Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Vývoj webové aplikace pro správu a evidenci školení

Bakalářská práce

Autor: Eduard Spousta
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Jaroslav Langer

Hradec Králové Duben 2024

Prohlášení:

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

V Hradci Králové dne 23.4.2024 Eduard Spousta

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Jaroslavu Langerovi za metodické vedení a možnost průběžných konzultací k danému tématu. Děkuji za všechny cenné rady poskytnuté v průběhu tvorby vedoucí k vylepšení této práce.

Abstrakt

Kvalifikační práce se zabývá vývojem webové aplikace pro správu a evidenci školení zaměstnanců. Teoretická část práce se věnuje obecnému rámci a možnému využití technologií Spring Boot a React, na kterých je systém tvořen. Mimo to se zabývá architekturou REST a MVC. Praktická část je zaměřena na vývoj softwaru, především na tvorbu klíčových částí aplikace spolu se zdrojovým kódem. Dále jsou popsány konkrétní problémy a jejich možná řešení.

Výsledná webová aplikace a s ní spojené korektní fungování bylo ověřeno pomocí automatizovaných testů a lidské interakce. Software splnil zadaná kritéria a je připraven k budoucímu užití. V závěru práce jsou sepsána doporučení pro možná pokračování ve vývoji a rozšíření funkcionalit softwaru.

Klíčová slova: Spring Boot, React, webová aplikace, RESTfulAPI, MVC

Abstract

Title: Development of web application to manage training records

The bachelor's thesis deals with the development of a web application for managing and storing training records. The first part of this work is devoted to the theory and possible use of Spring Boot and React technologies. Those technologies create foundation for this software. In addition, it deals with REST and MVC architecture. The second part (practical) is focused on the development of such software. It displays the creation of the key parts of this software including the source code. It also describes the problems during development and their possible solution.

The results are associated with the correct functionality of the software. The system was verified with the help of pre-written automated tests and human interaction. The software has met the specific criteria, and it is ready for potential future use. At the end of the thesis recommendations are written to showcase possible future development of such software.

Key words: Spring Boot, React, web application, RESTfulAPI, MVC

Obsah

Seznam obrázků

Seznam ukázek kódu

Seznam zkratek

API ..... Application Programming Interface
CORS ..... Cross-Origin Resource Sharing
CRUD ..... Create Read Update Delete
DOM ..... Document Object Model
DI ..... Dependency Injection
DTO ..... Data Transfer Object
GUI ..... Graphical User Interface
IoC ..... Inversion of Control
JWT ..... JSON Web Token
MVC ..... Model-View-Controller
REST ..... REpresentional State Transfer
URL ..... Uniform Resource Locator

1 Úvod

Za posledních dvacet let se vývoj webových stránek posunul z jednoduchého html kódu s mírnou úpravou vzhledu k velmi komplexním systémům, na kterých mnohdy pracuje několik desítek developerů. Aby se práce s těmito webovými systémy co nejvíce usnadnila, byla vyvinuta řada klíčových softwarů, které často mezi sebou vzájemně spolupracují.

Teoretická část práce se zaměřuje na vývoj softwaru, který využívá některé z těchto moderních technologií. Konkrétně jde o technologie Spring Boot, React a databázový systém MySQL, na kterých bude aplikační systém postaven. Práce detailně popisuje a vysvětluje, jak dané technologie fungují a jak je lze v praxi uplatnit.

Praktická část je věnována tvorbě webového programu pro správu a evidenci školení. Výše zmíněné technologie jsou implementovány při jeho tvorbě. V níže uvedených kapitolách jsou analyzovány klíčové problémy, ke kterým se váže i jejich řešení. V práci jsou dále vyobrazeny klíčové části kódu, kompletní návrhy architektury, popsáno je rovněž testování aplikace. Implementace těchto částí povede k optimálnímu chodu systému.

Za předpokladu, že výsledný software bude splňovat zadaná kritéria, může mít tento systém dopad na podnikatele, kterým umožní snadný monitoring lidských zdrojů a jejich provedených školení. Přímo sníží administrativní náležitosti a byrokracii, která je spojená s těmito procesy, a nepřímo tak povede ke snížení ekonomických výdajů jednotlivých podnikatelských subjektů.

2 Cíl a metodika práce

Cílem této práce je představit řešení informačního systému, který nabídne uživatelům snadnou a efektivní správu evidence školení zaměstnanců. Takový software je primárně určen pro společnosti bez speciálních programů pro personalistiku, který nahradí lokální evidenci v tabulkových editorech. Bude splňovat požadavky zaměstnavatelů na evidování zaměstnanců, definovat jednotlivá školení a zaznamenávat jejich provedení. Navržený informační systém bude připomínat nutný termín pro včasné absolvování školení. Řešení bude dostupné online, což umožní rychlý a efektivní přístup k datům.

První část této práce nahlíží na teoretickou stránku aplikace. Jsou v ní představeny a analýze podrobeny standardy programování společně s technologiemi, na kterých bude práce založena. V praktické části jsou aplikovány poznatky z teoretické části, které jsou využívány pro tvorbou samotného systému.

3 Teoretická východiska

Následující podkapitoly jsou věnovány teoretickým poznatkům nabytým při práci se zdroji k tématu této bakalářské práce.

3.1 Spring Boot

Spring Boot je open-source framework, který umožňuje vytváření aplikací v jazyce Java. Spring Boot je založen na technologii Spring, jehož flexibilita a jednoduchost umožňuje rychlý vývoj aplikací bez zdlouhavých konfigurací. Často je využíván také pro tvorbu mikro služeb, což je způsob vývoje, kdy jsou malé části aplikace vytvářeny nezávisle a na závěr se integrují do jednoho celku, ve kterém komunikují skrze služeb API. [1]

Tato práce využívá technologii Spring Boot pro tvorbu webové aplikace (serverové části). V následujících podkapitolách budou rozebrány všechny potřebné technologie a praktiky vztahující se k tvorbě aplikace.

3.1.1 Inverze řízení a správa závislostí

Inverze řízení neboli Inversion of Control (IoC) je koncept, který je využíván pro přenesení kontroly nad danou entitou jiné entitě nebo frameworku. Znamená to, že určité činnosti třídy jsou svěřeny k provedení třídě jiné. [2]

Injektáž závislostí, anglicky Dependency Injection (DI), je implementace IoC. O funkci se stará samotný framework (Spring Boot), díky němuž je umožněno definovat objekty mimo samotnou třídu objektu. [3] To znamená, že je možné pracovat s metodami tříd, aniž by byla vytvořena její instance *(např: X x = new X(...)).* Ve Spring Bootu se implementuje s pomocí mnoha anotací. Zde je několik příkladů:

- @Bean: Píše se uvnitř konfiguračních tříd. Třída je označena anotací @Configuration, která definuje základní funkcionality v systému. Tento Bean je spravován právě IoC kontejnerem. Konkrétním příkladem je implementace hashovací funkce (k dispozici v ukázce kódu č. 1). [4]
- @Autowired: Zajišťuje automatické vložení závislosti na jinou třídu. Díky tomu se nemusí explicitně vytvářet instance samotné třídy. Po inicializaci jsou následně dostupné veškeré funkce třídy. [3]

- Další anotace využívané pro tvorbu MVC architektury: @Service, @Controller, @RestController aj. [3]

```
@Bean
public PasswordEncoder passwordEncoder() {
   return new BCryptPasswordEncoder();
}
```

Ukázka kódu 1: @Bean pro hashovací funkci, převzato [5]

3.1.2 MVC architektura

Využití MVC architektury je dnes standardním přístupem při tvorbě webových aplikací. MVC je anglická zkratka pro Model-View-Controller. Autor Hartinger D. v článku uvádí, že cílem MVC architektury je rozdělovat aplikaci do tří vývojově oddělených částí, kde každá tato část má jasně daný účel a logiku.

- Controller: Česky kontrolér je zodpovědný za řešení požadavků zaslaných klientem. Komunikuje s ostatními komponentami a deleguje na ně práci.
- Model: Interaguje s databází a provádí veškeré operace s daty.
- View: Česky pohled, zajišťuje zobrazování dat pomocí definovaných šablon.

Lze tedy dojít k závěru, že model a view spolu vzájemně nikdy nekomunikují a vše je delegováno skrze controller. Ten je využíván jako tzv. middle man neboli prostředník pro tok dat. [6]

Pro vyobrazení toho, jak architektura vypadá v praxi, je možné představit následující scénář pro načtení detailu na webu [6]:

1) Klient zasílá http request pro načtení určité stránky ze serveru.
2) Server odesílá request do určitého Controlleru.
3) Controller zasílá request na data.
4) Model načítá data z databáze.
5) Model zasílá data zpět do Controlleru.
6) Controller se dotazuje na danou šablonu z Views.
7) Views nahraje data do šablony.
8) Controller zasílá odpověď (šablonu s daty) zpět uživateli.

3.1.3 RESTful API

RESTful API je označována jako architektura REST, což je zkratka pro Representational State Transfer. Jedná se o architekturu, která zajišťuje komunikaci mezi klientem a serverem. V praxi proces probíhá následovně [7]:

1) Požadavek klienta: Klient zasílá požadavek (request) na server skrze uživatelské prostředí. Developeři tyto requesty typicky obsluhují pomocí aplikace Postman. Tyto požadavky mohou mít různé metody a parametry (viz. níže).
2) Validace: Server zjistí, zda má klient právo odeslat tento požadavek. Jde o proces autentizace, kdy server provede ověření správnosti klienta. V praxi to znamená, že klient musí být přihlášen do systému. Následně je provedena autorizace, při které je zjištěno, zda daný uživatel má oprávnění na provedení dané operace.
3) Zpracování požadavku: Server provede požadavek, jaký mu klient zaslal.
4) Zaslání odpovědi: Klientovi je zaslána odpověď. Dostává informaci o stavové hlášce (viz. kapitola 3.1.4.2 Statusové http kódy) a případně také data, která si vyžádal.

3.1.4 HTTP metody

Http metody jsou akce odesílané na server za pomoci http protokolu. Udávají typ žádosti, dle které bude provedena konkrétní operace. [8]

3.1.4.1 Typy metod

Aby se dal požadavek co nejvíce optimalizovat, existuje spousta metod, které mají své specifické využití. Autor knihy HTTP: The Definitive Guide (Definitive Guides) uvádí následující příklady [8]:

- GET: požadavek na data,
- POST: uložení dat,
- PUT: aktualizace záznamů na základě odeslaných dat,
- DELETE: odstranění záznamu,
- HEAD: požadavek na metadat souboru.

Existuje mnoho dalších metod, ty jsou ale využívány spíše v ojedinělých případech, např. TRACE a PUBLISH. Dle oficiální dokumentace se ve Spring Bootu tyto metody jsou využívány uvnitř controlleru aplikace jako anotace. Jsou napsané

tzv. CammelCase, začínají zavináčem a končí vždy na „Mapping“. Tyto metody by tedy vypadaly následovně: @GetMapping, @PostMapping, @PatchMapping apod. [9]

3.1.4.2 Statusové http kódy

Statusové kódy jsou http odpovědi serveru, který zasílá informaci (číslo) o stavu požadavku. Navrácené číselné hodnoty reprezentují odpověď o tom, jak byl požadavek zpracován. Dělí se dle kategorií vždy od X00 po X99, kde X reprezentuje číslo v rozmezí 1 až 5. [8] V následujících podkapitolách jsou tyto kódy rozřazeny dle typu odpovědi.

3.1.4.2.1 Informační stavy (100-199)

Obecně se jedná o stavy, které jsou čistě informační. V praxi nejsou příliš běžné, proto se s nimi často v aplikacích vůbec nesetkáváme. Jedním z možných využití je odeslání většího množství dat (na více částí) [8].

- 100 CONTINUE: Jedná se o odpověď serveru, kdy uživatel zašle část požadavku, server ho obdrží a čeká na klientovo pokračování. [8]

3.1.4.2.2 Úspěch (200-299)

Jedná se o odpověď indikující úspěšné provedení požadavku klienta [8]:

- 200 OK: Stav, kdy server zpracoval požadavek bez komplikace.
- 201 Created: Dochází k vytvoření serverového záznamu.
- 202 Accepted: Server přijal požadavek, ale ještě nebyl zpracován. Znamená to, že request byl validní, ale server ho neprovedl. Může být například ve frontě.

