Gymnázium, Praha 6, Arabská 14

Obor programování

ROČNÍKOVÝ PROJEKT

Vojtěch Sobotka

ELibrary

Březen 2021

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole 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 ……. Vojtěch Sobotka ……

Název práce: ELibrary

Autor: Vojtěch Sobotka

Anotace: Můj cíl bylo naprogramovat aplikaci, které bude plnit funkci softwaru pro knihovnu. Aplikace bude mít obsahovat jak serverovou část (backend), tak i uživatelské rozhraní(frontend), v rámci své práce jsem se zabýval několika problémy, jako například navržení vhodné datové struktury, vytvoření vhodného zabezpečení s rozdělením a rolí a v neposlední řadě také se samotným propojením serverové části a uživatelského rozhraní.

Project name: ELibrary

Author: Vojtěch Sobotka

Annotation: The aim of my project was to create a functional application for real-life use in libraries. Application has a server side and a client interface. I faced some problems, when I was creating a working data structure, programing propriety authentication with roles, as well as connecting backend and frontend. My work was created in Spring Boot framework on the server side with MongoDB database and Angular framework on the frontend side.

Zadání

V době kompletace dokumentace z důvodu nezobrazení zadání na serveru nemám oficiální zadání k dispozici. Zadání doplním, jakmile bude k dispozici.

.

Obsah

1. Úvod

Obsahem této práce je popis tvorby webové aplikace pro knihovnu. Mým cílem byla snaha použít co nejvíce technologií a způsobů které se dnes používají při tvorbě webových aplikací.

V rámci projektu jsem se snažil o takzvaný (full stack development) v rámci kterého jsem navrhoval všechny komponenty sám, tedy serverovou i uživatelskou část.

Pro projekt jsem se snažil využít nové technologie pro tvorbu webových aplikací jako například framework Angular, nebo databázi MongoDB.

2. Struktura webové aplikace a zvolené technologie

Webová aplikace se obecně skládá ze dvou hlavních částí, uživatelského rozhraní a serverové části. K tomu musíme ještě počítat také s potřebou komunikačního protokolu, pomocí kterého propojíme serverovou část a klientskou aplikaci, k tomu všemu je také potřeba přidat databázi pro práci s velkým objemem dat který skrz aplikaci projde.

V dnešní době se z velké části těchto úkolů používají moderní frameworky, které značně zjednodušují práci.

2.1. Zvolené technologie backendu

Úkolem serverové části je zpracovávat samotná data, ukládat je do databáze a na základě žádosti je posílat do klientské aplikace. Backend tedy tvoří v podstatě most mezi klientem a daty, která ale musí zpracovat, či vyhledat správná.

V dnešní době je na trhu relativně velké množství frameworků. Můžeme je primárně rozdělit podle technologií na kterých daný framework běží, největší skupinou jsou jistě frameworky běžící na technologii NodeJs, těchto frameworků je prakticky nepřeberné množství. Většina z těchto frameworků používá jazyk JavaScript či jeho mutaci TypeScript. Původně jsem měl v plánu postavit celou moji práci na frameworku LoopBack 4. Výhody tohoto frameworku postaveného na populárním frameworku Expres jsou zejména relativně málo kódu, zejména díky jazyku Typescript. Framework také relativně dobře pracuje s rozhraním OpenApi, které je užitečné při výrobě Rest Api. Nevýhodami pro mě byla zejména z mého pohledu nedostatečná dokumentace pro moji práci (Zejména v oblasti vazeb dat a zabezpečení).

Nakonec Jsem se rozhodnul použít framework Spring Boot, který používá Maven pro zajištění svých buildů. Výhodami Springu je zejména relativně dobrá čitelnost kódu. V porovnání s Javascriptovými jazyky je Spring o dost čitelnější pro programátora. Také rozumím jazyku Java o dost více než Javascriptu. Nevýhodou Springu a obecně Javy je relativně dlouhý kód, sice čitelný ale o dost delší než například JavaScript.

