Střední průmyslová škola elektrotechnická  
a Vyšší odborná škola Pardubice

**STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ**

**MATURITNÍ PRÁCE – WEBOVÉ STRÁNKY**

**CINEMA TOWN**

březen 2024 Jan Rožek, 4.D

*„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.*

*Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“*

*V Pardubicích dne ........................... ................................................*

*podpis*

*Obsah obrázku text, snímek obrazovky, dokument, Písmo

Popis byl vytvořen automaticky*

*Obsah obrázku text, snímek obrazovky, Písmo, dopis

Popis byl vytvořen automatickyAnotace*

*Webová aplikace pro rezervování sedadel na promítání filmů. Bude umožňovat správu kin, filmů, promítání, žánrů filmů a možnost úpravy oprávnění u některých rolí. Přihlašování a registrace, bude řešena dvoufázovým ověřením za pomoci emailů.*

*Klíčová slova: rezervování sedadel, dvoufázové ověřování*

*Annotation*

*Web application for movie screening seat reservation. It will allow management of theaters, movies, screenings, movie genres and the ability to edit permissions for some roles. Login and registration will be handled by two-phase verification using emails.*

*Keywords: seat reservation, two-phase verification.*

**Obsah**

[Úvod 10](#_Toc159827334)

[1.1 CineStar 11](#_Toc159827335)

[1.1.1 Kladné stránky 11](#_Toc159827336)

[1.1.2 Záporné stránky 12](#_Toc159827337)

[1.2 Cinema City 12](#_Toc159827338)

[1.2.1 Kladné stránky 13](#_Toc159827339)

[1.2.2 Záporné stránky 13](#_Toc159827340)

[1.3 Kino Světozor 13](#_Toc159827341)

[1.3.1 Kladné stránky 14](#_Toc159827342)

[1.3.2 Záporné stránky 14](#_Toc159827343)

[2 Návrh projektu 15](#_Toc159827344)

[2.1 Cílové skupiny 15](#_Toc159827345)

[2.2 Popis projektu 15](#_Toc159827346)

[2.3 Use Case diagram 15](#_Toc159827347)

[2.4 Databáze 16](#_Toc159827348)

[3 Použité technologie 17](#_Toc159827349)

[3.1 JavaScrip 17](#_Toc159827350)

[3.1.1 TypeScript 17](#_Toc159827351)

[3.1.2 React 17](#_Toc159827352)

[3.1.3 Node.js 18](#_Toc159827353)

[3.1.4 NPM 19](#_Toc159827354)

[3.2 Java 19](#_Toc159827355)

[3.2.1 Spring | Spring Boot 19](#_Toc159827356)

[3.2.2 Maven 19](#_Toc159827357)

[3.3 MongoDB 19](#_Toc159827358)

[4 Backend projektu 21](#_Toc159827359)

[4.1 Struktura kódu 21](#_Toc159827360)

[4.1.1 Controllery 22](#_Toc159827361)

[4.1.2 Modely 23](#_Toc159827362)

[4.1.3 Servisy 23](#_Toc159827363)

[4.1.4 Repositáře 24](#_Toc159827364)

[4.2 Princip komunikace 25](#_Toc159827365)

[4.3 Koncové body a HTTP metody 25](#_Toc159827366)

[4.3.1 Controllery se standartním fungováním 26](#_Toc159827367)

[4.3.2 Controllery s deaktivovanými koncovými body 26](#_Toc159827368)

[4.3.3 Controllery s poupraveným chováním 27](#_Toc159827369)

[4.3.4 AuthController 29](#_Toc159827370)

[5 Frontend projektu 30](#_Toc159827371)

[5.1 Struktura kódu 30](#_Toc159827372)

[5.1.1 Volání API 30](#_Toc159827373)

[5.1.2 Modely 31](#_Toc159827374)

[5.1.3 Stránky 31](#_Toc159827375)

[5.2 Routování 32](#_Toc159827376)

[6 Popis jednotlivých částí aplikace 33](#_Toc159827377)

[6.1 Úvodní stránka / Program 33](#_Toc159827378)

[Závěr 35](#_Toc159827379)

[Seznam přístupových údajů 36](#_Toc159827380)

[Seznam použité literatury a zdrojů obrázků 37](#_Toc159827381)

[Seznam obrázků 38](#_Toc159827382)

[Přílohy 39](#_Toc159827383)

# Úvod

Jako maturitní projekt jsem se rozhodl vytvořit zjednodušenou aplikaci pro rezervování sedadel na promítání filmů, která umožní kompletní správu kin včetně sálů, filmů a jejich promítání. Také bude umět posílat uživatelům, kteří jsou přihlášeni, k odběru novinky (nově přidané filmy).