3.1.4.2.3 Přesměrování (300-399)

- 300 Multiple Choices: Klientovi je zaslán v případě, že má na výběr z několika možností. Například se objevuje v případech, kdy jsou k dispozici vícejazyčné stránky. Dostáváme tak na výběr, zda chceme zobrazit českou verzi nebo anglickou. [8]
- 302 Found: Zdroj byl dočasně přesunut na jinou adresu. Znamená to, že když klient zašle request, server zjistí, že obsah byl přesunut na jinou adresu, zašle tedy odpověď 302 Found společně s URL, prohlížeč provede přesměrování na novou URL odeslanou ze serveru a načte ji uživateli. Využívá se například při testování nebo aktualizaci webových stránek. [10]

3.1.4.2.4 Klientské chyby (400-499)

Všechny odpovědi v rozmezí 400-499 jsou chyby, kdy byl zaslán požadavek, ale server ho nedokázal zpracovat. [8]

- 400 Bad Request: Jedná se o obecnější chybu, která sděluje, že server nemůže zpracovat požadavek klienta. Jak je v dokumentaci pro Mozilla browser uvedeno, může jít např. o chybu v syntaxi nebo o špatné směrování požadavku. [11]
- 401 Unauthorized: Uživatel nemá přístup k tomuto end-pointu (nebyl autentizován). Stránka není dostupná pro nepřihlášené uživatele. [11]
- 403 Forbidden: Signalizuje, že nemůže provést požadavek klienta. Klient je autentizován, ale nemá oprávnění na provedení požadavku (autorizace). [8]
- 404 Not Found: Znázorňuje, že URL požadavku neexistuje. Pro server není možné reagovat a uživateli. [11] Uživateli je zobrazena typická chyba: 404 Page Not Found.
- 429 Too Many Requests: Indikuje, že klient zaslal příliš mnoho požadavků za stanovený čas [11].

3.1.4.2.5 Chyby serveru (500-599)

Chybám serveru jsou vymezeny stavové kódy v rozmezí od 500 do 599. Tyto chybové hlášky jsou spíše ojedinělé, požadavky klienta jsou v těchto případech validní, ale samotný server má problém s jejich zpracováním. [8] Z literatury HTTP: The Definitive Guide (Definitive Guides) byly vybrány následující odpovědi:

- 500 Internal Server Error: Obecná chyba.
- 503 Service Unavailable: Server není aktuálně schopen požadavek zpracovat. Může jít například o stav, kdy je server přetížený. [8]

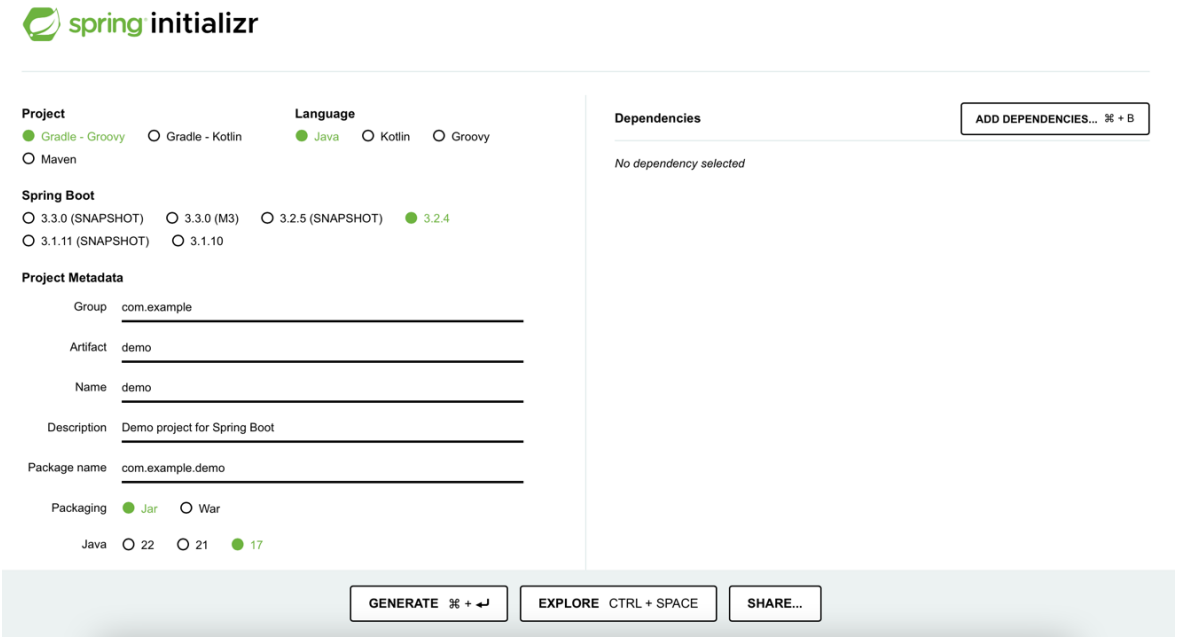
3.1.5 Práce s frameworkem

Jak již bylo zmíněno, se Spring Bootem je možné pracovat díky anotacím uvnitř tříd. Ty jsou obohaceny o Java kód a tím tak utváří aplikační celek. Vizualizaci a podstatu práce s frameworkem jsou věnovány následující podkapitoly.

3.1.5.1 Tvorba projektu

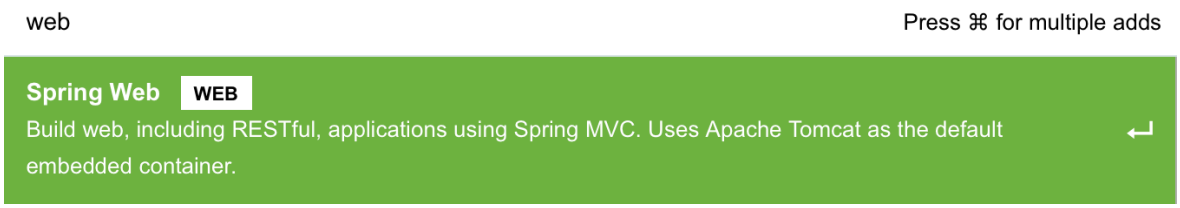
Pro snazší a časově úspornější inicializaci nových projektů byl vytvořen generátor zvaný spring initializr. Ten je dostupný na webu https://start.spring.io/.

Využití tohoto generátoru je doporučeno v oficiálním návodu v práci se Springem, je k dispozici na adrese https://spring.io/quickstart. [12]



Obrázek 1: Nástroj pro inicializaci projektu, vlastní snímek obrazovky z webu https://start.spring.io [13]

Z obrázku č. 1 je patrné, že díky generátoru je možné vytvořit projekt založený na technologii Gradle nebo Maven a také vybrat programovací jazyk. V následující sekci lze volit z několika verzí Spring Bootu. Zvolená verze se dá kdykoliv a snadno změnit v konfiguraci pox.xml projektu. Je však lepší vybrat stabilní verzi, která neobsahuje v názvu (SNAPSHOT). Následně jsou zvolena metadata projektu, jako je název, pojmenování balíčků nebo popis. Vybíráme, v jakém formátu si projekt stáhneme (Jar nebo War) a která verze Javy projektu bude spuštěna (17 nebo 21). Tato práce využívá Maven a Javu 17, verze Spring Bootu je 3.0.12. Velmi důležitým krokem je přidání závislostí (dependecies). Tyto závislosti se dají vyhledat přímo v prohlížeči a není nutné navštěvovat stránku https://mvnrepository.com, ze které by se v opačném případě tyto závislosti implementovaly. Veškerá tato rozšíření, která se v práci využívají, jsou detailně popsána v následující kapitole.

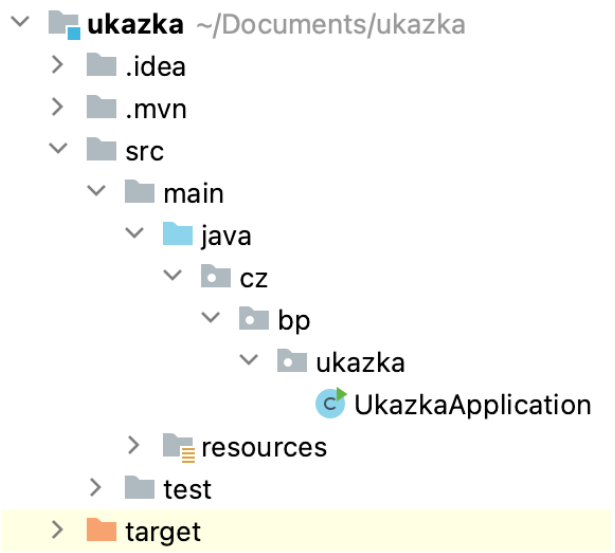


Obrázek 2: Vložení závislosti, vlastní snímek obrazovky z webu https://start.spring.io [13]

Po zadaní všech parametrů se projekt stáhne do PC a je dále na developerovi, jak se souborem naloží.

3.1.5.2 Tvorba end-pointu

Po využití technologie pro generaci projektu se vytvořila struktura vyobrazená na obrázku níže, která je dedikována čistě pro kód, zdroje a testy. Je zde vytvořen soubor pox.xml, ve kterém se nachází všechny přidané závislosti. Nyní je projekt připraven k používání a může se začít psát samotný kód.



Obrázek 3: Vygenerovaná struktura projektu, vlastní zpracování v Intellij IDEA

Pokud by se v aktuální situaci projekt spustil, fungoval by, ale nebylo by co zobrazit, jelikož dovnitř projektu nebylo doposud nic přidáno. Je proto nezbytné zaměřit se na tvorbu zdrojového kódu, který povede k určenému výsledku. Takováto aplikace musí být psána s výše zmíněnými standardy jako je MVC architektura.

Dle autora knihy Learning Spring Boot 3 a jeho postupů jsou prvně tvořeny koncové body pro API [14]. Z tohoto důvodu je nutné projekt obohatit o kontrolér, ve

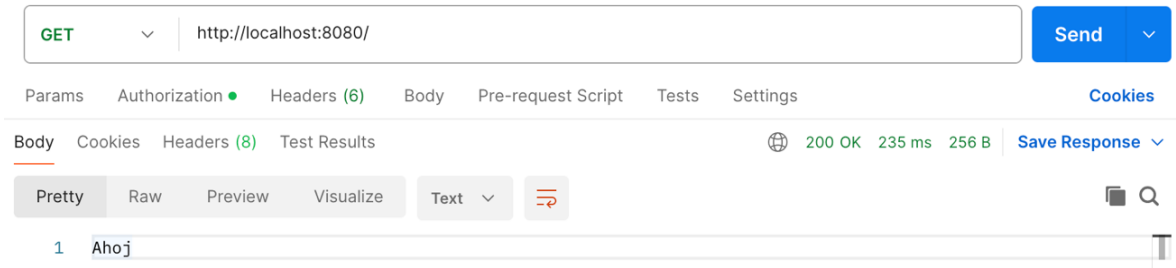
kterém bude dedikovaný kód pro řešení koncových bodů aplikace. Jak autor textu dále uvádí, ve Spring Bootu se vyskytují 2 typy controllerů. Existuje defaultní @Controller a @RestController. Obyčejný controller se využívá např. s technologií Thymeleaf, která umožňuje zobrazovat šablony uvnitř Spring projektu. Oproti tomu @RestController je určen pro tvorbu architektury Rest a je „serializován“ pomocí Jacksonu. V praxi by byl zvolen následující postup [14]:

1) vytvoření nové .java třída, s názvem UvitaciController,
2) přidání anotace @RestController,
3) přidání anotace pro typ metody (GET - @GetMapping),
4) provedení akce (vykreslení šablon, zavolání metod apod.).

```
@RestController
public class UvitaciController {
   @GetMapping("/")
   public String pozdrav() {
      return "Ahoj";
   }
}
```

Ukázka kódu 2: Tvorba testovacího controlleru, převzato a upraveno [14]

Po opětovném spuštění projektu je k dispozici první end-point na výše zmíněné adrese (…/). Pro zobrazení výsledku stačí zadat URL adresu do kteréhokoliv browseru nebo případně využít nástroj Postman, jak je níže vizualizováno. Výsledkem je zobrazení textu Ahoj.



Obrázek 4: Zaslaná odpověď, vlastní zpracování v Postman

3.1.5.3 Práce s databází

Pro možnost práce s databází je zcela nezbytné do projektu vložit další open-source frameworky, které umožní s databázemi pracovat. Pro tuto kvalifikační práci byla zvolena technologie MySQL.

Jedná se o relační databázi, což je ideální, neboť bude nezbytné implementovat vazby s řádně danou strukturou dat. [15]

Aby se dalo s těmito technologiemi pracovat, je třeba provést následující akce, na které autor knihy Learning Spring Boot 3 odkazuje. Implementace JPA, přidání konektoru pro využívanou databázi (MySQL) a přidání Lomboku (viz kapitola 3.1.6.4) jsou provedeny skrze konfigurační soubor pox.xml. Do tohoto souboru jsou také přidány tyto závislosti. [14] Je možné zvolit konkrétní vývojovou verzi v závislosti na požadavcích.

