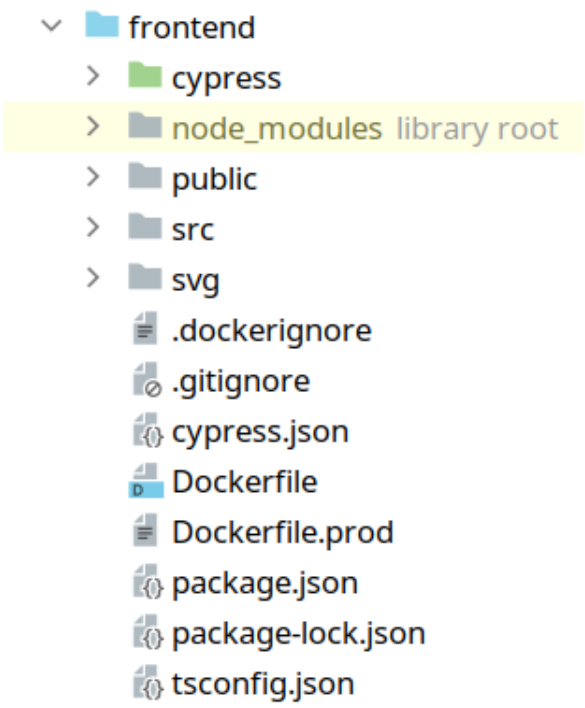
Obrázek 3: Struktura backendu

4.1.2 Frontend

Stejně jak v backendu, tak i ve frontendu jsem zanechal stejnou strukturu jako doporučuje React. Popis složek:

- `cypress` - nachází se zde soubory pro testování
- `node_modules` - zde jsou všechny knihovny
- `public` - obsahuje statické soubory, např. index.html a ikonu stránky
- `svg` - zde jsou uloženy všechnya patra mé mapy
- `src` - složka obsahující logiku frontendu:
  - `components` - zde jsou funkční komponenty, které renderují obsah
  - `reducers` - zde se ukládá stav aplikace
  - `services` - modul který komunikuje s API backendu

Soubor `tsconfig.json` obsahuje konfiguraci pro transpilaci TypeScriptu na JavaScript. V souborech `package.json` a `package-lock.json` se nachází seznam knihoven, které mají být nainstalovány.



Obrázek 4: Struktura frontendu

4.2 Mapa

Vytvořit pěknou a funkční mapu byl pro mě asi nejtěžší úkol. Musel jsem vymyslet způsob, jak efektivně zobrazit všechny učebny a jak zjistit, s kterou uživatel zrovna interagoval. Další požadavek byl lehká editace mapy, v případě změny místností ve škole. Při výběru technologie na vytvoření mapy jsem zvažoval tři možné kandidáty: HTML, Canvas a SVG [10].

4.3 Technologie na vytvoření mapy

4.3.1 HTML

HTML je značkovací jazyk, používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. HTML je hlavním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na internetu. Základní stavební bloky HTML jsou HTML elementy. Element jsou vytvářeny pomocí špičatých závorek a názvu elementu. Vzhled HTML můžeme změnit pomocí CSS. To se používá tak, že se vytvoříme soubor .css, pomocí CSS selektorů přidáme jednotlivým skupinám prvků vzhled, který chceme a soubor poté propojíme v HTML v hlavičce souboru. Ta se označuje elementem

`head` a jsou v ní potřebná metadata a konfigurace pro stránku. Informace z hlavičky se na výsledné stránce nezobrazují, vidíme pak jen to, co je definováno v elementu body [11].

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico"/>
    <title>React App</title>
</head>
<body>
<div id="root">
    <h1>My Headline!</h1>
</div>
</body>
</html>
```

Obrázek 5: Ukázka HTML kódu

Použití HTML elementů jako je např. Div pro zobrazení mapy má tu výhodu, že vytvářet elementy je jednoduché. Další výhoda je, že pomocí JavaScriptu můžeme reagovat na události, které uživatel provede na jednotlivých elementech. Problém nastává, když potřebujeme jiné tvary než čtverce a kruhy. Další problém může být rychlost vykreslování. Větší počet objektů a složitější stylování značně zpomaluje stránku. Kvůli tomuto se HTML hodí pouze na jednodušší struktury, s malým počtem prvků, což se nehodí pro moji aplikaci.

4.3.2 HTML Canvas

Element Canvas je nízkoúrovňové API rozhraní pro vykreslování rastrové grafiky v prohlížeči pomocí JavaScriptu [12]. Většinou se používá vytváření her a animací. Hodí se pro vývoj aplikací, kde je spousta menších objektů, s kterými potřebujeme často manipulovat. Jelikož to je ale rastrová grafika, tak její kvalita záleží na rozlišení. Při častém oddalování a přibližování by byla potřeba neustále vykreslovat objekty znova. Canvas také trpí špatným renderováním textu, kvůli zmíněné rastrové grafice. Jelikož je Canvas nízkoúrovňový, tak zde nemáme přístup k eventům na objektech, tak jak jsme zvyklí z HTML. Po

kliknutí zde dostaneme pouze souřadnice, kam uživatel klikl, tudíž si vývojář musí napsat spoustu kódu navíc, aby mohl s událostmi pracovat tak, jak potřebuje. Kvůli této přidané komplexitě, která se spíše hodí na vývoj her, jsem si ani canvas nevybral.

4.3.3 SVG

SVG je, podobně jak HTML, značkovací jazyk, a formát soboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML. Formát SVG je základním otevřeným formátem pro vektorovou grafiku na webových stránkách. Vektorová grafika, na rozdíl od rastrové, má tu výhodu, že nezáleží na rozlišení. Pokaždé má stejnou kvalitu. Je to díky tomu, že celá obraz se skládá z jednoduchých objektů, jako jsou přímky, obdélníky, elipsy apod. SVG obrázek pak můžeme jednoduše vložit do prohlížeče, přes svg element. Dále také můžeme aplikovat CSS stylování jak na normálních HTML elementech, a reagovat na událostí vytvořené uživatelem.

Díky těmto vlastnostem je SVG ideální pro mapy, a proto jsem si jej zvolil. Mapu jsem vytvořil v programu Adobe Illustrator. Jednotlivé učebny od sebe rozlišuji pomocí HTML tagu ID, což je unikátní identifikátor pro každou místnost.