Tato aplikace by neměla sloužit sama o sobě k zisku peněz. Byla by doplňkem k existujícímu podnikání ve filmovém průmyslu. Poskytla by jednoduše dosažitelné informace pro zákazníky stejně jako umožnila jednoduchou správu zaměstnancům za to zodpovědným.

Ke konci studia na střední škole jsem chtěl zhotovit nějakou webovou aplikaci, za kterou by se nemusela (s přivřenýma očima a s přihlédnutím, že na tom budu pracovat sám) stydět žádná větší firma jak po vzhledové, tak po funkční stránce.

Pro tento konkrétní projekt jsem se rozhodl, protože mi jsou filmy celkem blízké, protože jsem si nedokázal vybrat zaměření internetového obchodu, který se stal pro maturitní práce standardem.

Aplikace bude rozdělena do tří serverů, Node.js s pomocí knihovny React pro frontend, Spring Boot pro backend a MongoDB jako databáze zajistí uchovávání dat. Pro toto řešení jsem se rozhodl, protože bych se chtěl zdokonalit ve výše uvedených technologiích a to, protože v nich vidím největší budoucnost.

Analýza obdobných webových stránek

Analýzu je důležité provádět při každém vývoji nové aplikace zejména proto, abychom zjistili, jakou můžeme nabídnout konkurenční výhodu, čemu se vyhnout, anebo co je dobré z ostatních projektů převzít, případně vylepšit.

## CineStar

Adresa: [cinestar.cz](https://www.cinestar.cz)

Obsah obrázku text, oblečení, snímek obrazovky, Digitální kompozice

Popis byl vytvořen automaticky

Obrázek 1: CineStar (zdroj: vlastní)

CineStar je síť multikin s největší hustotou pokrytí v republice. Účelem webové stránky je informovat širokou veřejnost o programu kin (případně i jejich poloze) a samozřejmě umožňuje rezervaci míst na určitá promítání filmů. Po prvním vstoupení na stránku se ukáže mapa republiky s možností vybrat kino jehož program uživatele zajímá. Stránka s detaily kin se pak liší jen nejspodnější částí, kde se nachází program kina s jeho adresou a kontaktem. Toto řešení mi nepřijde nejlepší, protože uživatele, byť jen poprvé, zdržuje od vybrání filmu. Dále bych ocenil možnost v detailu filmu mít možnost provést rezervaci.

### Kladné stránky

* Přehledný program
* V programu jsou náhledy na filmy
* Filmy je možné filtrovat pomoví kategorií

### Záporné stránky

* Složitější cesta k programu
* V detailu filmu není možné provést rezervaci
* Je možné si prohlédnout filmy promítané jen ve vybraném kině

## Cinema City

Adresa: [cinemacity.cz](https://www.cinemacity.cz)

Obsah obrázku text, snímek obrazovky, grafický design, osoba

Popis byl vytvořen automaticky

Obrázek 2: Cinema City (zdroj: vlastní)

Cinema City je nadnárodní síť multikin s kořeny v Izraeli. Účel webové stránky se stejný jako u CineStar, a to informovat a rezervovat. Na úvodní stránce je možnost si prohlédnout nabídku filmů. Z počátku všechny filmy v různých kategoriích (promítáme, budeme promítat, speciální projekce a rodinné filmy), nebo při výběru kina už konkrétní program na určitý den (s výjimkou projekcí naplánovaných na dlouho dopředu). U každého filmu je jeho náhled (při kliku na něj se přehraje trailer na film) a nejpodstatnější informace o filmu. Posledním zobrazením filmů je „Podle filmu“ kde se po vybrání konkrétního filmu zobrazí všechna promítání tohoto filmu ve vybraný den.

### Kladné stránky

* Trailery na filmy
* Je možné rezervovat promítání z detailu filmy
* Možnost prohlédnout si celou nabídku filmů

### Záporné stránky

* Po vybrání kina se není možné vrátit do stavu nevybraného kina
* Vyhledávání filmu podle žánru je možné jen při vybraném kině
* Program by mi více vyhovoval jako one page s odkazy v hlavičce na jednotlivé dny

## Kino Světozor

Adresa: [kinosvetozor.cz](https://www.kinosvetozor.cz)

**Obsah obrázku text, snímek obrazovky, software, Multimediální software

Popis byl vytvořen automaticky**

Obrázek 3: Kino Světozor (zdroj: vlastní)

Stránka spojuje program několika menších pražských kin. I proto si může stránka dovolit být poměrně jednoduchá a přehledná. Nejdůležitější částí stránky je filtrovací formulář a program, který je umístěný hned pod ním. Program je možné přepínat mezi časovým (ten je vidět na ukázce) a filmovým (skládá se z dlaždic s náhledem na film a jeho jménem). Detail filmu se zobrazí jako popup s termíny projekce, ale rezervovat je možné jen nejbližší termín promítání.