Následně je nutné vytvořit konfigurační třídu, která definuje základní nastavení pro databázi. Tak je možné učinit dvěma způsoby. Prvním z nich je třída .properties, která konfiguruje soubor řádek po řádku. Druhou variantou je využít soubor .yaml, který je rychlejší pro psaní konfigurace a také více přehledný. [14] To lze porovnat na následujících ukázkách konfigurujících databázi.

```
spring.jpa.show-sql=true
spring.jpa.open-in-view=true
spring.jpa.hibernate.ddl-auto=update
```

Ukázka kódu 3: Náhled konfigurace v .properties, převzato a upraveno [16]

```
spring:
    jpa:
        show-sql: true
        open-in-view: true
        hibernate:
            ddl-auto: update
```

Ukázka kódu 4: Náhled konfigurace v .yaml, převzato a upraveno [16]

Při programování takového systému lze využít technologii XAMPP (více v 3.1.7.2), která umožňuje server hostovat lokálně. Pro kompletní konfiguraci serveru

je nezbytné přidat další konfigurace (k ukázce 3 a 4), jako je adresa na databázový server společně s přihlašovacími údaji apod.

3.1.5.4 Tvorba tabulky a repozitáře

V případě úspěšné implementace databáze lze pokračovat ve tvorbě aplikace. Aby bylo možné uchovat uživatelská data, je nezbytné definovat strukturu databázového serveru, na který se data budou ukládat.

Kniha Learning Spring Boot 3 věnující se Spring Bootu znázorňuje, jak v tomto případě postupovat. Udává nutnost označit tabulku anotací @Entity, která signalizuje Springu, že se jedná o tabulku, jež bude automaticky generována při spuštění systému. Odkazuje také na nutnost použít @Id pro označení primárního klíče spolu se strategií na auto inkrementování jeho hodnot. [14]

Z těchto poznatků lze vytvořit vlastní třídu, která bude tvořit strukturu paralelně vyvíjené ukázkové aplikace, jež bude uchovávat a zobrazovat pozdravy, které uživatel vložil do databáze. Mimo to dále implementuje technologii Lombok, díky které se nemusí psát repetitivní kód, neboť je automaticky vygenerován při spuštění aplikace (více v podkapitole věnované Lomboku). Tato struktura by mohla vypadat následujícím způsobem (Ukázka kódu č. 5).

```
@Entity
@Table(name = "seznam_pozdravu")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class PozdravyEntity {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @Column
   private String pozdrav;
}
```

Ukázka kódu 5: Vytvoření tabulky, převzato a upraveno [14]

Aby nebylo nutné psát vlastní query na získávání dat z databáze, má v sobě framework JPA zabudováno další využití. Jde o možnost rozšíření rozhraní (interface), kde se přidá generická entita a typ primárního klíče. Tento repozitář má v sobě implementované CRUD operace. [14] Konkrétně se jedná o metody jako jsou .save(), .findById atd. Návrh takového repozitáře by vypadal následovně.

```
public interface PozdravyRepository extends JpaRepository<PozdravyEntity, Long> {
}
```

Ukázka kódu 6: Vytvoření repozitáře, převzato a upraveno [14]

3.1.5.5 Tvorba a implementace služeb

Z hlediska dodržení zásad architektury MVC je zapotřebí do takového projektu zakomponovat model. Ten je obecně označován jako DTO (Data Transfer Object). Z jeho názvu vyplývá i jeho funkce. Jde o objekt, který má jasně definovanou strukturu dat a je využit pro přesuny těchto dat ze serveru ke klientovi a směrem opačným v případě zaslaní dat od uživatele. [6]

V případě ponechání stejného příkladu jako doposud by tento DTO model vypadal následovně.

```
@Data
@AllArgsConstructor
public class PozdravDTO {
   private Long id;
   private String pozdrav;
}
```

Ukázka kódu 7: Model definující strukturu dat, vlastní zpracování

Dle autora článku Vishamber Lal implementace služeb využívá právě zmíněné DTO, které se aplikuje pro přenos dat mezi kontrolorem a repozitářem. To vede k přehlednějšímu kódu a lepší logice aplikace. Mimo jiné lze DTO využít i k optimalizačním účelům, kdy jsou v dané tabulce uložena i data, která uživatel aktuálně nepotřebuje. Bez použití tohoto modelu by se přenášela veškerá data, což by při náročných operacích prodlužovalo odezvu. [17]

Tyto DTO se využívají uvnitř třídy s anotací @Service, ve které jsou napsány metody dané služby [17]. Tyto služby se následně implementují do kontroléru za pomoci již zmíněné DI s anotací @Autowired. Při zachování stále stejného příkladu by kód mohl vypadat následovně.

```
@Autowired
private PozdravyRepository pozdravyRepository;
public PozdravDTO ulozeniPozdravu(PozdravDTO pozdravDTO){
    PozdravyEntity pozdrav = new PozdravyEntity();
    pozdrav.setPozdrav(pozdravDTO.getPozdrav());
    PozdravyEntity ulozenyPozdrav = pozdravyRepository.save(pozdrav);
    PozdravDTO navratovyDTO = new PozdravDTO(ulozenyPozdrav.getId(),
                                        ulozenyPozdrav.getPozdrav());
    return navratovyDTO;
}
```

Ukázka kódu 8: Metoda uvnitř služby, vlastní zpracování

Na totožném principu by byly tvořeny i další metody, do budoucna nezbytné pro chod takovéto aplikace. Jsou to metody jako výpis všech parametrů nebo mazání. Tyto nově vytvořené metody jsou volány z kontrolérů uvnitř API end-pointů. V případě této ukázky by bylo výstižné zvolit URL adresu odkazující na pozdravy (…/pozdravy). Tímto by byly připraveny API end-pointy, odkud by si je uživatelská strana zavolala v případě potřeby. Samotná implementace vybraných kontrolérů pak bude vypadat takto.

```
@GetMapping("/pozdravy")
public List<PozdravDTO> vypisPozdravy(){
  return pozdravyService.vypisPozdravu();
}
@PostMapping("/pozdravy")
public PozdravDTO ulozPozdrav(@RequestBody PozdravDTO pozdravDTO){
  return pozdravyService.ulozeniPozdravu(pozdravDTO);
}
```

Ukázka kódu 9: Koncové body na adrese /pozdravy, vlastní zpracování

3.1.5.6 Definice CORS

CORS je zkratka pro Cross-Origin Resource Sharing, což ve volném překladu znamená sdílení zdrojů mezi různými doménami. Jde o mechanismus, který umožňuje definovat jednotlivé domény, schémata nebo porty, které mají přístup do aplikace. Dále je například umožněno specifikovat povolené metody či to, co budou dané požadavky moci obsahovat. Ve Spring Bootu se k implementaci CORS dá přistoupit dvěma následujícími způsoby [16]:

- CORS pro API End-point: Využívá se v případě, že jde o aplikaci s malým množstvím API end-pointů nebo v případě rozdílného požadavku na koncový bod od globálního nastavení. Autor v knize uvádí, že je možno nastavit toto povolení komunikace s pomocí anotace *@CrossOrigin(origins = "")*, kde je do uvozovek uvedena daná URL adresa, z jaké budou požadavky odeslány. [16] Pro příklad: V případě vývoje aplikace s Reactem na lokálním zařízení bude tato adresa vypadat takto: http://localhost:3000.
- Globální nastavení CORS: V naprosté většině případů je využívána tato možnost. Autoři knihy D. B. Duldulao a S. R. Villafranca uvádí, že za pomoci nové konfigurační třídy lze globálně nastavit CORS pro veškeré end pointy [16].

Ačkoliv pro stálý příklad s pozdravy by globální nastavení nebylo nezbytné, jedná se o běžný přístup. Toto nastavení by mohlo být vytvořeno následujícím způsobem.