2.2. Technologie frontendu

Účelem uživatelského rozhraní je zejména tvorba interface, do kterého může uživatel zadávat data, či příkazy které backend zpracuje do ucelených žádostí pro backend.

Uživatelské rozhraní tvoří obvykle HTML soubory, do kterých je vložen Javascript pro interaktivní prvky stránek. Pro tvorbu stylů webu se obvykle používá CSS, v dnešní době často zastoupené Bootstrapem. Důležité je při volbě technologie hledět zejména na to, aby dané technologie podporovala většina dnešních webových prohlížečů (Firefox, Chrome, Opera nebo Bing).

Pro tvorbu uživatelského rozhraní je dnes k dispozici několik relativně silných nástrojů, základem je ale vždy znalost HTML a Javascriptu. Můžeme použít například knihovnu React, která rozšiřuje JavaScript o interaktivní prvky pro komunikaci se serverem.

Pro můj projekt jsem se rozhodnul využít frameworku Angular vyvíjeného společností Google. Tento framework je napsaný v jazyce Typescript. Framework je sice relativně složitý na naučení ale poté poskytuje relativně rozsáhlé množství možností zejména v práci s objekty a komunikaci s backendem, zejména je vybaven relativně silným HTTP klientem, pro posílání žádostí na server a jejich zpracování.

2.3. Použitá databáze

Databáze je nutná k uskladnění velkého množství dat, sice většina frameworků nabízí vlastní „in-memory database“, ale ta není vhodná k použití na větších projektech. Databáze musí být rychlá v práci s daty a také mít, pokud možno relativně snadné rozhraní pro vložení dat a následnou práci s nimi.

Pro projekt tohoto typu můžeme použít zejména dva typy databází (grafové databáze nejsou pro použití v rámci mého projektu užitečné). Můžeme použít tradiční SQL neboli relační databázi. Těchto databází existuje prakticky nepřeberné množství. Výhodou je jednoduché použití, relativně dobrá rychlost, možnost tvoření jednoduchých vazeb mezi objekty.

Druhým typem databáze jsou NoSQL databáze, které umožnují ukládání celých objektů, včetně začlenění objektů do objektů, Jako nevýhodu lze vnímat relativně horší práci s vazbami mezi objekty a občas problémy s parsováním objektů. Na druhou stranu

databáze je opravdu hodně rychlá při práci s daty a umožnuje jejich ukládání do JSON objektů.

Rozhodnul jsem se použít pro svůj projekt databázi MongoDB, tato databáze je typu NoSQL, mezi její výhody patří zejména rychlost a možnost psaní příkazů pro databázi v Javascriptu a také široká podpora driverů. Zpětně musím říct, že využité této databáze nebylo tak užitečné, jak se na první pohled mohlo zdát, zejména kvůli horší práci s vazbami.

2.4. Použitá IDE a další technologie:

V rámci práce na projektu bylo potřeba použít další pomocné technologie. Zejména se jedná o různé podpůrné nástroje pro správu buildů, také různé technologie pro zálohování a v neposlední řadě editory pro lepší práci s kódem.

Maven jsem použil jako nástroj pro správu buildů backendu (Frameworku Spring). Maven spouští všechny potřebné pluginy backendu ve správném pořadí.

Pro práci s Angularem je potřeba mít framework NodeJs, na kterém Angular běží. Dále jsem použil pluginy, které výrazně zjednodušují tvorbu kódu v TypeScriptu jako Pretier nebo Tslint.

Backend jsem napsal v editoru IntelliJ Idea, který umožnuje relativně dobře spravovat scripty pro Maven. Front end jsem napsal s využitím editoru Visual Studio Code, které funguje s Typescriptem o dost lépe než například konkurenční Webstorm. Pro editaci rychlé čtení databáze jsem použil MongoDB compass.