### Kladné stránky

* V detailu filmu je možnost přehrát si trailer
* Přehledný program
* Jednoduchý a funkční dezign

### Záporné stránky

* Není možné vyhledávat podle žánrů
* V detailu filmu jsem byl chvilku zmatený při vybírání konkrétního promítání
* Na stránce není moc obsahu pro inspiraci

# Návrh projektu

## Cílové skupiny

Lidé, kteří rádi chodí do kina, aby se podívali na film.

## Popis projektu

Projekt jako celek bude rozdělen do tří samostatných částí (serverů). O ukládání dat se bude starat databázový server MongoDB, přístup k němu bude zprostředkovávat backendová část napsaná v jazyku Java za použití frameworku Spring Boot skrze REST API. O prezentaci dat uživateli se bude starat JavaScriptový kód s využitím knihovny React, který bude spuštěn v prostředí Node.js. Toto řešení by do budoucna umožnilo snadnou rozšiřitelnost funkcionality webové aplikace i například vytvoření mobilní aplikace (bylo by třeba jen napsat aplikaci, protože API už by bylo hotové).

## Use Case diagram

Obsah obrázku text, snímek obrazovky, kruh

Popis byl vytvořen automaticky

Obrázek 4: Use Case diagram

## Databáze

Obsah obrázku text, diagram, snímek obrazovky, Plán

Popis byl vytvořen automaticky

Obrázek 5: struktura databáze UML Class diagram

Na výše uvedeném diagramu je vidět struktura jednotlivých kolekcích v databázi (kolekce v NoSQL databázích představuje to co v relačních databázích tabulka), do kolekce se ukládají data ve formě dokumentů (dokument v NoSQL databázích představuje pak to co by v relačních databázích byl jeden záznam). Protože NoSQL nemá žádnou standardizovanou formu pro modelování grafů (jako ERD relačních databází), uvádím zde UML Class diagram, který výstižně zachycuje strukturu dat, kterou jsem použil.

Výše uvedený diagram reprezentuje strukturu databáze přesně až na jednu výjimku. Diagram např. říká, že v dokumentu Role jsou uložené dokumenty Permission. Toto řešení by ale mělo hned několik problémů počínaje porušením pravidel normalizace dat, přes zbytečně vysoké nároky na kapacitu databáze až po velmi náročnou editaci. Proto je ukládání dokumentů do dokumentů řešeno nahrazením dokumentu záznamem DBRef s parametrem kolekce, ve které se nachází nahrazený dokument a parametrem id nahrazeného dokumentu.

# Použité technologie

## JavaScrip

JavaScript je objektově orientovaný, interpretovaný jazyk (překládaný za běhu) s C-like syntaxí a dynamickým typováním, který využívá funkcionální paradigma (umožňuje uložení funkcí do proměnných). Jeho účel byl doplnit značkovací jazyky HTML a CSS na webových stránkách (manipulace s domem, složitější chování částí webu apod.) Dnes ale najde uplatnění i jako plnohodnotný programovací jazyk, v kterém může být napsaný i například program.

### TypeScript

Je nadstavba JavaScriptu, kterou vyvíjí Microsoft. Jedná se o samostatný programovací jazyk, který se kompiluje do JavaScriptu. Jeho přínos spočívá v zavedení statického typování do JavaScriptu, pohlídá volání nedefinovaných vlastností, zavádí třídy, rozhraní a výčtové typy. Přichází i s genericitou a union typy.

### React

React je open-source JavaScriptovou knihovnou, pro tvorbu UI (user interface), za jehož vývojem stojí společnost Meta a komunita individuálních vývojářů. React funguje na bázi tzv. „komponent“. Ty mají své vstupní vlastnost a vnitřní stavy, což usnadňuje vytváření znovupoužitelného kódu.

React dokáže automaticky reagovat na změnu stavů a překreslovat aplikaci jen když je to potřeba. Komponenty mohou být vytvořeny buď jako funkce nebo jako třídy.

Pokud je komponenta vytvořená jako funkce, pak musí vracet jeden JSX tag (JavaScript XML). Pokud je vytvořena jako třída, tak musí obsahovat metodu „render“, která vrací jeden JSX tag.

JSX má dost podobnou syntaxi jako HTML 5 s následujícími rozdíly:

* Je možné do něj vkládat reactové komponenty jako tagy
* Vlastnosti tagů mohou mít částečně jiný název (class je className, onclick je onClick )
* Mezi složené závorky {} je možné vkládat JavaScriptová data
* Každý tag musí být ukončen
* Všechny tagy jsou párové a jdou zapsat zkráceným zápisem
* Existuje prázdný tag, který se nevykreslí na stránku. Má využití, například když chceme, aby komponenta vracela více jak jeden tag.