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
        .allowedOrigins("http://localhost:3000")
        .allowedMethods("GET", "POST", "PUT", "DELETE");
  }
}
```

Ukázka kódu 10: Nastavení CORS, vlastní zpracování

3.1.6 Rozšíření pro Spring Boot

Existuje několik užitečných rozšíření, které se Spring Bootem mohou interagovat. Některé frameworky čistě zjednodušují práci developerovi, aby nemusel psát velké množství kódu. Jiné naopak nabízí rozšíření funkcionalit pro daný systém a vždy záleží na vývojáři, co od systému požaduje a s jakými technologiemi si zvolí pracovat. Níže uvedené technologie jsou v praxi běžně využívané a budou součástí systému tvořeného v praktické části.

3.1.6.1 Spring Boot Starter Web

Starter Web je elementární rozšíření, bez kterého by žádný systém nemohl fungovat. Obsahuje základní anotace pro tvorbu architektury REST a MVC [18]. Tyto anotace jsou označeny symbolem @. Existuje mnoho anotací a vždy záleží na tom, co je od systému požadováno. Z literatury Learning Spring Boot 3 je vybráno pár pomyslných příkladů [14]:

- DI: anotace @Autowired pro vložení závislostí,
- REST a MVC: anotace vztahující se k architektuře rest, například @RestControler, @RequestMapping, @GetMaping, @RequestParam atd.,
- další anotace zajišťující chod serveru: @SpringBootApplication (Main třída), @Configuration, @Bean apod.

3.1.6.2 Spring Data JPA

Toto rozšíření je využíváno k práci s relačními databázemi ve Spring Bootu. Redukuje psaní repetitivního kódu a má předdefinované rozhraní [19]. Co konkrétně framework zajišťuje, je pospáno v následujících podkapitolách.

3.1.6.2.1 Tabulky

Databázové tabulky jsou tvořeny skrze Java třídy pomocí speciálních anotací. Automaticky se generují (pomocí generace příkazu query) při spuštění projektu, a to včetně jejich relací, primárních i cizích klíčů apod.

Dle autora literatury Learning Spring Boot 3 pro označení entity (tabulky) je třeba doplnit anotaci @Entity [14]. Mimo to je nezbytné v této entitě označit primární klíč anotací @Id, který je celočíselného datového typu. Tento primární klíč je zapotřebí využít společně s anotací @GeneratedValue, která zajistí strategii pro tvorbu identifikátoru (auto inkrementace). [20]

Pomocí jiných anotací lze provést další úpravy, tj. nastavit jméno tabulky @Name(“nazev_tabulky“), nebo například definovat samotné vazby mezi nimi. Tyto vazby jsou využívány pro označení relace mezi tabulkami v závislosti na multiplicitě. Jde o anotace @OneToOne (1:1), @OneToMany (1:N), @ManyToOne (N:1) a @ManyToMany (N:M), které se vždy musí umístit do obou tabulek. [21]

3.1.6.2.2 Repozitáře

Tento framework dále umožnuje vytvořit rozhraní, které slouží k provádění operací nad samotnou databází. Jak autor literatury G. L. Turnquist zmiňuje, toto rozšíření umožňuje využívat query, aniž by to bylo napsané. K tomu jsou využity právě tyto repozitáře. V Javě se vytvoří nový Interface, který se rozšíří o JpaRepository<>, kde bude první generický typ odkazovat na entitu a druhý na datový typ dané entity (viz ukázka kódu č. 6). Při vytvoření instance této třídy jsou k dispozici metody k provedení operací CRUD spolu s dalšími, jako findById(), findAll() apod. [14]

3.1.6.3 MySQL Driver

Zajišťuje komunikaci a spojení s databázovým serverem. Dle oficiální dokumentace je nutné uvnitř konfiguračního souboru definovat přístup k databázi, tzn. adresu a přihlašovací údaje. Mimo jiné se také definuje strategie pro aktualizování databáze, což je při vývoji velmi užitečné. Příkladem je přístup create-drop, ve kterém se při kompilaci vygeneruje databáze a při jejím vypnutí se zase vymaže. Existují další strategie a to none, update, create, kdy každá z nich plní jinou funkci. Přidáním scriptu spring:jpa:show-sql:true do konfiguračního souboru (.yaml nebo .properties) je možné zobrazit všechny prováděné query do konzole. [22]

3.1.6.4 Lombok

Lombok je knihovna pro jazyk Java, která zjednodušuje vytváření a případnou modifikaci tříd. Dle principu zapouzdření jsou vytvářeny privátní atributy uvnitř tříd, ke kterým je přistupováno díky getterům a setterům. [23]

Dle dokumentace tento nástroj pomáhá eliminovat tzv. boilerpalte kód. To znamená, že nemusí být psán repetitivní kód, jako již zmíněné gettery a settery, ale také nemusí být ručně psány různé typy konstruktorů apod. S pomocí Lomboku tak v mnoha případech díky napsání jedné anotace @Data lze nahradit většinu kódu celé třídy. Při kompilaci se automaticky vygenerují všechny gettery a settery, dále

konstruktor obsahující finální atributy. Mimoto jsou vytvořeny i metody jako toString(), hashCpde() a equals(Object). [23]

Pro případy, kdy není třeba vytvářet všechny tyto metody, existuje více specializovaný přístup, díky kterému je možné tvořit jen některé z nich. Jde například zvlášť o tvorbu getterů @Getter a setterů @Setter, konstruktorů @NoArgsConstructor @AllArgsConstructor apod. [23]

3.1.6.5 Spring Security

Spring Security je framework, který poskytuje zabezpečení aplikace. Dle oficiální dokumentace je toto rozšíření standardem pro zabezpečení. Umožňuje definovat nastavení dle vlastních požadavků. [24] To je prováděno v konfigurační třídě uvnitř implementované metody na SecurityFilterChain, do které vstupuje objekt typu HttpSecurity, na kterém se nastavení provádí. Zde lze jasně stanovit hranice systému pro přihlášené a nepřihlášené uživatele spolu s povolenými metodami atd., jak autor uvádí. [14] Umožňuje také definovat nastavení pro akceptaci JWT tokenu, které bude využito v praktické části aplikace k autentizaci a autorizaci.

- Autentizace: Proces ověření identity uživatele na základě identifikačních údajů (např. hesla) [25].
- Autorizace: Proces ověření práv uživatele. Zjištění, zda má možnost provést danou operaci (např. přístup na end-point). [25]

Tento proces (autorizace a autentizace) má několik způsobů implementace, a to nejen ve Spring Boot Security. Existují tyto přístupy:

- Session-based: Po přihlášení klienta server generuje novou session (relaci). Ta je přiřazena konkrétnímu uživateli. Server zašle ID této relace klientovi, kterému je vložena do Cookies. Klient při každém dotazu zasílá toto sessionID, jež ověřuje, že se jedná o daného uživatele. [26]
- Token based (JWT): Po přihlášení klienta server generuje nový JWT token, jež obsahuje podpis, v případě potřeby také data, a je zašifrován. U klienta je token uložen a je následně připojen ke každému požadavku uživatele. [26]

3.1.7 Nástroje pro vývoj

Při vývoji aplikace je vhodné využití dalších nástrojů pro zrychlení a usnadnění práce. Dvěma nástrojům, Postman a XAMPP, se věnují následující podkapitoly.

3.1.7.1 Postman

Dle oficiálních stránek je Postman softwarová desktopová aplikace vyvinutá společností Postman, Inc. K dnešnímu dni ji využívá více než 30 milionů developerů po celém světě. Je dostupná pro všechny tři nejpoužívanější operační systémy. Jedná se tedy o nástroj určený vývojářům API. Ty si díky němu mohou vytvářet testy pro end-pointy aplikace. [27]

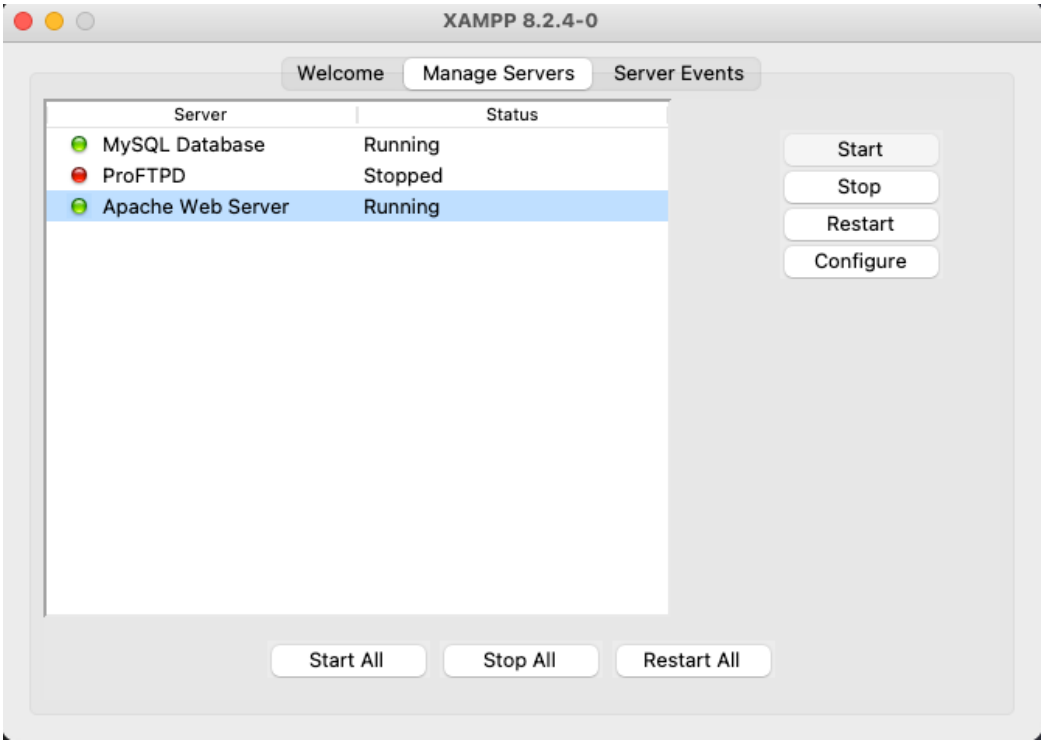
Jinak řečeno, aplikace Postman umožňuje zasílání http požadavků (GET, POST atd.) na různé URL adresy. Do těchto požadavků lze vkládat různé proměnné. Mezi ty nejvíce používané se řadí následující:

- Body (tělo): odeslání dat ve formátu JSON, XML apod.,
- Headers (hlavička): zaslání klíče nebo tokenu (autorizace),
- Params (Query parametry): Jsou připojeny na konec URL adresy požadavku. Data jsou uvedena za symbolem ? od základní URL adresy. Tato data udávají, co si přeje uživatel s výslednými daty vykonat (filtrování, seřazení a podobně). [28]

3.1.7.2 XAMPP

Nástroj XAMPP je softwarový nástroj, ve kterém jsou zabudované technologie potřebné pro vývoj webového systému. Jedná se o open-source program, který obsahuje mnoho technologií, jako například Apache web server nebo MySQL databázi. Jedná se o nástroj sloužící developerovi pro vývoj, aby nemusel využívat externí server a vše mohlo ve vývojové fázi probíhat na osobním počítači. [29] V případě této práce budou ze systému využity následující technologie:

- Apache Web Server: Je open-source webový server, pro hostování aplikace [29].
- MySQL Databse Server: Umožní hostování databáze na lokálním počítači. Přehled databáze a všech údajů v ní je dostupný na adrese *http://localhost/phpmyadmin/index.php*. Jedná se o nástroj phpMyAdmin, pomocí kterého je možno provádět další operace (scripty) nad samotnou databází. [29]



Obrázek 5: GUI nástroje XAMPP, vlastní snímek obrazovky z aplikace XAMPP

3.2 React

React je označován také jako ReactJS. Jak je na oficiálních stránkách uvedeno, React je JavaScript knihovna, která je vývojáři využívána pro tvorbu uživatelského rozhraní. Byla vytvořena společností Meta (dříve Facebook) a je open-source stejně jako Spring Boot. [30]

Z pohledu MVC architektury je účelem Reactu tvorba šablon (View). Využívá se tedy čistě pro komunikaci s klientem. Odesílá požadavky na server a vykresluje navrácená data. Obsahuje pouze nezbytnou logiku pro správné fungování uživatelského rozhraní. [6]

3.2.1 Front-end a Back-end

I když technologie Spring Bootu dovoluje vytvořit klientské prostředí, např. za využití nástroje Thymeleaf, není to nejoptimálnější řešení.

Dělení vývoje webové aplikace do dvou vývojových částí (back-end a front-end) se běžně v praxi provádí. Znamená to, že se vždy zahájí jednou částí a následně na ni navazuje druhá. Ty spolu komunikují skrze koncové body. [31]

Jak autor článku Vikram Joshi uvádí, existuje několik následujících příkladů proč vývoj dělit [31].

- Správa a údržba: Dělením do dvou vývojových částí vzniknou jasně dané hranice rozlišující to, která část spravuje konkrétní funkce. Tato fakta se využijí při správě, vývoji i „debuggingu“, jelikož lze díky nim snáze lokalizovat původ chyby.
- Výkon: V případě, kdy se provádějí náročné operace s daty (např. výpočty), jsou tyto operace závislé na rychlosti zařízení uživatele. S využitím serveru jsou výpočetní algoritmy delegovány na server, který pouze vrací výsledná data k zobrazení.
- Technologie: Vzhledem k tomu, že neustále vycházejí nové verze technologií, nastane doba, kdy bude nezbytné využívanou verzi „upgradovat“ (např. z důvodu nekompatibility nebo zranitelnosti). To by mohlo ohrozit funkčnost celé aplikace (ne pouze jednu stranu). Mimoto umožňuje snadnější změnu technologií, jak na straně klienta, tak na straně serveru, v případě potřeby.

3.2.2 Klíčové koncepty a výhody Reactu

Pro efektivní využívání Reactu je zapotřebí pochopit, jaké výhody React může nabídnout. Jde o několik klíčových konceptů, které poskytuje a developer je tak může implementovat do každého projektu.

- Virtuální DOM: DOM je zkratka pro Document Object Model, který je vytvořen ze samotného HTML kódu stránky. Díky virtuálnímu DOMu dochází k optimalizaci aktualizací webového rozhraní, neboť minimalizuje množství změn, které je potřebné provést v reálném DOMu. Při provádění aktualizace, např. zobrazování nového záznamu z databáze do uživatelského okna, se využije právě virtuální DOM, který se porovná s DOMem novým a provede jen nezbytné změny, aby se znovu nemusela načítat celá stránka. Tím je zajištěna větší efektivita a plynulost aplikace, což vede k pozitivnímu uživatelskému zážitku. [32]
- Deklarativní přístup: Deklarativní přístup definuje, čeho je potřeba dosáhnout bez nutnosti specifikace veškerých kroků, tedy jak výsledku docílit. V Reactu je k tomuto programovacímu přístupu využita JSX syntaxe. To v podstatě znamená, že React umožňuje kombinovat HTML elementy s java-scriptovým kódem. [14]

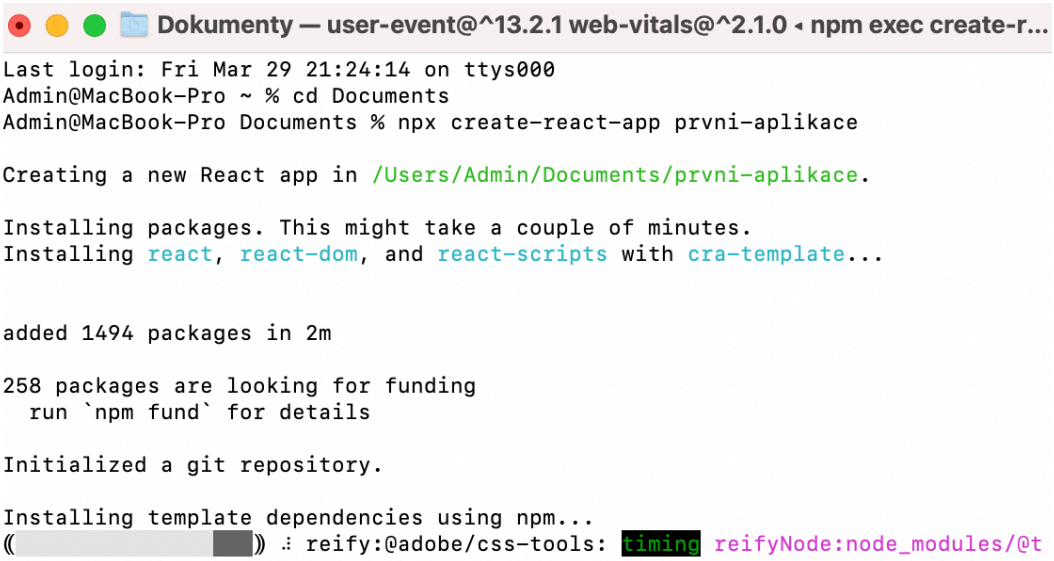
- Široká nabídka rozšíření a aktivní komunita: Na základě dat z roku 2023 ze serveru statista je React druhá nejpopulárnější knihovna pro vývoj webu hned za Node.js. Ve statistice je uvedeno, že právě 40,58 % vývojářů využívá React. [33] Díky této popularitě pro React neustále vznikají nové knihovny a jejich aktualizace, které jsou k dispozici pro ostatní vývojáře. Takto aktivní komunita je užitečná i pro řešení různých „errorů" a chyb s tím spojených, které v průběhu vývoje či chodu aplikace mohou nastat. S velkou pravděpodobností se již se stejným či velmi podobným problémem některý vývojář setkal a jeho řešení je tak dostupné na stránkách jako stackoverflow (https://stackoverflow.com).

3.2.3 Práce s Reactem

Pro pochopení, jak React využít, je vhodné prvně nahlédnout přímo do zdrojového kódu. Následující podkapitoly pojednávají o práci s Reactem a jeho komponenty.

3.2.3.1 Tvorba projektu

Projekt je vždy vytvářen od úplného začátku. Je nezbytné do PC nainstalovat všechny potřebné technologie (např. node). Dle autora literatury React Key Concepts je nejjednodušší postup pro tvorbu nového projektu využití terminálu, kam se zadá příkaz npx create-react-app my-react-project [34]. Před napsáním tohoto příkazu je dobré zvolit lokalitu adresáře, do kterého bude projekt vytvořen. Postup by mohl vypadat následovně.



Obrázek 6: Tvorba nového React projektu, vlastní zpracování

3.2.3.2 Struktura projektu React

Struktura projektu je automaticky vytvořena, stejně jako tomu bylo u Spring Bootu. Obsahuje níže zmíněné direktoráře a soubory [34]:

- node_modules: Adresář obsahující seznam všech potřebných závislostí (dependecies) pro chod aplikace.
- public: Adresář, který obsahuje všechny statické soubory (obrázky, fonty, html stránky atd.).
- src: Adresář obsahující zdrojový kód pro všechny soubory.
- packege.json: Konfigurační soubor pro projekt.
- package-lock.json: Soubor, který obsahuje metadata o instalovaných závislostech (včetně verzí).

Obrázek 7: Struktura vygenerovaného projektu, vlastní zpracování ve Visual Studio Code

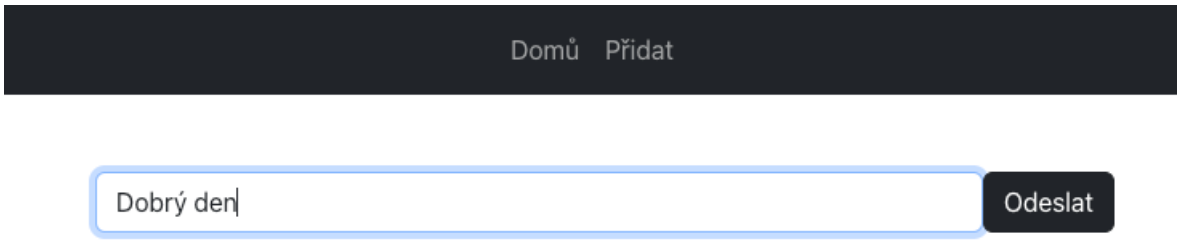
3.2.3.3 Tvorba šablon

Aby měl uživatel možnost cokoliv v aplikaci provádět, je třeba definovat jednotlivé šablony, minimálně pro CRUD operace. Tyto šablony budou komunikovat s vyvinutým, případně zamýšleným, API. Uvnitř šablony tak budou vytvořena tlačítka, která budou tyto požadavky inicializovat, a také hierarchické struktury, jež budou schopny data dynamicky zobrazit. Při zachování stejného typu ukázek pro aplikaci na pozdravy by bylo nutné vytvořit šablony pro zobrazení dat, ale i jejich náhled, mazání a přidání. Část zdrojového kódu pro odeslání dat na server by vypadala následovně.