```
<g id="204">
    <g id="g1466">
        <rect
                id="rect1656"
                x="1172.7"
                y="1704.5"
                transform="matrix(0.9495 -0.3137 0.
                class="st5"
                width="109.2"
                height="165"/>
        <g
                id="g1982">
<g
        id="g1980">
    <path
            id="path1974"
            d="M1169.8,1733.5c0.5-0.2,1-0.2,1.4-0.1
    <path
            id="path1976"
```

Obrázek 6: Ukázka SVG kódu

4.3.4 Pohyb na mapě

Pohybování, přibližování a oddalování jsem vyřešil pomocí knihovny **d3.js** [13]. Knihovna má v sobě tyto metody zabudované, takže jsem je akorát propojil s SVG.

```
const coords = (element as SVGSVGElement).getBBox();

// @ts-ignore
const handleZoom = (e) => {
    d3.select( selector: '#svg-map g')
        .attr( name: 'transform', e.transform);
}

const zoom = d3.zoom()
    .scaleExtent( extent: [0.25, 3])
    .on( typenames: 'zoom', handleZoom);

// @ts-ignore
d3.select( selector: "#svg-map").transition().duration( milliseconds: 500).call(zoom.translateTo, coords.x, coords.y);
```

Obrázek 7: Automatické přiblížení místnosti při kliknutí

4.4 Frontend

4.4.1 Komunikace s API

Frontend komunikuje s backendem pomocí knihovny **axios** [14]. V modulu `api.ts` jsou nadefinovány funkce, které pošlou požadavek na backend, přijmou data v JSONu, a ten se automaticky převede na TypeScript objekt.

```
const makeQuery = async (query: string): Promise<any> => {
    const res = await axiosInstance.get( url: `${BASE_URL}search/?query=${query}`);
    return res.data;
};

const getRoom = (id: string): Promise<any> => {
    return axiosInstance.get( url: `${BASE_URL}room/${id}`).then((res : AxiosResponse<any> ) => res.data);
};
```

Obrázek 8: Funkce pro komunikaci s API

4.4.2 Komponenty

Hlavním stavebním kamenem v Reactu jsou komponenty. Komponent je funkce, která může přijímat data přes argumenty - nazývají se *props*, a vrací HTML.

Uvnitř komponentu můžeme napsat logiku pro komponent, např. reagování na eventy nebo načítání dat z backendu. HTML se v komponentu zapisuje pomocí JSX, což je HTML rozšíření pro JavaScript, a díky kterému lze psát HTML pohodlněji. V JSX můžeme v složených závorkách spouštět JavaScript kód nebo vkládat další komponenty. Komponent se v JSX zapisuje stejně jako HTML element. Argumenty do komponentu vkládáme stejně jako do HTML elementů vkládáme atributy.

```
const Search = () => {
    const results = useAppSelector( selector: (state : RootState ) => state.result.value);
    const dispatch = useAppDispatch();
    const selectedRoomID = useAppSelector( selector: (state : RootState ) => state.selected.value.room);

    useEffect( effect: () => {
        dispatch(clearSelected());
        dispatch(clearTypes());
    }, deps: []);

    const handleSearch = (query: string) => {
        makeQuery(query).then((data) => {
            dispatch(setSelectedType(data.type));
            const firstResult = data.result[0];
            if (selectedRoomID !== firstResult.room_id) {
                dispatch(setSelectedRoom( state: firstResult.room_id + ""));
            }
        });
        dispatch(clearResult);
    };

    return (
        <div className="search">
            <div className={"search-fill"}></div>
            <SearchBox/>
            {results.length !== 0 ? (
                <Results handleSearch={handleSearch}/>
            ) : (
                ""
            )}
        </div>
    );
};

export default Search;
```

Obrázek 9: Příklad komponentu

4.4.3 Ukládání stavu aplikace

V Reactu používám dva způsoby pro ukládání stavu. Ten první je použití hooku *useState*. State Hook nám dovoluje ukládat stav. Tento stav je ale viditelný jen pro daný komponent, a hodí se ho používat jen pro lokální věci. Definujeme ho voláním funkce *useState()*, do které vložíme výchozí stav. Funkce nám vrátí dvě proměnné, jednu pro přístup k datům, a druhou k změně dat.

V aplikaci ho využívám například pří získávání a následnému zobrazení dat z databáze. Při načtení stránky se stav `fetched` nastaví na *false*, a pošle se požadavek na backend. Mezitím, co je stav *false*, tak se uživateli zobrazí ikona načítání. Po získání dat se stav změní na *true* a data budou zobrazena uživateli.