JSX může vypadat následovně:

        <>

            {/\*\* Komentář \*/}

            {/\*\* Vložení funkce k eventu onClick \*/}

            <div className='management-reference' onClick={() => navigate(url)}>

                {/\*\* ukázka zkráceného zápisu \*/}

                <div className='management-reference-img' style={imgBg} />

                {/\*\* vložení komponenty filter \*/}

                <Filter />

                {/\*\* důkaz o párovosti všech tagů \*/}

                <br></br>

                {/\*\* vložení textu do odstavce \*/}

                <p>{text}</p>

            </div>

        </>

Webové rozhraní vytvořené pomocí reactu je pak SPA (Single Page Application – celá aplikace je obsažena v jednom HTML a jedno JS souboru).

### Node.js

Je prostředí, které umožňuje spouštění JavaScriptového kódu mimo webový prohlížeč. Jeho primárním účelem je tvorba serverové části webových aplikací s důrazem na vysokou škálovatelnost (schopností obsloužit mnoho připojených klientů současně). Té dosahuje tím, že jednotlivé požadavky klientů zachytává jedno vlákno, která je pak distribuuje mezi asynchronní vlákna. Asynchronní vlákna pak požadavky zpracovávají.

### NPM

Node.js ackage Manager je správce JavaScriptových balíčků, který usnadňuje jejich instalaci a údržbu.

## Java

Je objektově orientovaný programovací jazyk se striktním typováním a C-like syntaxí. Programy napsané v Javě jsou platformě nezávislé (i na operačním systému, tak na mikroprocesoru). Tuto vlastnost má díky JVM (Java Virtual Machine). Implementuje automatickou správu paměti (garbage colletor).

### Spring | Spring Boot

Jedná se o Framework pro jazyk Java, jehož cílem bylo ulehčení vývoje wabových aplikací v Javě.

Čistý Spring nepoužívám, protože by to vyžadovalo nastavení tzv. servlet kontejneru (webového serveru). Nastavení servlet kontejneru se obecně považuje za náročnější akci do, které jsem se nechtěl pouštět a místo toho jsem použil knihovnu Spring Boot, která má nastavený kontejner už implementovaný (tomcat). Dalším důvodem pro využití Spring Bootu pro mě bylo to, že Spring pouze definuje rozhraní, zatímco ve Spring Bootu jsou už i implementována.

### Maven

Je správce balíčků, který je možné použít pro Java projekty. Zařizuje přidání Spring Bootu a jiných knihoven do projektu.

## MongoDB

Jako databázi jsem zvolil dokumentově orientovanou NoSQL databázi MongoDB. MongoDB neukládá data v tabulkách a už vůbec ne v řádcích. Data jsou ukládána ve formátu BSON (Binary JSON) a jsou organizována ve formě dokumentů (v relačních databázích bychom to nazvali řádek), které jsou seskupeny do kolekcí (v relačních databázích bychom řekli tabulek). Každý dokument ale může mít jinou vnitřní strukturu (aplikace si musí sama hlídat jaká data do databáze ukládá).

Zatímco vztahy v relačních databázích jsou řešeny pomocí cizích klíčů MongoDB zná dvě vazby mezi objekty. Objekt může obsahovat jiný objekt anebo může obsahovat referenci (odkaz). Druhý způsob je podobný cizím klíčům relačních databází, ale přichází s problémem integrity dat, kterou musí opět vyřešit aplikace.

# Backend projektu

Backendová část je napsaná jako REST API (Application Programming Interface) v programovacím jazyce Java s použitím frameworku Spring Boot, který využívá pro uchování dat NoSQL databázi, MongoDB.

## Struktura kódu

Aplikace odpovídá třívrstvé architektuře. Prezentační vrstvu představují třídy Controller a DTO (zástupné třídy, které nesou data. Slouží pro jednodušší komunikaci s frontendem). Aplikační vrstva je tvořena Modely třídami Service. Datovou vrstvu představují rozhraní Repository.

Obsah obrázku text, snímek obrazovky, Písmo, dokument

Popis byl vytvořen automaticky

Obrázek 6 Struktura backendové části

### Controllery

Controller registruje metody pro obsluhu určitého dotazu k určité cestě na webovém serveru (URI). Až na třídu AuthController, všechny dědí z abstraktní třídy RestController. RestController definuje základní zpracování HTTP požadavků GET, POST, PUT a DELETE. Spolu s nimi deklaruje ještě dvě konstanty modifikátorem dostupnosti protected pro získávání JWT z hlaviček HTTP požadavků. Poslední vlastnosti ResrControlleru, které v něm jsou definovány jsou dvě proměnné rovněž s dostupností protected. První z nich je generická proměnná service, která musí rozšiřovat abstraktní třídu CrudService. Závislost na toto komplexní entitu je přidána pomocí návrhového vzoru Dependenci Injection, za pomocí anotace @Autowired. Stejným způsobem je vytvořena i závislost na entitu třídy ObjectMapper. Instance této třídy převání data, která metody vracejí na textové řetězce formátu JSON. Data, která controller zpracovává jsou určena genericky a musí implementovat rozhraní Entity.