```
try {
	const response = await axios.post('/pozdravy', {
		pozdrav: greeting,
	});

	if (response.status === 200) {
		console.log('ÚSĚCH');
		setGreeting('');
	} else {
		console.log('CHYBA PŘI ODESÍLÁNÍ');
	}
} catch (error) {
	console.log('CHYBA: ' +  error);
}
```

Ukázka kódu 11: Zaslání požadavku na server, vlastní zpracování



Obrázek 8: Zaslání požadavku z pohledu uživatele, vlastní zpracování

3.2.3.4 Zprovoznění aplikace

Pro to, aby aplikace fungovala, jak je zamýšleno, je nezbytné využít a instalovat další technologie (knihovny) do projektu, a to pro rozšíření základních funkcionalit.

- Axios: „Java-Scriptová“ knihovna, která zjednodušuje práci s http požadavky a odpověďmi. Obsahuje asynchronní funkce [34].
- Bootstrap: Poskytuje designově předdefinované styly, které se aplikují přidáním specifické třídy. [34]
- React-router-dom: Zajišťuje routing (změny URL adres) [34].

Za využití výše zmíněných technologií lze dovést aplikaci do funkčního stavu, kdy knihovna bootstrap není nezbytná. Jedná se o rychlý designový nástroj, který umožní přidávat styly bez nutnosti tvořit css třídy vlastní. Naopak axios je velmi důležitý pro „handling“ jednotlivých end-pointů. Poslední zmíněná technologie (react-router-dom) je využita pro přesměrování z jedné URL adresy na jinou. Toho lze využít například v navigačním menu, které přesměrovává uživatele na různé URL adresy a klientovi zpřístupňuje všechny aplikační funkce. Samotné routy se však definují do hlavní .js třídy (App.js). [34] To by mohlo vypadat následovně.