```
const [fetched, setFetched] = useState( initialState: true);

return (
    <>
        {!fetched ? <div>page is loading </div> : <div>your data</div>}
    </>
);
```

Obrázek 10: Využití hooku *useState*

V aplikaci jej využívám například pří získávání a následnému zobrazení dat z databáze. Při načtení stránky se stav `fetched` nastaví na *false*, a pošle se požadavek na backend. Mezitím co je stav *false*, tak se uživateli zobrazí ikona načítání. Po získání dat se stav změní na *true*, a data budou zobrazena uživateli.

Jako druhý způsob ukládání dat jsem použil Redux Toolkit, což je sada nástrojů, která vylepšuje knihovnu Redux, a ta slouží jako manažer globálního stavu. Zde ukládám věci, které jsou potřeba v celé aplikaci, např. výsledky vyhledávání, vybraná místnost nebo patro, na kterém se uživatel zrovna nachází. Pro každý typ dat které chceme ukládat si vytvoříme *slice*, což je funkce, která nám vytvoří akce a reducer pro data. Akce jsou funkce, které nám říkají co máme dělat se stavem, a reducer tyto funkce provádí. Slice soubory se nachází ve složce `reducers` a mají jméno ve formátu `[TypDat]slice.ts` [15].

Po vytvoření slice souborů musíme vytvořit store. Ten obsahuje všechny naše reducery a přes něj měníme a získáváme stav. Pro změnu stavu použijeme hook *useDispatch*, který nám vrátí funkci *dispatch*. Pomocí této funkce poté posíláme jednotlivé akce do storu. Přístup k datům získáme přes hook *useSelector*, do kterého jako parametr vložíme funkci, která určuje k jakým datům budeme přistupovat. Hook nám vrátí proměnnou, ve které je uložen stav [16].

```
type arrayAction = PayloadAction<Array<EmployeeWithRooms | Room>>;
interface ResultState {
    value: Array<EmployeeWithRooms | Room>
}

export const resultSlice = createSlice( options: {
    name: "result",
    initialState: {
        value: [],
    } as ResultState,
    reducers: {
        setResult: (state : Draft<State> , action: arrayAction) => {
            state.value = action.payload;
        },
        clearResult: (state : Draft<State> ) => {
            state.value = [];
        },
    },
});

export const { setResult, clearResult } = resultSlice.actions;

export default resultSlice.reducer;
```

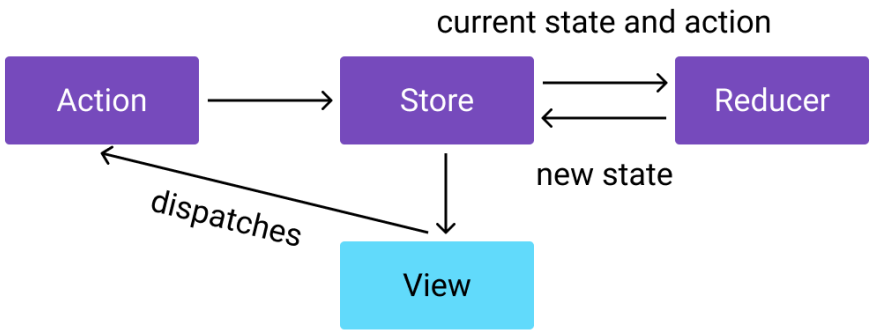
Obrázek 11: Vytvoření slice souboru

```
const store = configureStore( options: {
    reducer: {
        result: resultReducer,
        selected: selectedReducer,
        types: typeReducer,
        floor: floorReducer,
    },
});

export default store;

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Obrázek 12: Vytvoření storu



Obrázek 13: Diagram pro Redux Toolkit

4.5 Návrh databáze

4.5.1 Room

Model room obsahuje dva identifikátory. ID místnosti, což je řetězec, který má každá místnost ve škole, a pokud je místnost zároveň i učebna, tak má ještě jedno číslo. U ostatních místností má tento sloupec hodnotu `null`. Sloupec je klapka kabinetu – poslední 3 čísla kterými končí telefonní číslo do kabinetu. Předchozích 6 znaků má každý kabinet stejné, tudíž nejsou uložené pro každý kabinet zvlášť.

4.5.2 Employee

Model obsahuje standardní údaje jako jsou, jméno, pohlaví a email. Nachází se zde také sloupec *phone_number*, který obsahuje pracovní číslo daného zaměstnance. Většina zaměstnanců toho číslo nemá, a mají pouze pevnou linku v kabinetu. Poslední 3 čísla, tzv. klapka, jsou obsáhnuty v modelu Room.

4.5.3 EmployeeRoom

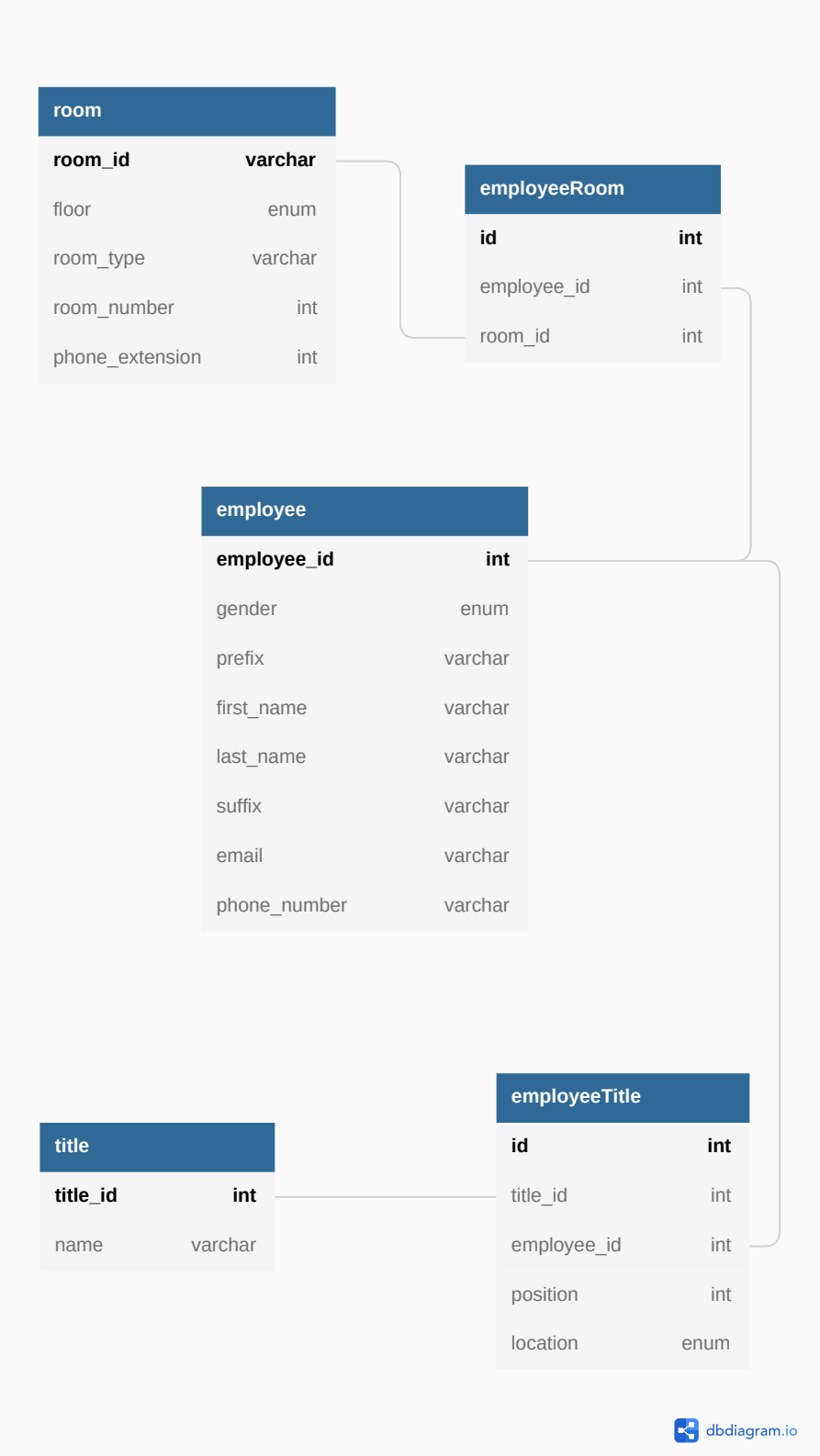
Spojovací tabulka pro spojení kabinetu a zaměstnance. Jelikož jeden zaměstnanec může mít více kabinetů, a v jednom kabinetu může být více zaměstnanců, tak je to M:N vazba.