Toto řešení pomocí abstraktní třídy RestController řeší ohromnou redundanci kódu a také dovoluje nechat většinu ostatních Controllerů téměř prázdných, aniž by zabránila snadné úpravě chování pro určitý typ dat. V konkrétním controlleru je třeba definovat kromě typu dat i konkrétní Service a URI cestu ke koncovým bodům. V konkrétním controlleru může být definována i obsluha pro jiný koncový bod, než jaké definuje RestController.

@RestController

@CrossOrigin(origins = {"https://www.mp.home-lab.rozekja.fun"})

@RequestMapping(path = "/api/age\_categories")

public class AgeCategoryController extends cz.rozek.jan.cinema\_town.controllers.RestController<AgeCategory, AgeCategoryService> {

}

Celá třída AgeController

AuthController je poté zodpovědný za obsluhu uživatelských požadavků volaných za účelem autentizace.

@org.springframework.web.bind.annotation.RestController // Díky anotaci bude řída rozpoznána jako RestController

@CrossOrigin(origins = {"https://www.mp.home-lab.rozekja.fun"}) // povolení zdroje dotazů na API (URL ze které přijde dotaz)

@RequestMapping(path = "/auth") // určení URI cesty k endpointům

public class AuthController {

Anotace definující controller

### Modely

Třídy modelů definují strukturu dat a základní operace na nich. Například metodu, která zkontroluje, zda je instance validní.

#### Primární Modely

Modely, které bych nazval jako primární reprezentují data, tak jak jsou uložena v databázi. Všechny tyto modely implementují rozhraní Entity. Toto rozhraní deklaruje metody get() a set() pro Id typu String a metodu validate() s návratovým typem void a možností vyvolat výjimku ValidationException (k vyvolání výjimky dojde pokud je metoda zavolána na nevalidní instanci).

Každý primární model těží ze tří anotací:

@Document("films") // určí, že jde o třídu, jejíž instance je možné uložit do databáze mongoDB a to to kolekce 'films'

@Data // anotace poskytne metody get a set pro všechny vlastnosti třídy

@NoArgsConstructor // zaručí třídě bezparametrický konstruktor

public class Film implements Entity {…}

Ukázka anotací primárního modelu

#### DTO (Data Transfer Object)

Objekt DTO slouží k zapouzdření dat pro jejich posílání mezi částmi aplikace (takže i většina primárních modelů funguje i jako DTO). Jednou takovou třídou je například třída TokenDeviceId, která slouží k obalení přihlašovacího JWT a JWT, který slouží pro udělení důvěry zařízení, ze kterého se uživatel přihlašuje.

### Servisy

Tyto třídy implementují logiku práce s daty. Podobně jako u Controllerů je zde abstraktní třída, konkrétně CrudService, která obsahuje výchozí implementaci metod CRUD.

#### CrudService

Tato třída deklaruje vlastnosti repository a authService s dostupností protected spolu s jejich injectody. Injectory jsou abstraktní metody, které musí každá třída, která bude tuto abstraktní rozšiřovat implementovat.

Kromě základních metod readAll(), readById(), create(), update() a delete() tato třída deklaruje a implementuje i metodu verifyAccess, tato metoda ověří, zda je dotazovaný oprávněný k provedení dané akce. Volání této metody je prováděno v každé z metod CRUD a mělo by k tomu docházet i při přepisování těchto metod v konkrétních třídách service. Aby bylo možné pracovat už na abstraktní vrstvě s konkrétními oprávněními bylo nutné deklarovat abstraktní metody pro získání každého typu oprávnění (jedno oprávnění pro jednu metodu CRUD).

#### AuthService

Třída implementuje logiku pro správu autentizace a autorizace. Při svém zavedení si vytvoří nebo načte páry klíčů pro asymetrické podepisování JWT, které vydává a ověřuje.

#### PdfService

Poskytuje metodu, pro vygenerování lístků (potvrzení platby) ve formátu .pdf.

#### PaymentService

Konstruktor slouží jako injector, která přidá instanci závislost na mapu, která pod typy plateb obsahuje metody pro zpracování platby. Díky tomuto je přidání dalšího způsobu platby pouze otázkou napsání třídy, která bude anotovaná anotací Payment a bude rozšiřovat funkcionální rozhraní IPayment s implementací metody pay(), která zpracuje platbu. Takto napsanou třídu není třeba nikde dál uvádět díky anotaci Payment bude automaticky obsažena v mapě dostupných plateb.

##### EmailService

Poskytuje metody pro načtení rozložení emailu a pro poslání emailu s možností přílohy.

### Repositáře

Každé rozhraní rozšiřuje generické rozhraní MongoRepository. Díky tomu je přístup da databáze řešen frameworkem, který implementuje základní operace CRUD. Pro pokročilejší vyhledávání je zde použita anotace @Query.