Poslední zmíněnou technologií je Postman, pomocí kterého jsem testoval endpointy backendu v průběhu vývoje.

3. Rest Api:

Rest Api (Wikie) je typem používané webové architektury, který jsem se rozhodnul využít ve své práci. Některé prvky této architektury jsou podobné s http protokolem, některé prvky jsou dokonce přejaté a využíval jsem je ve své práci, například http statusové kódy. Aplikace, která využívá principů REST, se obecně dá označit jako RESTful.

Architektura má dva základní poznávací prvky prvním je využití čtyř základních operací pro práci s daty.

- Get Pro získání dat ze serveru, metoda nemá žádné tělo, parametry se dají získat ze samotné adresy, na kterou žádost posíláme.
- Put Používá se pro aktualizaci dat uložených na serveru, v odpovědi obvykle vrací obměněný objekt.
- Post Dotaz s tělem parametrů, obvykle se používá pro ukládání dat, nebo žádost s velkým množstvím parametrů.
- Delete Pro smazání dat ze serveru.

Druhým poznávacím znamením této architektury je forma, kterou jsou data posílána. Možnosti jsou data v xml souboru nebo data ve formátu JSON. V projektu používám právě dat a ve formátu JSON, zejména kvůli dobré návaznosti na MongoDB databázi a také snadnému parsování dat.

4. Serverová část projektu

Serverová část projektu je zodpovědná za práci s databází, autentifikaci uživatelů a v neposlední řadě za samotné plnění dotazů z frontendu. Jak je již zmíněno, server je napsán v jazyce Java a běží na frameworku Spring Boot. V této části se budu věnovat návrhu datové struktury, autentifikaci uživatelů, samotné práce s daty a také odpovídání na žádosti z frontendu

4.1. Datová struktura pro knihovnu

Při návrhu datové struktury je nutné zejména myslet na zvolenou databázi a také na samotný typ dat které budeme chtít zpracovat, je také nutné myslet na to, jakým způsobem budeme chtít data zpracovat. Pro potřeby své aplikace jsem tedy navrhl objekty následovně.

- Genr (žánr) Má atributy id a obsahuje jméno žánru
- Author (autor) Má atributy id a obsahuje jméno autora
- Book (kniha) Má atributy id, titul, popisek knihy pro popisek a rok vydání, dále mám pole pro žánry a autory, pro které používá jako klíč jméno autora nebo žánru. Dále má atribut pro uživatele, jemuž je propůjčena, jako klíč je použito jméno uživatele. Posledním atributem je datum, do kdy má být vrácena, ve formátu Java LocaleDate.
- Role (role uživatele) Model pro rozlišení role uživatele. Má atribut role, která má vlastní enum, pro jednodušší práci s daty.
- User (uživatel) Obsahuje informace o uživatel včetně hashovaného hesla a role.

Jako id používám defaultní string který generuje MongoDB, pomocí tohoto id se dá například určit čas kdy byl objekt vytvořen nebo jakého je typu.

Každý z těchto modelu je uložen v samostatném dokumentu (název pro místo kde jsou uloženy JSON soubory). Samotná data jsou uložena ve formátu JSON jako objekty, což je výhodné pro velké množství dat.

4.2. Vazby mezi daty

MongoDB není určeno pro vazby stejného typu jako se vyskytují v běžných relačních databázích. Tyto vazby lze importovat pomocí anotace *@DBRF*. (Wikie) Této vazby využívám pro udílení uživatelských práv.

Druhým způsobem je použití vlastní klíčů a následné filtrování objektů s těmito klíči, toto využívám ve své práci často, například pro vazby mezi knihou a autorem nebo knihou a žánrem.

Žánr a Autora používám jako referenční prvek pro vyhledávání knih podle těchto parametrů. Nejprve uživatel obdrží seznam všech autorů a žánrů které jsou v knihovně a následně si z nich vybere požadované parametry pro knihy.