4.5.4 Title

Tabulka obsahující akademické tituly.

4.5.5 EmployeeTitle

Spojovací tabulka pro akademické tituly jednotlivých zaměstnanců. Každý záznam má sloupec location a position. Location nám určuje, jestli je titul před jménem (hodnota before) nebo za jménem (hodnota after). Sloupec position je pořadí, v kolikátém se titul píše ve jméně. Takže například pro jméno `Mgr. Petr Novák, Ph.D.` bude mít `Mgr.` hodnoty *1* a *before*, a `Ph.D.` hodnoty *2* a *after*.



Obrázek 14: Schéma databáze

4.6 Backend

4.6.1 Komunikace s databází

Komunikace s databází je zajištěna pomocí ORM nástroje od Djanga. Díky tomu nemusíme psát SQL dotazy ručně a můžeme s databází pracovat jako s objekty v Pythonu. Model definujeme jako třídu v Pythonu, jednotlivé sloupce a vazby definujeme jako vlastnosti třídy [17].

```
class EmployeeRoom(models.Model):
    employee_room_id = models.AutoField(primary_key=True)
    employee_id = models.ForeignKey("Employee", on_delete=models.CASCADE)
    room_id = models.ForeignKey("Room", on_delete=models.CASCADE)

    class Meta:
        db_table = "employee_room"

    def __str__(self):
        return f"{self.room_id.room_id} - {self.employee_id.display_name()}"
```

Obrázek 15: Definice modelu pomocí ORM

4.6.2 API

V Djangu se mapují jednotlivé koncové body k funkcím v souboru urls.py, který se nachází v modulu api. Definuje se zde URL adresa, možné parametry v ní, jejich typy a poté funkce která se spustí pří dotazu od uživatele. Tyto funkce se nazývají views a jsou umístěny v souboru views.py. View je funkce která v argumentech přijímá request objekt, který obsahuje informace o příchozím dotazu, a poté jednotlivé parametry jako další argumenty. Funkci obalíme dekorátorem, ve kterém definujeme, jaké HTTP metody povolujeme pro daný koncový bod [18]. Funkčnost API jsem testoval pomocí nástroje Postman [19].

```python
@api_view(["GET"])
def all_rooms_by_floor(request, floor: int):
    rooms = Room.objects.filter(floor=floor)
    room_serializer = RoomSerializer(rooms, many=True)
    return Response(room_serializer.data)


@api_view(["GET"])
def get_room(request, room_id: str):
    room = Room.objects.get(pk=room_id)
    room_serializer = RoomSerializer(room)
    return Response(room_serializer.data)
```