    @Query("{ 'halls.?0' : { $exists : true } }")

    Cinema findByHall(String hallId);

Použití @Query pro načtení kina, ve kterém je sál se zadaným id

## Princip komunikace

Komunikace mezi backendem a frontendem je založena na protokolu HTTP (Hypertext Transfer Protocol). Díky nasazení projektu za reverzní proxy se ke komunikaci používá zabezpečená verze HTTPS.

Veškerá data, která si aplikace vyměňují, jsou textové řetězce ve formátu JSON (JavaScript Object Notation). Tento formát dat umožňuje v podstatě jakoukoliv API komunikaci

Pro autorizaci v rámci aplikace slouží JWT (JSON WEB TOKENy). O jejich samotnou implementaci se stará jose4j.

        <dependency>

            <groupId>org.bitbucket.b\_c</groupId>

            <artifactId>jose4j</artifactId>

            <version>0.9.4</version>

        </dependency>

Závislost, která pracuje s JWT

## Koncové body a HTTP metody

Metody HTTP použité pro volání API odpovídají specifikaci. GET – načítá data, POST – předá data, PUT – edituje data, DELETE – odstraní dat. Odpovědi na požadavky mohou vracet stavové kódy:

* 200 (Ok) – když vše proběhne správně
* 202 (přijato) – když při přihlašování bude vyžadována druhá fáze ověření
* 400 (špatný požadavek) – když data nejsou v požadovaném formátu
* 401 (neautorizováno) – když je vyžadována autorizace
* 403 (přístup odepřen) – když uživatel nemá potřebná oprávnění
* 404 (nenalezeno) – když záznam nebyl nalezen
* 405 (metoda není povolena) – když koncový bod existuje, ale nic nezpracovává
* 500 (interní chyba serveru) – Nastala neočekávaná chyba na serveru

### Controllery se standartním fungováním

Tyto controllery poskytují koncové body, které poskytuje abstraktní třída RestController a jejich chování neupravují. 

Obrázek 7 Tabulka s koncovými body standartních controllerů

### Controllery s deaktivovanými koncovými body

Níže zmíněné controllery se od standartních liší tím, že dostupnost některých koncových bodů, které dědí z RestControlleru, nebyla žádoucí.

Získávání všech se sedadel nemá žádný význam a správa sedadel jednoho po druhém by bylo pro uživatele příliš nepohodlná. Správa sedadel je řešena při spravování sálů. Proto jsou níže uvedené metody nedostupné. 

Obrázek 8 Tabulka s koncovými body sedadel

Města jsou uchovávána pro menší redundancí dat. Jsou spravována při správě kin. Jejich samostatné spravování by činnost zbytečně komplikovalo. Oprávnění není možné přidávat a editovat ani mazat už jen jako ochrana proti administrátorům aplikace samotné. Pokud by mělo dojít k nasazení aplikace nejlepší způsob editace těchto dat by byl přímo v databázi osobou k tomu pověřenou. 

Obrázek 9 Tabulka s koncovými body měst a oprávněních

Role jsou implementovány bez moznosti přidání nové nebo smazání staré. Aby bylo možné řešit přidávání a mazání muselo by dojít k ustanovení komplexní politiky rolí (například by bylo třeba řešit které role může spravovat, které role, nebo zda má role oprávnění přidělovat konkrétní oprávnění). V kontextu celé této aplikace se tato problematika zdá být zbytečná. Pokud by bylo nejhůř osoba za to zodpovědná by mohla vytvořit novou roly přímo v databázi. 

Obrázek 10 Tabulka s koncovými body rolí

### Controllery s poupraveným chováním

Jedná se o controllery, které upravují nebo přidávají koncové body.

Controller pro kina přidává koncový bod, který vrací kino, ve kterém se nachází sál d daným id. 

Obrázek 11 Tabulka s koncovými body kin

U filmů je třeba s API získat filmy, které se mají použít pro galerii na úvodní stránce (aktuální trháky). Dále controller disponuje koncovým bodem pro nahrání obrázku k filmu.

Obrázek 12 Tabulka s koncovými body filmů

Aby bylo možné zachovat integritu databáze není z ní možné odebrat ty sály, ve kterých už bylo naplánováno nějaké promítání. Uživatel na stránce vidí, které sály může odebrat a které ne. Pro zjištění toho, které sály je možné odebrat a které by se odebrat nepovedlo, poskytuje specifický koncový bod. 

Obrázek 13 Tabulka s koncovými body sálů

Při získávání naplánovaných promítání je pro aplikaci důležité, zda promítání už proběhlo, nebo ne. Controller upravuje metodu pro získání všech záznamů tak, že vrací pouze budoucí promítání. Také umožňuje získat promítání, která budou probíhat podle id konkrétního filmu. Aby bylo možné prohlížení už proběhlých představení implementuje ještě koncový bod pro archivovaná promítání. 