```
const App = () => {
 return (
        <Router>
                <div>
                        <Navbar />
                        <Routes>
                                <Route path="/" element={<Pozdravy />} />
                                <Route path="/pridat" element={<PridatPozdrav />} />
                        </Routes>
                </div>
         </Router>
 );
};
```

Ukázka kódu 12: Návrh směrování, převzato a upraveno [34]

3.2.3.5 Inicializace

Projekt se inicializuje napsáním příkazu npm start do příkazového řádku. Defaultně se aplikace spustí na adrese localhost:3000, kde je v počátečním stavu točící se logo Reactu s odkazem na dokumentaci. [34]

V případě, že je na lokálním počítači současně spuštěn webový server spolu s databází, je inicializována serverová část aplikace (Spring Boot) a právě tato klientská část je pak výsledkem plnohodnotná webová aplikace, která byla poskládala ze všech zmíněných komponent. Při spojení všech ukázkových souborů do jednoho velkého softwaru by uživateli byla k dispozici naprogramovaná aplikace se seznamem pozdravů. Ta by mohla vypadat následujícím způsobem.

Obrázek 9: Vizualizace systému, vlastní zpracování

4 Praktická část

Následující podkapitoly jsou věnovány tvorbě aplikačnímu systému. Navazují na poznatky teoretické části.

4.1 Analýza

Webová aplikace bude tvořena pomocí zmíněných technologií Spring Boot 3.0. a React. Jelikož potenciální využití systému není jen u jedné firmy, ale systém lze využít pro několik desítek firem, musí se již od samotného počátku přemýšlet, jak docílit co nejvyšší efektivity a plynulosti, neboť systém bude pracovat s velkým objemem dat. Z tohoto důvodu musí být navržen tak, aby zvládl rychle a efektivně obsluhovat všechny požadavky. Je proto nezbytné rozdělit vývoj na dvě části, serverovou a uživatelskou.

Aby takovýto systém mohl obecně fungovat, je zapotřebí věnovat čas na řešení klíčových problémů. Je nezbytné vytvořit pro serverovou část ideální strukturu databáze, ve které bude optimalizována co nejkratší cesta ke správným uživatelským datům. Dále je třeba optimalizovat operace prováděné s daty, zajistit jejich bezpečnost včetně přístupu k nim. Pro klientskou část se musí implementovat dotazování na server i vytvořit intuitivní a snadno obsluhovatelné uživatelské prostředí, které umožní data zobrazit a pracovat s nimi.

4.1.1 Zabezpečení

Zabezpečení celého projektu je bezprecedentně důležité, neboť na serveru budou uchovávána uživatelská data, která podléhají zákonům, jakým je např. GDPR. Jakékoliv narušení by mohlo způsobit vážný problém. Bezpečnost je zajištěna na několika následujících úrovních.

- Struktura databáze: Databáze je navržena tak, aby uživatelská data byla vždy navázána na ID společnosti, čímž by v teorii měla být zajištěna správná provázanost jednotlivých dat.
- Uživatelský token: Při každém přihlášení či nové registraci je uživateli vygenerován nový token. Jedná se o token zvaný JWT, který má tři části. První z nich je takzvaný *HEADER*, který obsahuje data o šifrovacím algoritmu, který je využíván (HS256). Druhá část tokenu, do které jsou vložena data, je nazývána *PAYLOAD*. Tato data chceme v tokenu uchovat. V tomto konkrétním případě jde

o údaje uživatele (e-mail). Podle e-mailu si server dohledá id firmy, o kterou se jedná. Poslední částí je takzvaný *VERIFY SIGNATURE*, ve které je zašifrována unikátní sekvence znaků uložených na serveru. Takovýto JWT token se skládá vždy z těchto tří částí, kde jednotlivé části jsou odděleny tečkou. Správnost vygenerovaného tokenu lze ověřit na oficiální stránce https://jwt.io. [35]

- Přístup k datům na základě tokenu: Pro uživatele existuje vždy pouze jeden token, který má časově omezenou platnost. Tento token je na straně uživatele uchován v Cookies, odkud se automaticky přidává do každého zasílaného requestu na server. Na serverové straně jsou jednotlivé tokeny uloženy v databázi. Token na serveru ověřuje jeho platnost. Ta je nastavena s časovým limitem a existencí pouze jednoho tokenu pro uživatele. Tedy v případě, že platnost tokenu ještě nevypršela a daný uživatel se odhlásil a následně opět přihlásil, tak se mu při přihlášení generuje token nový a platnost staršího tokenu zaniká. Vytvoření tokenu je zajištěno přihlášením uživatele s pomocí uživatelského e-mailu a vlastního hesla. Při vytváření tokenu se tedy kontroluje existence uživatelského jména a korespondujícího hesla s ním, respektive s jeho otiskem, neboť uložení samotného hesla by bylo další bezpečnostní riziko, proto se ukládá pouze jeho otisk s využitím technologie Bcrypt. Po provedení těchto kroků je uživateli vygenerován nový token, který je obohacen o již zmíněnou šifrovací sekvenci.
- Nastavení CORS: Dle literatury Spring Boot and Angular CORS poskytuje další úroveň zabezpečení aplikace, které se věnuje zasílaným požadavkům na server. Díky této modifikaci na serveru lze určit kritické parametry pro bezpečnost. Dle autora jde například o adresu domény, od které server přijímá požadavky nebo o to, jaké metody těchto požadavků může akceptovat. Bez těchto základních parametrů by bylo možné volat API z úplně cizí domény. [16]

4.1.2 Uživatelské prostředí

Uživatel skrze vytvořené GUI komunikuje se serverem. Ten obstarává požadavky klienta, které jsou na server zasílány bez jeho vědomí (po kliknutí na tlačítko). Uživatel má k dispozici všechna svá data na obrazovce. Stránky, které jsou zobrazovány uživateli, se skládají z komponent, jelikož celý front-end musí být pojat dynamicky a zobrazovat data, která uživatel vložil na server. Tyto šablony jsou využívány k obsluze všech jednotlivých operací, jakými je detail, mazání či úprava, kdy

jsou uživateli zobrazena původní data. Bez těchto šablon by nebylo možné požadavky obsloužit a celý koncept vykreslování by nebylo možné realizovat.

Ať jde o šablony nebo o jiné html stránky, které obsluhují další požadavky klienta, jsou složeny z částí navigace a obsah. Komponenta navigace je vložena do každé stránky, zobrazuje a obsluhuje navigaci v uživatelském prostředí, díky kterému je klient schopen orientovat se v aplikaci. Tato navigace není však fixní, mění se v závislosti na uživatelských Cookies. Existují dva následující stavy:

- Nepřihlášen: V tomto stavu nejsou uloženy žádné cookies v browseru pod jménem token. Navigace nabízí pouze dvě následující možnosti, registrovat se nebo přihlásit se. Po úspěšném provedení jedné z těchto dvou akcí jsou vloženy cookies do browseru klienta pod jménem token, který obsahuje samotný JWT token. Aplikace automaticky detekuje přítomnost tokenu a aktualizuje navigaci uživatele na následující stav přihlášen, viz dále.
- Přihlášen: Ve stavu přihlášen má uživatel v cookies pod jménem token nějaká data. Navigace se nachází ve stavu přihlášeného uživatele, kdy jsou v navigaci nabídnuty stránky dedikované zobrazování a práci s daty. Dále je zde k dispozici tlačítko pro odhlášení, které nahrazuje tlačítka pro přihlášení a registraci z předchozího stavu. V případě, že se nebude jednat o token nebo nebude validní, a to i v případě vypršené platnosti, systém tento fakt automaticky detekuje při zaslaném požadavku a změní se stav zpět na nepřihlášen, kdy se současně i aktualizuje navigace a znepřístupní se tak celá aplikace. Při pokusu do browseru vložit cookies s jakýmikoliv neplatnými daty si systém poradí, neboť pro prvotní aktualizaci navigace je zapotřebí odeslat požadavek na server. Při zaslaném požadavku se vždy tato cookies kontrolují a v případě neplatných dat se automaticky odstraní, a tudíž klientovi zůstane nadále status nepřihlášen.

Mimo jiné je důležité zajistit, aby uživatel měl přístup jen a pouze ke svým datům a nemohl se dostat k datům cizím. Za předpokladu, že je přihlášen do systému, je zapotřebí zavést další bezpečnostní opatření, díky kterému je právě tato situace ošetřena. To znamená, že při každém dotazu je nutné zkontrolovat, zda je dotaz skutečně proveden na data, která jsou provázána s identifikátorem firmy, který je totožný s přihlášeným uživatelem. To platí nejen pro zobrazování dat, ale i pro jejich

veškerou modifikaci. V případě, že takováto situace nastane, uživateli se nesmí data zobrazit. Funkce je bezpečnostní pojistkou pro uživatele využívající tento software.

4.2 Tvorba aplikace

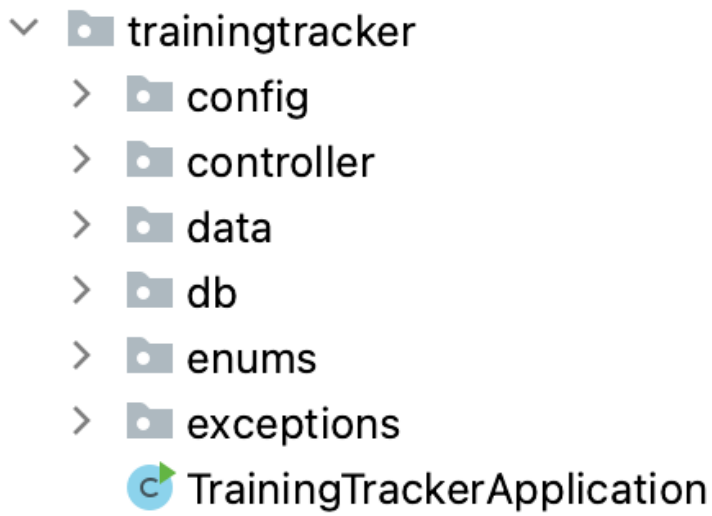
Při tvorbě aplikace je důležité dodržet veškeré zmíněné postupy z teoretické části. První je nutné rozdělit vývoj do dvou částí, kdy je zapotřebí se zaměřit zvlášť na serverovou a uživatelskou část. Tvorba serverové aplikace je založena na MVC architektuře pro zajištění správné hierarchie kódu. Struktura je tedy rozdělena do několika „podbalíčků“, které vždy zajišťují určitou funkcionalitu. Tvorba aplikace byla započata od serverové části, neboť programováním tohoto úseku by byly nalezeny případné nedostatky nutné k rozšíření v datové architektuře. Předtím, než se programují jednotlivé end-pointy aplikace, je potřebné vytvořit rozhraní, které zvládne tyto požadavky obsloužit. Jedná se o rozhraní, jež komunikuje s databází a předává tak jednotlivé výsledky *SELECTŮ* ve formě Java objektů, se kterými je dále možné snadno nakládat. Mimo to byl pro každý tok dat vymodelován DTO model, který je věnován vždy jednomu, případně několika, požadavkům, které obsahují požadovaná data. Na klientské straně je nutná implementace potřebných koncových bodů spolu s vytvořením několika šablon, které budou schopny tato data dynamicky zobrazovat. Níže v této práci jsou rozebrány konkrétní implementace částí aplikace.

4.3 Návrh architektury

Následující podkapitoly jsou věnovány architektuře systému, a to klientské i serverové části. Současně jsou vizualizovány na obrázcích s čísly 10 a 11.

4.3.1 Serverová část

Model serverové části aplikace byl navrhnut tak, aby byl schopen pojmout veškeré potřebné informace, které uživatel systému potřebuje vložit. Je zcela zásadní zvolit efektivní řešení, které umožní účinné zacházení s daty, a to jak v případě manipulace s nimi, tak i s ohledem na přístup k nim. Model serveru, znázorněn níže na obrázku č. 10, je rozdělen do několika balíčků. Hlavní balíček (trainingtracker) obsahuje tzv. Main třídu, která je zodpovědná za spuštění programu. Mimoto zahrnuje řadu „podbalíčků“, do kterých jsou strukturovány jednotlivé Java třídy dle svých funkcionalit.

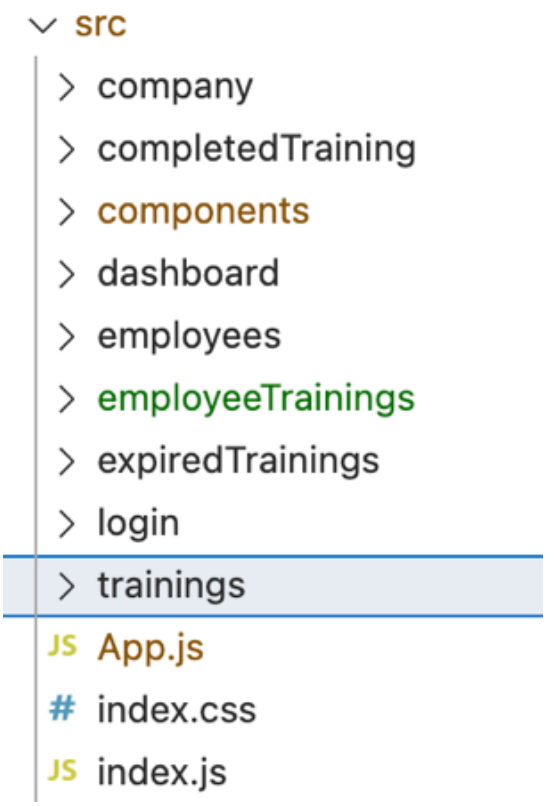


Obrázek 10: Model serveru, vlastní zpracování v Intellij IDEA

Balíček config obsahuje třídy týkající se konfigurace zabezpečení serveru. Jde konkrétně o určení, který end-point (URL adresa) je využíván k přihlášení a odhlášení. Dále se zde nachází třídy odkazující na hashovací funkci či implementaci samotné logiky, která značí, jaké stránky jsou uživateli dostupné bez autentizace a případné autorizace. V balíčku controller se nachází třídy s jednotlivými ovladači. Každý tento ovladač má svou základní URL adresu (například: /employee), která je obohacena o identifikátor prostředku, jež se odlišuje dle akce a dotazu na server. Složka data se dělí do dvou podsložek. První z nich je DTO, to definuje strukturu dat, která se přenáší z databáze k uživateli nebo opačným směrem. Druhou složkou jsou services neboli služby. V této části se vyskytují třídy definující mapování dat od uživatele či z tabulek databáze do specifického DTO. Metody těchto tříd jsou volány z kontrolérů. Společně s mapováním zajišťují vyhledávání relativních dat podle uživatelského tokenu. V některých případech jsou obohaceny o další logiku, jakou je filtrace či samotná generace uživatelských dat. V sekci db je zahrnuto vše týkající se databáze. Zde jsou definovány jednotlivé tabulky, a to včetně vazeb mezi nimi. Nachází se zde také repozitáře, které se dědí ze třídy JpaRepositories, díky čemuž jsou implementovány základní operace databází a jejich tabulky. Některé repozitáře jsou obohaceny o vlastní query, které jsou využívány pro vyhledávání, filtraci či propojení konkrétních dat. Uvnitř adresáře enums jsou vytvořeny vlastní proměnné, které reprezentují roli uživatele. Balíček exceptions definuje krajní situace programu, jinak řečeno výjimky, které mohou v případě nefunkčnosti z jakéhokoliv důvodu nastat, a je potřeba je nějakým způsobem ošetřit.

4.3.2 Klientská část

Klientská část je využívána k interakci s uživatelem. Uživatel skrze toto prostředí zasílá specifické požadavky na server, který odesílá zpět odpovědi dle toho, jak byl naprogramován. Uživatelské rozhraní je navrženo tak, aby bylo uživatelsky přívětivé a klient tak mohl snadno a zcela intuitivně využívat všechny funkcionality systému. Struktura rozhraní na obrázku níže byla vytvořena v programu Visual Studio Code.



Obrázek 11: Struktura GUI, vlastní zpracování ve Visual Studio Code

Projekt byl prvně vygenerován pomocí příkazu npx create-react-app frontend a následně očištěn o nepotřebné .js soubory. Struktura projektu je rozvržena do několika složek, uvnitř kterých jsou vždy .js soubory, jež zobrazují data dle nastavených parametrů (tabulky) a dále s nimi pracují (podrobný náhled, úprava, mazání dat).

Spouštěcím souborem je soubor s názvem index.js, který generuje veškeré potřebné podklady do browseru uživatele z App.js., jež je mozkem celého projektu, neboť se v tomto souboru nachází klíčové kódy, bez kterých by nebylo možné zajistit správné fungování systému. Konkrétně se zde implementují algoritmy pro přihlašování uživatele a následné přesměrování, dekódování a ukládání tokenu, obsluha cookies nebo routy aplikace. V adresáři components, tj. v komponentech, je

vytvořena navigace, díky které se uživatel může snadno pohybovat skrze webovou aplikaci, a docílit tak zamýšlených výsledků.

V jednotlivých balíčcích jsou vždy vytvořeny soubory, které zajišťují zasílání požadavků na server a následné zobrazení konkrétních dat. V těchto balíčcích se nachází soubory, které zajišťují zobrazení přehledu, úpravu dat a jejich mazání. Aby z uživatelského hlediska nedošlo k nechtěnému smazání dat, je pro vybrané mazací operace uživatel prvně přesměrován do nového okna. Uvnitř tohoto okna je zobrazen detail dat spolu s dvěma tlačítky (zpět, smazat). Do adresáře login jsou vloženy algoritmy týkající se přihlašování a registrací nových uživatelů. Dochází ke komunikaci se serverem a na základě jeho odpovědi je provedena akce. Ve složce company jsou uloženy kódy pro zobrazení a editaci dat, které uživatel zadal při registraci o společnosti.

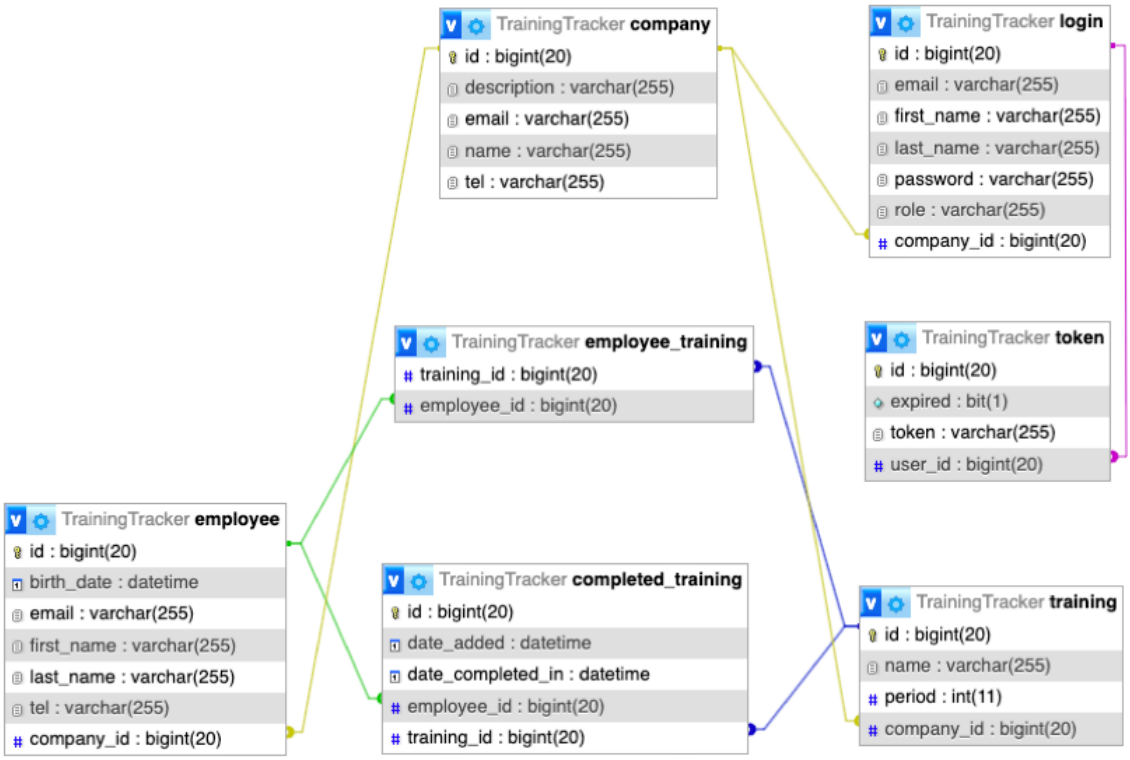
Uvnitř balíčku dashboard je soubor Dashboard.js, na který je uživatel vždy automaticky přesměrován po registraci nebo úspěšném přihlášení. Soubor je dedikován přehledu dat, jež uživateli dává informace o tom, která školení musí zaměstnanci firmy v blízkém horizontu absolvovat. Je však zapotřebí dat v určitých pravidelných intervalech aktualizovat, neboť je vždy třeba vložit záznam o tom, kdy a který zaměstnanec byl nově proškolen. Přehled se tak pokaždé aktualizuje a uživatel získává aktuální sérii dat (nutných školení k provedení). Po zadání pracovních dat do systému je uživateli k dispozici nástroj pro správu a monitoring těchto dat.

4.4 Implementace řešení

Následující podkapitoly jsou věnovány implementaci řešení daných problémů.

4.4.1 Databáze

Je třeba splňovat pravidla normalizace, která zajistí správnost a integritu celého systému. Díky Spring Bootu a využití dalších pomocných knihoven není nutné mít pokročilou znalost SQL, neboť databáze není tvořena ručně. Je realizována anotací přímo v jazyce Java. Na obrázku níže je znázorněna struktura databáze pro tento projekt, která byla zachycena softwarem phpMyAdmin.



Obrázek 12: Model databáze, vlastní zpracování v phpMyAdmin

Na obrázku č. 12 je vyobrazen serverový model samotného systému. Struktura databáze je tvořena sedmi tabulkami. Všechny tabulky jsou ve vazbě 1: N, vyjma zaměstnanec-školení (employee_training), ta bude představena níže. Tabulka company prezentuje společnost, jež bude využívat tento software. Nese svůj identifikátor, jméno a popis společnosti, dále kontaktní údaje společnosti pro potencionální komunikaci se zákazníkem. Tato tabulka je zcela klíčová pro fungování celého systému, neboť se na ni přímo či nepřímo váží tabulky ostatní. Přímo na firmu jsou navázány tři tabulky, což zajistí propojení dat se specifickou firmou skrze cizí klíč. Konkrétně jde o tabulky login (přihlášení), training (školení) a employee (zaměstnanec). Do tabulky login jsou ukládána data uživatelů, kteří mají přístup k aplikaci. Jedná se o e-mail, heslo, jméno, příjmení, roli a vazbu na firmu. Uživatel do systému bude přihlášen pomocí e-mailu. Z toho důvodu chybí v návrhu uživatelské jméno. Na tento login je vázaná tabulka token, která nese vazbu na data uživatele, obsahuje identifikátor, samotný token a platnost daného tokenu. Tabulka školení se váže ke společnosti, neboť je zapotřebí identifikovat, pod kterou společnost dané školení spadá. Nese tyto informace o školení: identifikátor, název, perioda a vazba. Paralelně je připojena tabulka zaměstnanců na společnost. Ta obsahuje veškerá data, která jsou o zaměstnancích uchovávána. Konkrétně jde o jméno, příjmení, datum

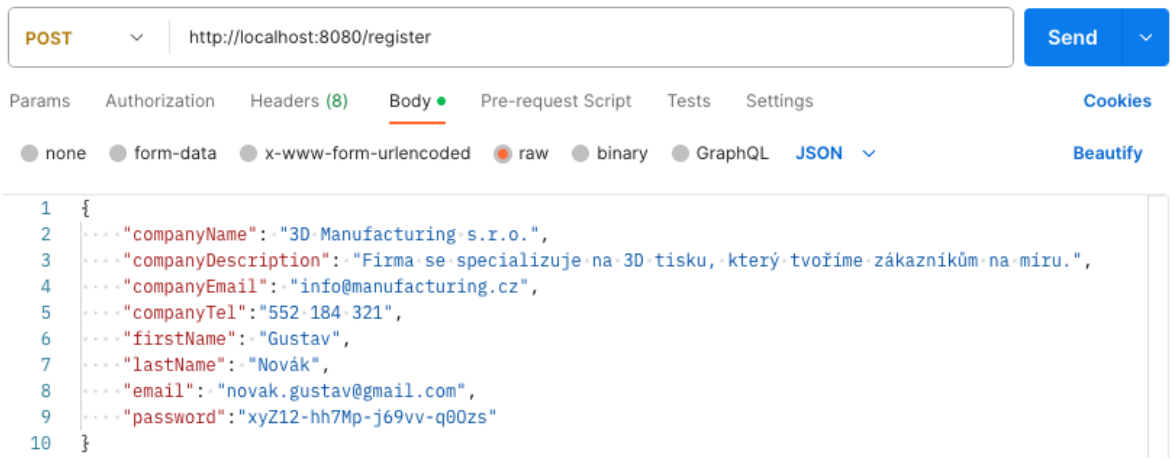
narození, kontaktní údaje (e-mail a telefon), dále identifikátor a cizí klíč odkazující na společnost. Šestou tabulkou z návrhu je tabulka zaznamenávající data provedených školení (completed_training). Ta je využívána k záznamu uskutečněného školení, které zaměstnanec absolvoval. Obsahuje tedy dva cizí klíče, dále identifikátor, datum uložení a datum ze dne provedení školení. Tato tabulka se již nemusí odkazovat na společnost, neboť vazby jsou uloženy v předchozích tabulkách a nabídka školení nebo zaměstnanců bude limitována cizími klíči samotné společnosti. Poslední tabulkou je employee_training (zaměstnanec–školení). Nejedná se o běžnou tabulku. Jde o tabulku pomocnou, bez vlastního identifikátoru, do které jsou ukládány záznamy kombinací cizích klíčů. Bez ní by nebylo možné vytvořit vazbu mezi zaměstnancem a školením, jelikož se nachází ve vztahu N:M, kdy jeden zaměstnanec firmy může absolvovat jedno a více školení a současně lze jedno školení přiřadit více zaměstnancům.

4.4.2 Koncové body

Koncové body aplikace zajišťují vzájemnou komunikaci mezi serverem a klientem. Tímto způsobem spolu obě strany komunikují.

4.4.2.1 Registrace a přihlášení uživatele

Pro úspěšnou registraci zcela nového uživatele je zapotřebí odeslat následující údaje: jméno firmy a její popis, firemní e-mail a telefon, jméno a příjmení uživatele, jeho e-mail a heslo pro přístup k účtu. Pokud dojde k úspěšnému provedení registrace, bude uživatel automaticky přihlášen a přesměrován na hlavní stránku systému. Jestliže registrace nebude úspěšná, bude mu vrácena chybová hláška a registraci bude zapotřebí opakovat. Pro znázornění struktury dat je k dispozici následující ukázka.



Obrázek 13: Koncový bod pro registraci uživatele, vlastní zpracování v Postman

Pro přihlášení do systému je zapotřebí, aby uživatel vyplnil přihlašovací formulář s e-mailem a heslem. V případě správného zadání dat bude uživateli přidělen nový unikátní token, který mu umožní opětovný vstup do systému a správu jeho dat. V opačném případě zůstane na přihlašovacím formuláři, dokud nezadá správné údaje.

4.4.2.2 Načítání, odesílání, úprava a mazání dat

Tento systému musí implementovat minimálně veškeré CRUD operace, které se vztahují na každou tabulku, jež je součástí aplikace. Pro všechny tyto tabulky (vyjma tokenu) tak musí být vytvořeny vlastní API end-pointy, které umožní provádět tyto operace:

- GET: Zobrazení dat, jeden výsledek výsledku (podle id) i seznamy (vše nebo podle parametru např. LIMIT).
- POST: Ukládání dat na server, id se generuje automaticky.
- PUT: Editace záznamů, nezbytné specifikovat id.
- DELETE: Odstranění dat, nutno přidat id.

4.4.3 Implementace Query

Do projektu je zakomponováno několik desítek query, které obsluhují požadavky na databázi. Tyto dotazy není nezbytné psát ve standartním formátu například SELECT c.name FROM company c ORDER BY c.id, neboť existuje o dost jednodušší řešení. Jak již bylo zmíněno v teoretické části, existuje rozšíření zvané spring-boot-starter-data-jpa, které zautomatizuje psaní těchto dotazů. Díky znalosti tohoto frameworku stačí přidat název metody dle dokumentace, která umožní vyhledávání dat z databáze na základě vstupních údajů.

Konkrétní implementace je vyobrazena na ukázce níže. Prohledává databázi uživatelů (UserEntity) za účelem najít specifický e-mail, který je dán v parametru metody. Pokud by bylo potřebné najít uživatele např. podle identifikátoru, změnil by se pouze název metody z findByEmail na findById a do parametru metody by byl přidán (Long id).