4.3. Architektura Spring Serveru

Rest server má svojí danou architekturu, některé prvky architektury je nutné dodržet, jiné zde řadím pouze pro lepší orientaci v serveru.

Ve složce Models se nachází všechny Modely objektů pro databázi MongoDB. Každý model musí mít anotaci @Document pro jméno, podle kterého je následně vytvořena kolekce/dokument v databázi, nutným prvkem je též anotace @Id pro označení neměnitelného id objektu. Id je vždy ve formátu string. Dále dokument obsahuje konstruktor, gettery a settery pro naše objekty. Poslední nutnou částí je zde prázdný konstruktor, který potřebuje MongoDB pro deserializaci objektů.

Další částí struktury je repositář, interface MongoRepository (MongoDB)obsahuje základní 4 funkce pro práci s repositářem

- Create Pro nahrání dat.
- Read Pro čtení dat.
- Update Pro aktualizaci dat.
- Delete Pro smazání dat.

Dále je zde možné psát vlastní metody pro práci s daty pomocí velice jednoduchého MongoDB Query Language. Tento umožnuje prakticky pomocí slov popsat co operaci kterou chceme provést s naším repositářem.

Třetí nutnou částí serveru jsou controllery, které vyznačují samotné endpointy, tedy adresy, na které bude klient posílat příkazy, jim se budu věnovat v samostatné kapitole.

Poslední dvě kategorie v rámci struktury mé aplikace jsou requests a security.

Requesty jsou pomocné třídy pro naše objekty, umožnují nám zpracovávat tělo žádosti podle různých modelů definovaných v této třídě, například login, register atd.

Security obsahuje soubory týkající se JWT autorizace a autentifikace kterou jsem použil v pro zabezpečení projektu.

4.4. Controllery

Controllery (Documentation) jsou třídy které jsou použité pro práci s dotazy, jsou zde definované endpointy, podle kterých jsou vyřizovány žádosti od klienta. V controllech se vyskytuje relativně velké množství anotací.

- RestController – Označuje controller, který je poté kompilován při buildu.
- RequestMapping – Označuje základní cestu do controlleru.
- PostMapping – Označuje, jakou Rest funkci controller vykonává.
- Cors – Cross Control Origin – definuje ze kterého portu se můžeme k aplikace připojit.
- Autowered – Injektuje do controlleru interface repositáře.

Jednotlivé metody používají objekt Response Entity pro odesílání JSON souborů zpět na ke klientu. Třída Response Entity umožnuje odeslat buď výsledný objekt jako odpověď, nebo například http status v rámci odpovědi, například *http status not found nebo no content.*

4.5. Autentifikace

Autentifikace je soubor tříd, která zajišťuje zabezpečení endpointy, do aplikace se tedy dostanou pouze uživatelé, kteří mají svůj profil, také je spojené s autorizací, která rozlišuje mezi uživateli, v našem případě mezi administrátory a prostými uživately.

Pro svůj projekt jsem použil JWT Token autentifikace (Bezkoder, 2021), která generuje Token na základě login žádosti, kterou se uživatel přihlašuje k serveru. V praxi to funguje tak, že uživatel pošle uživatelské jméno a heslo v post request a následně obdrží token a další údaje k autentifikace pro své použití. Token má omezenou živnost nastavenou uživatel, v našem případě 1 hodina. Token uloží klient a interceptor ho při každé další žádosti připojí do hlavičky žádosti. Výhodu JWT tokenu je právě využití lokální paměti prohlížeče, backend pouze validuje tokeny, takže nemusíme mít nastaven další server pro autentifikaci uživatelů. Pro svojí práci jsem částečně použil autentifikaci podle návodu z internetu.

Pro hashovaní hesel používám knihovnu BCrypt, takže administrátor neví, jaké uživatel má heslo pouze výsledný hash.

Třída UserDetails obsahuje detaily uživatele, pomocí kterých je následně generován token.