Obrázek 14 Tabulka s koncovými body představeních

Při vytváření rezervace dochází okamžitě k provedení platby, takže editace rezervace možná není. K rezervová jsou potřebná jiná data, než předpokládá abstraktní RestController a proto je i standartní koncový bod pro vytvoření deaktivován a nahrazen jiným. Aby bylo možné zjistit, jaká místa už jsou na jakém promítání už rezervovaná poskytuje controller ještě koncový bod pro „cenzurované načtení“ rezervací pomocí id promítání (Cenzura znamená odebrání konkrétních typů lístků, uživatele a data, kdy rezervace proběhla).



Obrázek 15 Tabilka s koncovými body rezervací

### AuthController

Koncové body tohoto controlleru neslouží pro vyměňování dat aplikace jako všechny ostatní, ale jsou zodpovědné za autentizaci uživatele. 

Obrázek 16 Tabulka s koncovými body pro autentizaci

# Frontend projektu

Frontendová část je napsaná v jazyku JavaScript a běží v prostředí Node.js s použitím knihovny React s nadstavbou TypeScript.

## Struktura kódu

Kód je strukturován do balíčků:

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Obrázek 17 Struktura frontendové části

* assets – uložení obrázků
* components – komponenty, balíček se dělí na další balíčky. Každý vnořený balíček souvisí se stejně jmennou stránku
* global\_functions – zde jsou umístěny funkce volané na více místech aplikace. Např.: generické volání API
* layouts – obsahuje layouty
* models – balíček obsahuje, šablony pro datové struktury
* pages – balíček obsahuje komponenty, které vykreslují celé stránky

### Volání API

Volání API je zpracováno genericky v souboru ServerApi. Funkce pro volání API jsou asynchroní. Pro výměnu dat slouží tyto:

* **loadData()** – Slouží pro načtení dat z backendu. Generický parametr určí typ vrácených dat. Parametr modelEndpoint určí koncový bod, který se má použít pro volání API. Parametr ids má výchozí hodnotu nastavenou na prázdné pole a určuje, jaké záznamy se mají načíst. Pokud dojde k volání funkce s parametrem ids, který se rovná prázdnému poli, předpokládá se, že je snaha načíst všechna data. Návratová hodnota je vždycky pole záznamů.
* **storeData()** – Slouží pro uložení dat na backend. Generický parametr určuje typ záznamů, které vrátí volání API. Metoda pak vrací pole záznamů, které jsou typu Entity. Parametr modelEndpoint určí koncový bod, který se má použít pro volání API. V parametru data, jsou záznamy, které se mají nahrát na backend. To, zda se použije HTTP metoda POST nebo PUT se rozhodne podle vlastnosti id. Když je id null nebo undefined, tak, se volá POST jinak se volá PUT. Funkce vrací pole Entit (typu Entity), která dostane jako odpovědi na požadavky.
* **deleteData()** – Slouží pro smazání dat v backendu. Generický parametr určuje typ dat parametru data. Parametr modelEndpoint určí koncový bod, který se má použít pro volání API. Při úspěšném odebrání metoda vrací data, která dostala jako parametr.

Parametr modelEndpoint je typu ModesEndpoints. ModesEndpoints je výčtový typ (enum), který obsahuje dostupné koncové body. Tyto metody chyby nezpracovávají, ale vyhazují je výš (z důvodu debugování jsou dotazy na API v try catch blocích i přesto, že v bloku catch výjimku pošlou víš).

### Modely

Modely jsou TypeScriptové třídy, které odpovídají modelům v backendové části. Aby bylo možné použít generické parametry v API metodách musejí modely rozšiřovat třídu Entity. Ta definuje parametr id. Modely, které nerozšiřují Entity jsou ty, které odpovídají DTO modelům (ReservationDTO, TokenDeviceId). V souborech s definicí modelu je i definována instance to usnadňuje vložení úvodních dat do formuláře.

### Stránky

Stránky jsou, stejně jako jejich části, reactové komponenty. Jsou rozděleny do balíčků podle to ho k čemu souží.