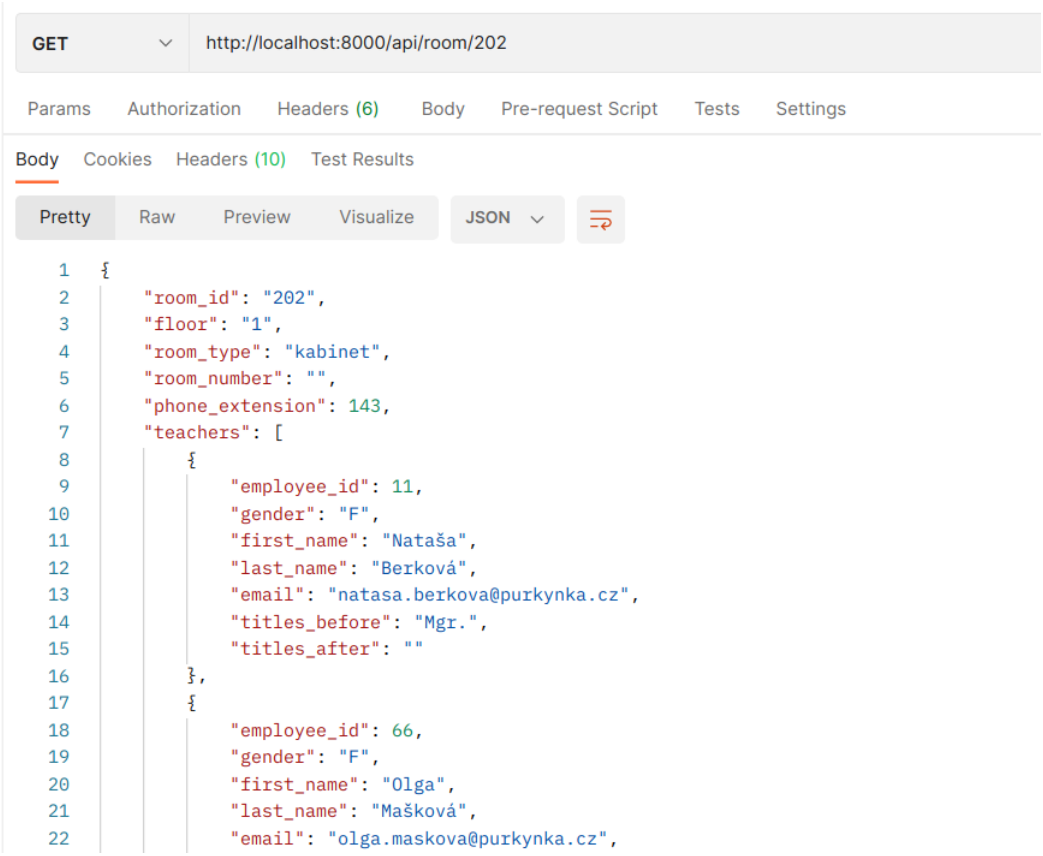
Obrázek 16: Definice views pro API

```python
urlpatterns = [
    path("room/<str:room_id>", get_room),
    path("employee/<str:employee_id>", get_employee),
    path("employee/<str:employee_id>/room/<str:room_id>", get_employee_in_room),
    path("employee/", all_employees),
    path("room/", all_rooms),
    path("search/", search),
    path("room/floor/<int:floor>", all_rooms_by_floor),
]
```

Obrázek 17: Mapování endpointů k příslušným funkcím

4.6.3 Serializace

Data, se kterými pracujeme v backendu, jsou uložené jako objekty v paměti a my je potřebujeme dostat do formátu, který můžeme poslat přes HTTP. Nejpoužívanější takový formát je JSON. K převedení objektu na JSON si nadefinujeme serializér, do kterého poté vložíme objekt a získáme JSON, který, pošleme jako odpověď na frontend. Serializér se definuje podobně jako ORM model. Vytvoříme si třídu, zvolíme si sloupce, které budeme chtít serializovat, a vybereme model, na který se serializér vztahuje [20].



Obrázek 18: Ukázka z testování API pomocí nástroje Postman

```python
class RoomSerializer(serializers.ModelSerializer):
    teachers = EmployeeSerializer(read_only=True, many=True, source="employee_set")

    class Meta:
        model = Room
        fields = ("room_id", "floor", "room_type", "room_number", "phone_extension", "teachers")
```

Obrázek 19: Serializér pro spojovací tabulku

4.6.4 Zabezpečení přihlašovacích údajů

K bezpečné práci s přihlašovacími údaji k databázi jsem zvolil jejich uložení do proměnných prostředí, které jsou definovány v souboru .env. Po spuštění aplikace se načtou do proměnných, odkud backend získá jejich hodnoty.

```
"default": {
    "ENGINE": "django.db.backends.postgresql_psycopg2",
    "NAME": os.getenv("DB"),
    "USER": os.getenv("USER"),
    "PASSWORD": os.getenv("PASSWORD"),
    "HOST": os.getenv("HOST"),
    "PORT": os.getenv("PORT"),
},
```

```
DB="test"
USER="test_user"
PASSWORD="test_pwd"
HOST="1.1.1.1"
PORT=5432
```