Třída SecurityConfig je konfigurační soubor celé aplikace, kde jsou nastaveny jednotlivé prvky, jako např Bcrypt nebo nastavené endpointy ke kterým se dá připojit bez hesla (např login)

4.6. Filtrování dat a práce s daty

V rámci knihovny je nutné filtrovat data na základě žádosti uživatelů, v našem případě tedy potřeba najít knihy podle zadaných parametrů uživatele. Uživatel si můžeme vybrat na základě kterých autorů či žánrů chce vidět knihy. To funguje tak že nejprve obdrží seznam všech autorů a žánrů v knihovně a následně si vybere. Odeslanou žádost server zpracuje a každou knihu v databázi porovná s daným parametrem, který přidá do výsledné skupiny, toto by ale znamenalo že uživatel dostane více zástupců jedné knihy v poli výsledků, experimentálně jsem zjistil, že hashset nedokáže zpracovat velké objekty, a proto tedy tvořím hashset z id všech knih z výsledné množiny pomocí které ho poté vygeneruje seznam výsledků.

Práce s daty je značně jednoduchá díky knihovně Java LocalDate (LocalDate) vyvíjené vědci z Cern, která umožnuje jednoduše porovnávat všechna data pomocí metody isBefore(). Seznamy nevrácených knih jsou řazeny pomocí interface comparable (Overflow) které poté řadí list od nejvíce prošlé knihy.

5. Front end

Front end je klientskou částí aplikace, která zobrazuje uživateli data, a také mu umožnuje odesílat příkazy serveru. Pro frontend používám Angular (), což je prakticky plnohodnotný framework, ke svému běhu potřebuje NodeJS nebo NPM (node package manager) ale také spoustu dalších podpůrných programů jako JQuerry, Popper nebo Bootstrap (Bootstarp).

Angular objektivně není nejjednodušší framework k naučení. Nemá úplně jednoduchou strukturu a zejména typování v typescriptu mi přejde hodně nepřehledné, aplikace celkově celkem jednoduše sklouzne k trochu nepřehlednému kódu, na druhou stranu Angular má opravdu široké možnosti z hlediska toho, co se dá dělat s frameworkem. Základnám stavebním kamenem Angularu je komponenta, která se skládá z templatu, ts souboru a css souboru.

Struktura aplikace se dá rozdělit do tří částí, na základní app-component, další komponenty a http klienty.

Na straně klienta se také kontrolují data tak, aby byla správně vyplněná, například aby uživatel měl uživatelské jméno.

5.1. Http klient

Angular disponuje silným http klientem, který umožnuje posílat žádosti serveru, původně jsem měl v plánu použít klient Axios, ale funkcionalitou základní klient Angularu je dostačující, dokonce jsem se později dočetl že Axios byl vyvinut z klientu Angularu.

V projektu jsou http klienti uloženy ve složce services. Pro každý controller je zde samostatný http klient. Každý klient obsahuje konstantu se základní adresou serveru. Každá metoda poté přidává přesné určení metody, do které je žádost určena. Dále jsou přidány parametry cesty, například id cílového objektu. Pro žádosti post a put je také připojeno tělo žádosti, do které jsou vkládány jednotlivé prvky. K tělu žádosti je následně připojena hlavička s informací, že obsah je typu JSON.

5.2. Autentifikace na straně serveru

Autentifikace na straně klienta je poměrně jednoduchá, to je asi největší výhoda JWT (Bezkoder). Server odešlé žádost s heslem a uživatelským jménem uživatele. Pokud je

heslo správné v odpovědi přijdou informace uživatele jako id, role, a zejména token. Všechny jsou uloženy do lokální paměti prohlížeče (Local storage).

Interceptor (Bezkoder) poté následně při zpracování každé žádosti připojí do hlavičky token s informacemi uživatele které ověří server.