```
public interface UserRepository extends JpaRepository<UserEntity, Long> {
   Optional<UserEntity> findByEmail(String email);
}
```

Ukázka kódu 13: Generovaný dotaz, vlastní zpracování

Pokud by nastal případ, že by tyto základnější funkce nedostačovaly našim požadavkům, a to ať již pro komplexní dotaz nebo cílené optimalizace, je možné napsat ručně přímo do stejné třídy vlastní query. To je umožněno zapsáním anotace @Query s atributy value, kam je vložen samotný script s atributem nativeQuery = true. Příklad takového požadavku je popsán níže.

```
public interface TokenRepository extends JpaRepository<TokenEntity, Long> {
   @Query(value = "SELECT t " +
         "from TokenEntity t " +
         "INNER JOIN " +
         "UserEntity u ON t.user.id = u.id " +
         "WHERE u.id =:userId " +
         "AND (t.expired = false)",
         nativeQuery = true
   )
   List<TokenEntity> findAllValidTokens(Long userId);
}
```

Ukázka kódu 14: Dotaz na token, vlastní zpracování

Následné využití této metody obstarává služba. Před prací s repozitářem je nezbytné provést injektáž závislostí. Poté již bude možno s injektovaným repozitářem svévolně pracovat.

```
@Service
public class LoginService {
   @Autowired
   private TokenRepository tokenRepository;
```

Ukázka kódu 15: Injektáž závislosti, vlastní zpracování

4.4.4 Autorizace požadavků

Po přihlášení do systému je uživateli zaslán JWT token, který je uložen klientovi do cookies. Token nese svůj zašifrovaný signature neboli podpis, což brání možnosti padělání tokenu. Kdykoliv klient zašle požadavek na server, je token odeslán součástí requestu v headers. Na serverové straně je token ověřován díky implementaci a konfiguraci frameworku spring-boot-starter-security.

Pro snazší představu jsou přidány ukázky kódů, ve kterých uživatel kliká na tlačítko zaměstnanci v GUI, které spustí akci zaslání požadavku GET spolu s JWT tokenem.