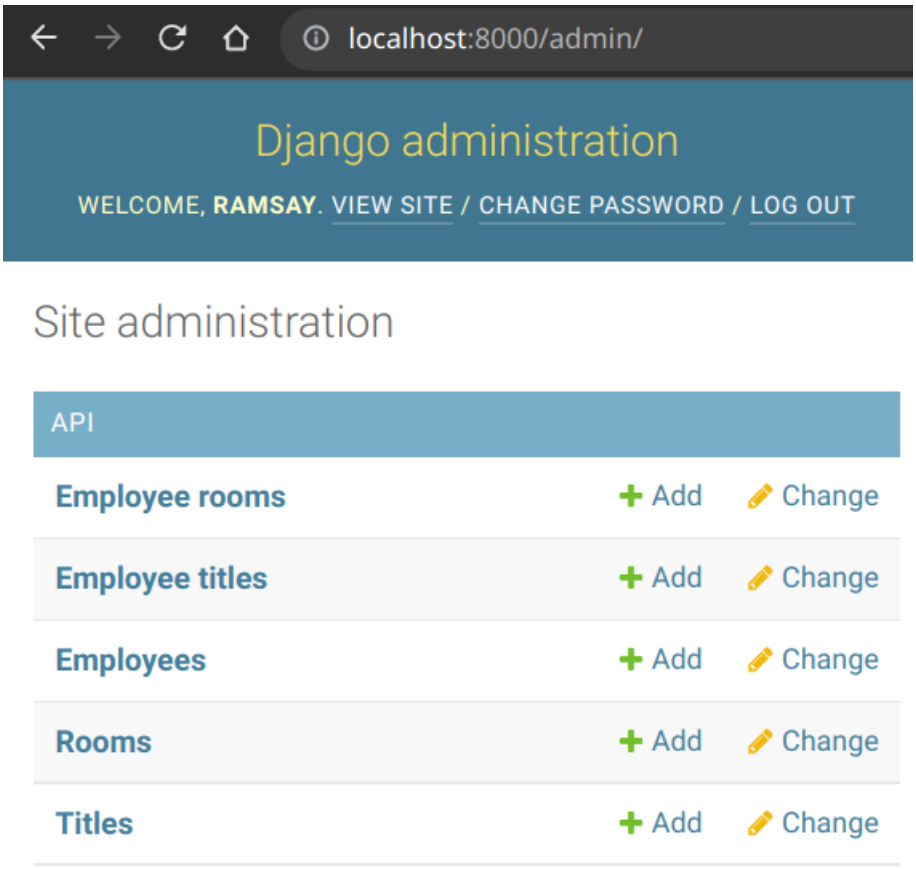
Obrázek 20: Načtení `.env` souboru

4.7 Administrační panel

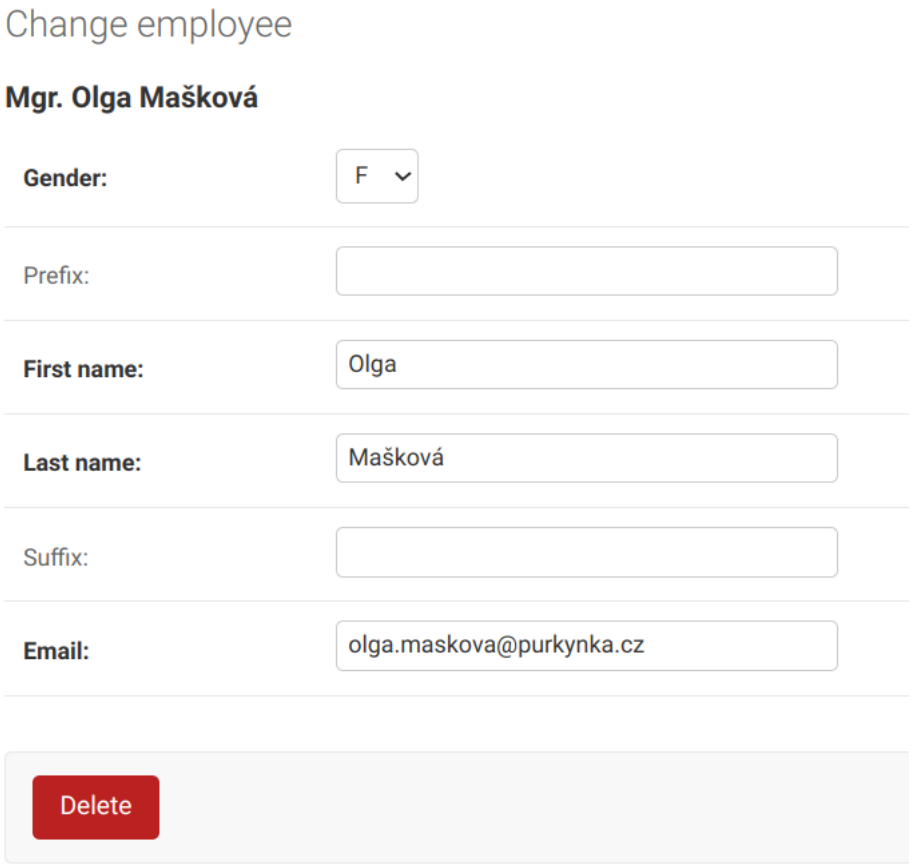
K vytvoření panelu pro administraci jsem použil nástroj od Djanga. V souboru `admin.py` jsem nadefinoval, které modely bude možné editovat v panelu, a Django vše potřebné vygeneruje. Administrační panel se nachází na adrese `/admin`. Lze zde vytvářet, upravovat i mazat záznam v databázi [21].

```
admin.site.register(Room)
admin.site.register(Employee)
admin.site.register(Title)
admin.site.register(EmployeeTitle)
admin.site.register(EmployeeRoom)
```

Obrázek 21: Definice modelů



Obrázek 22: Úvod administračního panelu



Obrázek 23: Editace zaměstnance

4.8 Testování aplikace

Ve své aplikaci jsem na testování použil E2E testy. E2E testy se zaměřují na testování celé aplikace, jak frontendu, tak i backendu. Úkolem těchto testů je interagovat se stránkou stejně, jak by interagoval uživatel. V mé maturitní práci k tomu používám JavaScript knihovnu **Cypress** [22], která si spustí svůj prohlížeč, a následně provádí akce na frontendu. Testy jsou nadefinované v složce `frontend/cypress/integration`. Nepíšou se typickým JavaScriptem, na definici testu se používá jednoduchá syntaxe.

Na obrázku 24 máme test, který kontroluje, jestli se učebna, kterou vyhledáme, správně zobrazí na mapě. V testu klikneme na vyhledávání, napíšeme číslo učebny, vybereme první výsledek, a poté ověříme, jestli má učebna CSS třídu *selected-room*, která nám značí, že třída je vybrána.

```
it( title: "Search room, click on it, show it on map", config: () => {
    cy.get("#search-input").click();
    cy.get("#search-input").type( text: "204");
    cy.get(".results").first().click();
    cy.get(`#204`).children().first().should( chainer: "have.class", value: "selected-room");
});
```

Obrázek 24: Testování zda se výsledek vyhledávání zobrazí na mapě

Na obrázku 25 testuji, jestli se zobrazuje správný počet výsledků při vyhledávání. Přes frontend vyhledáme jméno, a poté pošleme stejný dotaz na backend a zkontrolujeme, jestli mají oba výsledky stejnou délku.

```
it( title: "Number of results is correct", config: () => {
    cy.get("#search-input").click();
    cy.get("#search-input").type( text: "Ivana");
    cy.get(".results").children().its( propertyName: "length").then( fn: size => {
        cy.request("http://localhost:8000/api/search/?query=Ivana").then( fn: data => {
            cy.wrap(data.body.result).should( chainer: "have.length", size);
        });
    });
});
```