Role uživatele jsou posuzovány také podle informací uložených v lokáním úložišti, podle nich je zobrazován uživateli obsah, pokud se uživatel náhodou dostane na stránku, kde nemá být, nic se nestane, protože žádná data se nezobrazí kvůli chybícímu správnému tokenu.

Při odhlášení jsou uživatelská data vymazána z paměti prohlížeče.

5.3. Uživatelské nastavení

Při vytvoření serveru je defaultně vygenerován admin, který může tvořit knihy a prohlížet seznamy nevrácených knih, či jmenovat nové administrátory, má také jako jediný právo smazat uživatele, to lze ale pouze pokud má uživatel všechny knihy vrácené.

Uživatel si může změnit heslo zadáním starého, které je porovnáno na serveru, jestli se shoduje a je možné zadat nové v rámci možností validátorů.

5.4. Komponenty

Aplikace je rozdělaná na jednotlivé komponenty, mezi kterými se uživatel naviguje pomocí routingu. Komponenty mají implementovaný klient, ze kterého přebírají metody pro práci s databází. Většinou obsahují formulář na vložení dat. Nebo doplňují data do templatu. Seznamy knih mají vždy tlačítko pro zobrazení individuálního profilu knihy.

Protože jsem se snažil, aby aplikace byla maximálně dynamická je možné filtrovat knihy pomocí dropdown seznamů, které vždy načítají aktuální seznamy žánrů a autorů z databáze

5.5. Routing

Aplikace používá základní Angular routing (Angular Doc) pro navigaci mezi jednotlivými komponentami, cesty jsou uloženy v souboru app routing. Některé cesty mohou mít parametr, pro zobrazení stránky s konkrétním obsahem.

Navbar je použit pro navigaci po aplikaci, na navbaru jsou vždy zobrazeny jenom aplikace, ke kterým má uživatel přístup. To je zajištěno načtením aktuální role z navbaru.

6. Informace pro spuštění

Všechny informace pro spuštění jsou obsažené v readme souboru.

7. Závěr

Na závěr bych rád zhodnotil svůj projekt, myslím že se mi povedlo naplnit cíl projektu, a to sestavit plnohodnotnou aplikaci pro knihovnu, aplikace je plně funkční a myslím že po lehkém doladění zejména frontendu připravena pro použití v praxi.

Trochu špatně jsem odhadnul časovou náročnost projektu, kdy jsem si snažil hodně vyhrát s backendem, a na frontend mi zbylo relativně málo času.

Z hlediska naplnění mích osobních cílů se mi podařilo přiučit se hodně nového, jak z hlediska nových technologií, tak i práce s časem. Jsem rád že jsem se naučil používat nové technologie, které se dnes používají k práci ve firmách.

(Bezkoder, 2021)

8. Citace

Angular. Angular Documentation. *https://angular.io/dochttps://angular.io/docs.* [Online]

Angular documentation. *https://angular.io/dochttps://angular.io/docs.* [Online]

Bezkoder. Bezkoder Angular Auth. *https://bezkoder.com/angular-10-spring-boot-crud/.* [Online]

—. 2021. JWT Authentication. *https://bezkoder.com/spring-boot-jwt-authentication/.* [Online] 1. 21 2021.

Bootstarp. Boot Strap documentation. *https://getbootstrap.com/docs/4.1/content/reboot/.* [Online]

Documentation, Spring Boot. Spring Boot documentation. *https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.* [Online]

Dropdown, NG. NG multiple dropdown. *https://www.npmjs.com/package/ng-multiselect-dropdown.* [Online]

LocalDate. Java LocalDate. *https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html.* [Online]

MongoDB. MongoDB. *https://docs.mongodb.com/tools/.* [Online]

Overflow, Stack. Stack overflow. *https://stackoverflow.com/questions/29201103/how-to-compare-localdate-instances-java-8.* [Online]

Wikie. Wikipedia. *https://en.wikipedia.org/wiki/Representational_state_transfer.* [Online]