* usersPages – stránky pro standartní chování na webu (domovská stránka, stránky s filmy, stránka pro rezervování …)
* management – zde jsou umístěny stránka pro zobrazení a editaci dat
* login – v tomto balíčku je nachází stránky pro přihlášení, registraci, ověření a pro změnu a obnovu hesla

## Routování

Směrování zajišťuje komponenta App.tsx prostřednictvím BrowserRouter. Jednotlivé routy obsahují parametry:

* path – URL adresa stránky
* element – element, který se má na stránku vykreslit

Routy se mohou zanořovat do jiných. Obsah elementů ve vnořených routách se do nadřazených pomocí komponenty <Outlet/>

   <BrowserRouter>

      <Routes>

        <Route path='/' element={<MainLayout />}>

          <Route index element={<Home />} />

          {/\*\* Stránky pro přihlašování \*/}

          <Route path='/register' element={ (verifyAccess() ? loginErr : <Register />) }/>

          <Route path='/login' element={ (verifyAccess() ? loginErr : <Login />) }/>

          {/\*\* Stránky pro rezervování \*/}

          <Route path='/film/:fimlId' element={ <FilmDetail /> }/>

          <Route path='/my-reservation/:userId' element={ (verifyAccess() ? <MyReservations /> : accessDenite) } />

          <Route path='/management'>

            <Route path='' element={ (verifyAccess("projection-create") ? <Management /> : accessDenite) } />

Ukázka části routovací komponenty

Jak je vidět v ukázce, tak všechny stránky používají jeden základní layout, který je hodně obecný a specifikuje jen navigaci a footer. Elementy se jsou vyplňovány na základě ověření práv.

# Popis jednotlivých částí aplikace

V této části se budu věnovat jednotlivým stránkám, které si může uživatel zobrazit.

## Úvodní stránka / Program

Po otevření webové stránky uživatele překvapí rovnou stránka s programem, která je rozdělena do tří částí (navigaci nepočítám).

První části tvoří automatická galerie obrázků filmů, které byly zvoleny jako propagační. Galerie v nezmenšené formě obsahuje dva filmy a ve zmenšené verzi jeden. Obrázky se automaticky mění po 10 sekundách. Nové filmy přijíždí z prava

Obsah obrázku text, oblečení, osoba, plakát

Popis byl vytvořen automaticky

Obrázek 18: Úvodní stránka 1. část

Druhou část tvoří filtr, ten umožňuje vybrat jedno kino a také použít vyhledávací pole, které vyhledává automaticky pro psaní a porovnává podle: jména filmu a názvu každého žánru každého filmu. Pokud není vybráno žádné kino (počáteční stav), tak je jedno v jakém kině bude probíhat promítání. Vyhledávací okno nerozlišuje malá a velká písmena, myslím si, že je to tak lepší (pro uživatele pohodlnější). Poslední částí filtru je pak čtrnácti denní přehled. Po kliknutí na den je uživatel přesunut na sekci, kde začínají filmy, které se v daný den budou promítat

Poslední, třetí, částí úvodní stránky jsou pak samotná naplánovaná promítání. Ta jsou dělena podle dnů, kdy k nim má dojít, do sekcí. Sekce jsou uspořádané podle data ze shora dolů. Promítání se skládá z obrázku filmu, nadpisu, stručného popisu filmu a dalších informací, které by mohly budoucího návštěvníka zajímat (žánry, režisér, herci, doba trvání …). Favikon uprostřed obrázku vyzívá uživatele, aby si přehrál trailer k filmu (trajler je umístěn na YouTobe a do stránky jen vložen). Uživatel se může přes název filmu překliknout na detail filmu a odtud provést rezervaci.

Obsah obrázku text, snímek obrazovky, Webové stránky, Webová stránka

Popis byl vytvořen automaticky

Obrázek 19: Úvodní stránka část 2 a 3

# Závěr

Popište význam projektu, zdůvodněte problémy při řešení daného tématu a odchylky od zadání, popište využití maturitní práce a plány do budoucna.

# Seznam přístupových údajů

URL adresa webu: [www.mp.home-lab.rozekja.fun/](http://www.mp.home-lab.rozekja.fun/)

|  |  |  |
| --- | --- | --- |
| Úroveň oprávnění | Přihlašovací jméno | Heslo |
| Administrátor | ucet1.cinema.town@gmail.com | J0w89jROWMHVzlDiriw4 |
| Registrovaný uživatel | ucet2.cinema.town@seznam.cz | 8uWeVRsncX3NsUmkX3xx |

# Seznam použité literatury a zdrojů obrázků

Vložte seznam literatury a on-line stránek, které jste použili pro vyhledávání informací a studium.

Do seznamu zahrňte zdroje obrázků použité v písemné práci i v samotném webu (uveďte práva k použití – používejte pouze obrázky zdarma použitelné pro komerční i nekomerční využití). Pokud jste některé obrázky vytvořili sami (vlastní grafika či vlastní fotografie), uveďte jejich seznam nebo popis a zdroj vlastní.

Seznam můžete vygenerovat na [www.citace.com](http://www.citace.com)

# Seznam obrázků

Vygenerujte seznam obrázků použitých v dokumentaci (obrázky musí mít zapsané titulky) včetně čísel stránek (stránky menší než polovina stránky mohou být vloženy ve stránce, bez obtékání textu, velké obrázky budou v příloze).

# Přílohy

Use case diagram

Responzivita – vložte obrázek webu na počítači, na mobilu, příp. na tabletu.