Obrázek 25: Testování počtu výsledků

Obrázek číslo 26 zobrazuje test, zda je možné kliknout na všechny místnosti prvního patra. V testu si z API načteme všechny místnosti, a poté na ně klikáme. Zjišťujeme jestli mají po kliknutí CSS třídu *selected-room*.

```
describe( title: "Checking if all rooms are clickable", fn: function () {
    it( title: "First floor", config: () => {
        cy.visit("http://localhost:3000");
        cy.request("http://127.0.0.1:8000/api/room/floor/1").then( fn: (data) => {
            const x = data.body.map(k => k.room_id);
            cy.wrap(x).each( fn: j => {
                const el = cy.get(`#${j}`).children().first();
                el.click( options: { force: true }).should( chainer: "have.class", value: "selected-room");
            });
        });
    });

});
```

Obrázek 26: Testování zda funguje interakce s mapou

Díky testování aplikace si můžu být jistý, že kdykoliv přidám nějaké změny, tak hned vím, zda se něco rozbilo.

4.9 Webový server a reverzní proxy

Používám **Nginx** [23] jako webový server a zároveň reverzní proxy. Webový server slouží k přijímání HTTP požadavků a následně k odesílání odpovědí. Reverzní proxy směřuje HTTP provoz do aplikací na serveru. Tyto aplikace mohou běžet pouze lokálně a šifrovaný provoz stačí nastavit pouze pro reverzní proxy. Konfigurační soubor se nachází v adresáři `nginx`. Na obrázku 27 je konfigurace pro backend. Říkáme zde proxy, aby všechny požadavky které začínají `/api`, `/auth` a `/admin` přesměrovala na lokaci `proxy_api`, která je definována jako adresa backendu, který běží na serveru pouze lokálně.

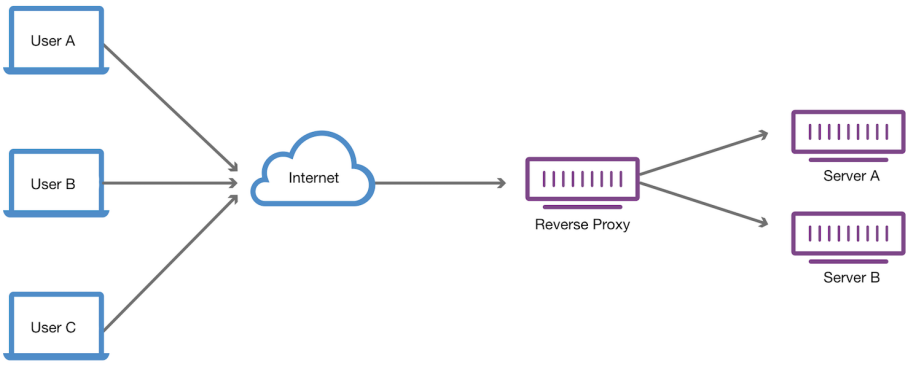
```
location /api {
    try_files $uri @proxy_api;
}

location /auth {
    try_files $uri @proxy_api;
}

location /admin {
    try_files $uri @proxy_api;
}

location @proxy_api {
    proxy_set_header X-Forwarded-Proto http;
    proxy_set_header X-Url-Scheme $scheme;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass   http://backend:8000;
}
```

Obrázek 27: Část konfigurace reverzní proxy



Obrázek 28: Ukázka jak funguje reverzní proxy

4.10 Docker

Docker je open source projekt, jehož cílem je poskytnout jednotné rozhraní pro izolaci aplikací do kontejnerů. Jedná se o „odlehčenou virtualizaci“, kde kontejner obsahuje pouze soubory aplikace a pro ně specifické soubory, ale neobsahuje (virtualizovaný) operační systém. Díky tomu se výrazně snižují režijní nároky na rozdíl od standardních virtuálních strojů.

Výhoda Dockeru je, že můžete spustit svoji aplikaci na každém zařízení, na kterém je Docker nainstalovaný. Stačí pouze vytvořit `Dockerfile`. Dockerfile je textový soubor ve kterém jsou příkazy k vytvoření image. Většinou tyto příkazy obsahují instalaci balíčků a kopírování souborů. Docker image je pak spustitelný soubor s těmito příkazy, každý příkaz je jedna vrstva. Když pak image spustíme, tak se jeho instance nazývá kontejner. V něm běží naše aplikace. Když kontejner vypneme, tak se smaže, stav se neukládá. Právě to je výhoda kontejneru, kdykoliv a kdekoliv můžeme vytvořit čistou instalaci naší aplikace.

Rozbor Dockerfilu na obrázku 29:

- `FROM` - Stáhne se image obsahující Node.js s operačním systémem Alpine a použije se jako podklad
- `WORKDIR` - vytvoří se a nastaví se uvedený adresář jako základní adresář
- `ENV` - knihovny se přidají jako globální spustitelné soubory
- `COPY package.json` - zkopírují se konfigurační soubory z disku do kontejneru
- `RUN npm ci -silent` - nainstalují se potřebné knihovny
- `COPY . ./` - celá aplikace se zkopíruje do kontejneru
- `RUN npm run build` - vytvoří se produkční verze programu
- `COPY ./build /var/www/html/` - vytvořená verze se zkopíruje do složky, odkud bude Nginx stránku zobrazovat

Doporučení je dělat pro každou aplikace jeden Dockerfile. V mém projektu mám jeden Dockerfile pro backend, a jeden pro frontend. Dále mám také verze pro produkci a pro vývoj. Liší se tím, že ve vývojové verzi nepoužívám Nginx jako webový server, a místo toho mám zapnutý jen jednoduchý Node.js server pro frontend, který se automaticky restartuje při změně. V produkční verzi se celý frontend transpiluje do JavaScriptu a je uložený jako statická webová stránka.

Pro propojení více Dockerfilu používám nástroj **Docker Compose**. Ke konfiguraci se používá soubor `docker-compose.yml`. V souboru na obrázku 30 nakonfigurujeme všechny služby. Určíme pro ně adresáře, příkazy které se mají spustit, a namapujeme porty v interní síti kontejnerů, na reálné porty na

```
FROM node:14.17.0-alpine

WORKDIR /app

ENV PATH /app/node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN npm ci --silent

COPY . ./

RUN npm run build

COPY ./build /var/www/html/
```

Obrázek 29: Produkční Dockerfile pro frontend

našem zařízení. Pomocí příkazu `volumes` namapujeme složky s projektem do kontejnerů, a díky tomu je nemusíme kopírovat. Tohoto nastavení využívám pouze při vývoji k urychlení procesu vytváření kontejneru, v produkční verzi se kopíruje celý projekt do kontejneru.

```yaml
version: '3.7'

services:
    backend:
        build:
            context: ./backend
            dockerfile: ./Dockerfile
        command: python manage.py runserver 0.0.0.0:8000
        volumes:
            - ./backend:/usr/src/app
        ports:
            - "8000:8000"
    frontend:
        build: ./frontend
        ports:
            - "3000:3000"
        environment:
            - CHOKIDAR_USEPOLLING=true
        volumes:
            - ./frontend:/usr/src/app
        command: npm run start
```

Obrázek 30: Konfigurační soubor pro Docker Compose - verze pro vývoj

5 Závěr

Zadání maturitní práce jsem splnil ve všech bodech. Výstup je aplikace, která umožňuje rodičům a studentům jednoduchou navigaci po škole. Během práce na tomto projektu jsem si rozšířil mé znalosti ve vývoji webových aplikací, a to hlavně v práci s SVG.

Seznam obrázků

Odkazy

[1] DAMI development s.r.o. „REST API.“ (2022), URL: https://www.damidev.com/slovnik/rest-api?lang=en#:~:text=REST%20API%20je%20specifick%C3%BD%2C%20velmi,se%20m%C5%AF%C5%BEe%20naz%C3%BDvat%20REST%20API. (cit. 21. 04. 2022).

[2] Wikipedia. „Python.“ (2022), URL: https://cs.wikipedia.org/wiki/Python (cit. 21. 04. 2022).

[3] Django Software Foundation. „Django - Web Framework.“ (2022), URL: https://www.djangoproject.com/ (cit. 21. 04. 2022).

[4] PostgreSQL Global Development Group. „PostgreSQL.“ (2022), URL: https://www.postgresql.org/ (cit. 21. 04. 2022).

[5] R. Serge. „DBeaver.“ (2022), URL: https://dbeaver.io/ (cit. 21. 04. 2022).

[6] Microsoft Corporation. „TypeScript.“ (2022), URL: https://www.typescriptlang.org/ (cit. 21. 04. 2022).

[7] Meta Platforms, Inc. „React.“ (2022), URL: https://reactjs.org/ (cit. 21. 04. 2022).

[8] Mozilla Foundation. „SVG.“ (2022), URL: https://developer.mozilla.org/en-US/docs/Web/SVG (cit. 21. 04. 2022).

[9] inkscape.org. „Inkscape.“ (2022), URL: https://inkscape.org/ (cit. 21. 04. 2022).

[10] S. Simon. „HTML5 Canvas vs. SVG vs. div.“ (2011), URL: https://stackoverflow.com/questions/5882716/html5-canvas-vs-svg-vs-div (cit. 21. 04. 2022).

[11] Mozilla Foundation. „HTML.“ (2022), URL: https://developer.mozilla.org/en-US/docs/Web/HTML (cit. 21. 04. 2022).

[12] ——, „Canvas API.“ (2022), URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (cit. 21. 04. 2022).

[13] d3js.org. „D3.js.“ (2022), URL: https://d3js.org/ (cit. 21. 04. 2022).

[14] axios. „axios.“ (2022), URL: https://axios-http.com/docs/intro (cit. 21. 04. 2022).

[15] A. Dan. „Redux Toolkit createSlice.“ (2022), URL: https://redux-toolkit.js.org/api/createslice (cit. 21. 04. 2022).

[16] ——, „Redux Toolkit store configuration.“ (2022), URL: https://redux-toolkit.js.org/api/configureStore (cit. 21. 04. 2022).

[17] Django Software Foundation. „Django data models.“ (2022), URL: https://docs.djangoproject.com/en/4.0/topics/db/models/ (cit. 21. 04. 2022).

[18] django-rest-framework.org. „Django Views.“ (2022), URL: https://www.django-rest-framework.org/api-guide/views/ (cit. 21. 04. 2022).

[19] Postman, given=Inc., giveni=I. „Postman.“ (2022), URL: https://www.postman.com/ (cit. 22. 04. 2022).

[20] django-rest-framework.org. „Django Serializers.“ (2022), URL: https://www.django-rest-framework.org/api-guide/serializers/ (cit. 21. 04. 2022).

[21] Django Software Foundation. „Django Admin Page.“ (2022), URL: https://docs.djangoproject.com/en/4.0/ref/contrib/admin/#:~:text=One%20of%20the%20most%20powerful,an%20organization's%20internal%20management%20tool. (cit. 21. 04. 2022).

[22] cypress.io. „Cypress.“ (2022), URL: https://www.cypress.io/ (cit. 21. 04. 2022).

[23] A. Jesin. „How To Configure Nginx as a Web Server and Reverse Proxy for Apache on One Ubuntu 18.04 Server.“ (2018), URL: https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-web-server-and-reverse-proxy-for-apache-on-one-ubuntu-18-04-server (cit. 21. 04. 2022).

Přílohy

Odkaz na GitHub - `https://github.com/TheRamsay/school-navigation`
Manuál k dokumentaci - `V4C-HumlDominik-1f-manual.pdf`
Zdrojový kód - `V4C-HumlDominik-1f-code.zip`