```
const token = Cookies.get('token');
const response = await axios.get(
        '/employee',
        {
                params: {
                tokenDTO: token,
                },
                headers: {
                Authorization: `Bearer ${token}`,
                },
        }
);
```

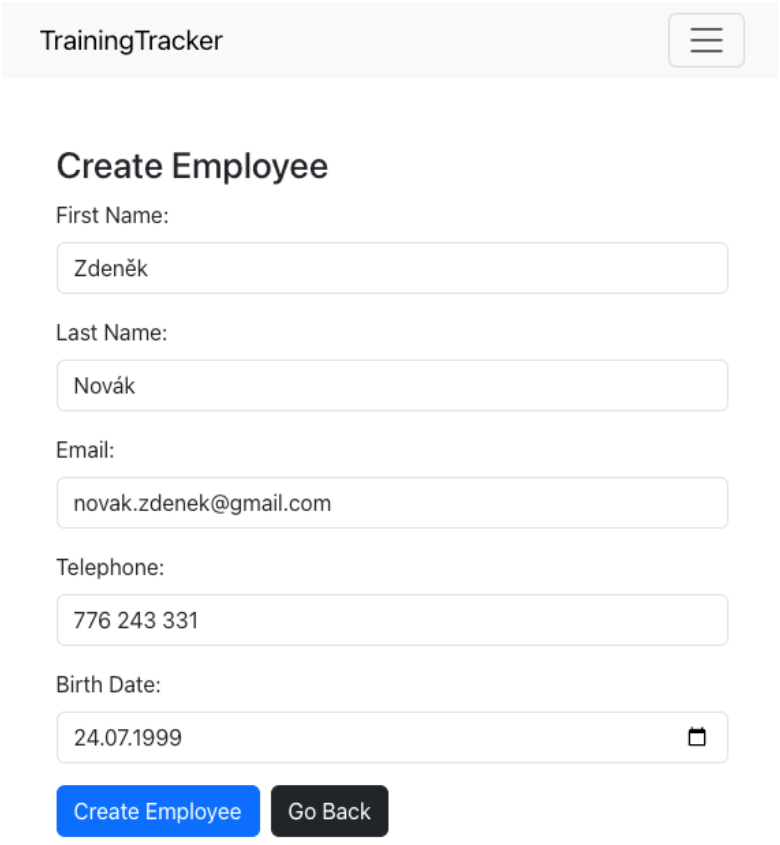
Ukázka kódu 16: Zaslání požadavku na server, vlastní zpracování

Na serverové straně se tento požadavek přijme. Následně dojde k ověření identity uživatele, při kterém je zjištěna platnost tokenu na základě zaslaného tokenu v headers. Pokud tato operace proběhne v pořádku, provede se kód napsaný uvnitř metody obsluhující koncový bod "/employee" s metodou GET. Pro tento konkrétní příklad se zavolá metoda showAllEmployees.

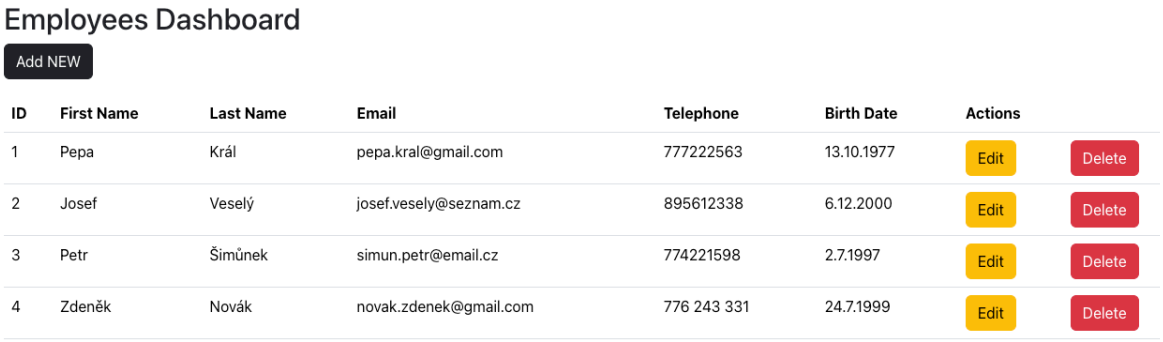
```
@GetMapping
public ResponseEntity<List<EmployeeDTO>> getAllEmployees(@RequestParam String
tokenDTO){
   return ResponseEntity.ok(employeeService.showAllEmployees(tokenDTO));
}
```

Ukázka kódu 17: Koncový bod pro zobrazení zaměstnanců, vlastní zpracování

Server následně postupuje hlouběji do zdrojového kódu, kde kontrolér volá metodu ze služby. Služba následně volá repozitář. Uvnitř repozitáře je proveden dotaz na server, který vrací data skrze DTO. Ta postupně putují stejnou cestou přes return statement (návratový příkaz) zpět k uživateli. Za předpokladu, že jsou uživateli data k dispozici a nedošlo k žádné chybě, je uživatel přesměrován na jinou URL adresu, které je dedikována jiná zobrazovací šablona. Na tuto šablonu jsou následně data vyobrazena.



Obrázek 14: GUI – Přidání zaměstnance, vlastní zpracování



Obrázek 15: GUI – Seznam zaměstnanců, vlastní zpracování

4.4.5 Odhlášení uživatele

V případě, že si uživatel nepřeje provádět další operace ve webovém rozhraní, je mu nabídnuta možnost se odhlásit pomocí navigace (tlačítko vpravo nahoře). Kliknutí na tlačítko „odhlásit se“ spustí následující akci. Z klientské strany je odebrána Cookies s JWT tokenem. Systém tuto změnu detekuje a uživatel je přesměrován na výchozí stránku pro nepřihlášené. Dostává se zpět do stavu nepřihlášen, který byl zmíněn výše.

Existuje zde i možnost, že se klient sám neodhlásí. Aby tato akce nepředstavovala potenciální bezpečnostní riziko a nezůstávaly tak dlouze platné tokeny v databázi, je zde vytvořena časová podmínka, respektive doba, po které je token automaticky zneplatněn.

Pokud se uživatel bude chtít znovu přihlásit, token mu bude přidělen nový. Jeho starý token v případě předčasného odhlášení není v tuto chvíli nutné deaktivovat, jelikož je šifrovaný a má platnost pouhých 24 hodin. V aktuálním stavu je systém připraven na další rozšíření systému, jakožto možnost zůstat přihlášen napořád, jelikož eviduje primární token do databáze. Tomuto přístupu se však tato práce dále nevěnuje a jedná se tak o nadstavbu aktuálního systému.

4.4.6 Plánování školení

Jedním z cílů tohoto systému je, aby zajistil včasné upozornění uživatele, že je potřeba provést školení některého ze zaměstnanců. Takovýto algoritmus by se pro lepší přehlednost dal rozdělit do tří částí.

Prvním krokem je vytvořit end-point, ve kterém jsou přijata potřebná data k obsluze požadavku (token). Na základě tokenu uživatele je získán seznam identifikátorů všech zaměstnanců společnosti spolu s identifikátorem požadovaného

školení. Druhým requestem se získává seznam identifikátorů provedených školení spolu s identifikátorem zaměstnance a školení, které absolvoval.

V druhé části se tato získaná data musí seřadit a „očistit“ o „duplicity“. To platí pro zmíněný druhý dotaz, kdy je velmi pravděpodobné, že se zde „duplicity budou nacházet“, např. jsou vrácena data: {1,2,2}, {2,2,2}. To nám říká, že uživatel dané školení dříve absolvoval. V tomto případě je nezbytné zjistit, které z těchto školení má novější datum absolvování, to zachovat a zbytek „duplicit“ odstranit.

Ve třetí části se cyklem prohledávají data ze seznamů. V případě, že je nalezena shoda, dochází k výpočtu času na základě dat definovaných uživatelem v systému, které se vyhledají podle ID. Takováto data by vypadala následujícím způsobem (tzn. datumSkoleni = 05.04.2022; interval = 24; delay=90; datumUpozorneni = 06.01.2024). V závislosti na aktuálnímu datu (currectDate) se výsledná data vyhodnotí a uloží do nového DTO, které je přenese k uživateli. Takovýto výsledek by vypadal následovně (obrázek 16).

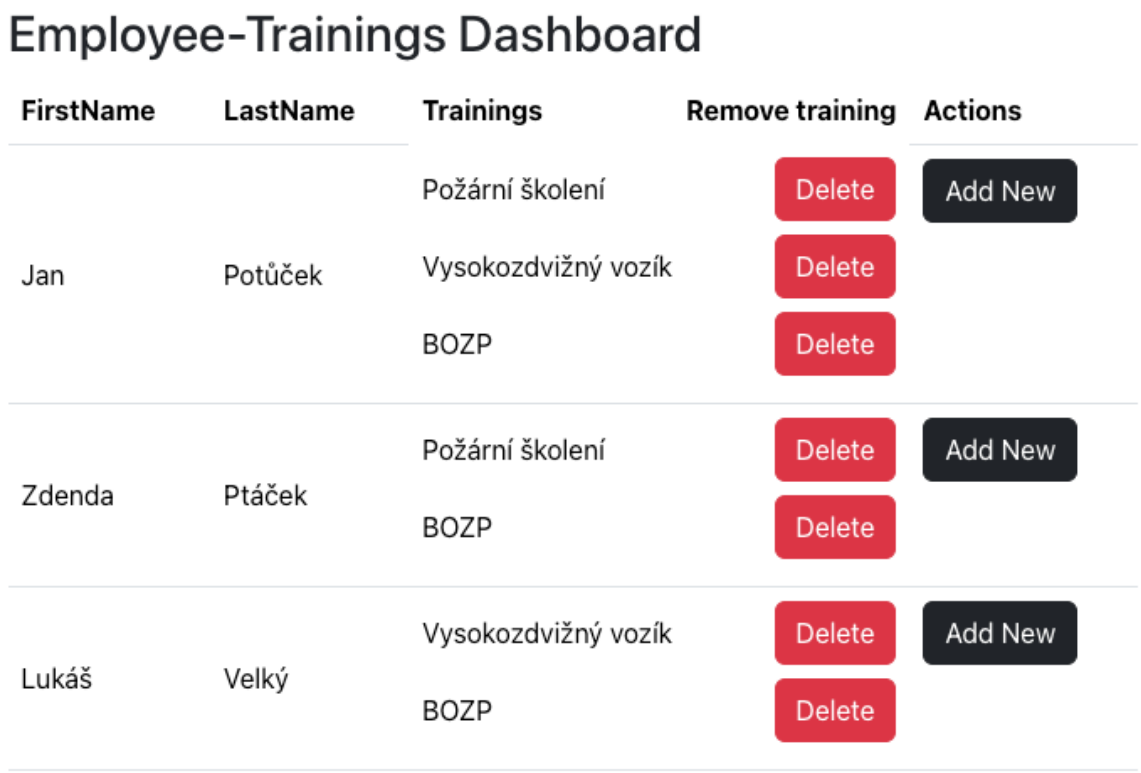
Training reminders

First Name	Last Name	Training Name	Deadline Date	Remaining Days
Jan	Potůček	BOZP	18. 7. 2024	88
Zdenda	Ptáček	BOZP	NO TRAINING DONE	0
Lukáš	Velký	BOZP	15. 6. 2024	55
Lukáš	Velký	Vysokozdvižný vozík	NO TRAINING DONE	0
Jan	Potůček	Požární školení	15. 2. 2024	-65
Zdenda	Ptáček	Požární školení	12. 2. 2024	-68

Obrázek 16: GUI pro varování nutnosti provést školení, vlastní zpracování

Na klientské straně je přidána logika pro zpracování a zobrazení dat. Data jsou seřazena dle ID školení, aby byl výsledek přehledný a školení se mohla provádět pro více zaměstnanců současně nikoli individuálně. Výsledkem je zobrazení dnů, které zbývají k vypršení platnosti školení zaměstnance. Pro větší přehlednost jsou přidány tři barvy, kde každá má svůj význam. Žlutá signalizuje, že stále školení platí, ale v blízké době (< 90 dní) bude ukončena platnost a bude nezbytná rekvalifikace. Červená

signalizuje problém, jelikož zaměstnanec nemá platné školení nebo mu ve velmi blízké době bude končit jeho platnost (<= 7dní). Pokud již školení daného zaměstnance není nutné provádět, lze individuálně odebrat monitoring ke školení. K tomu je vytvořeno následující GUI.



Obrázek 17: GUI pro odebrání a přidání vazeb zaměstnance na školení, vlastní zpracování

Po kliknutí na tlačítko „Delete“ je provedeno odstranění vazby mezi zaměstnancem a školením, tudíž se pro daný řádek dále nebude provádět kontrola. Dojde k aktualizaci výsledného pohledu.

4.5 Testování

Testování aplikace samotné je velice důležité. V tomto kroku je možné odchytit klíčové chyby, které mohly být v průběhu samotné tvorby programu vytvořeny. V podstatě se takovéto testování dá rozdělit do dvou kategorií, kdy obě hrají v tomto procesu důležitou roli.

4.5.1 Automatizované testy

Tyto testy jsou tvořeny programátory a jsou jakýmsi standardem současného programování. Využívají se k ověřování správnosti daného kódu spolu se zamýšlenou funkcionalitou. Programátor tak získává při každém zásahu do zdrojového kódu okamžité výsledky chodu aplikace. Na základě těchto výsledků lze také detekovat kritické chyby, které by mohly být potenciálně vpuštěny do produkce. Samotné testy se skládají ze tří částí: nastavení parametrů, provedení činnosti (volání metod), porovnání výsledku algoritmu oproti zamýšlenému výsledku. Za předpokladu, že se porovnávané výsledky shodují, test prošel úspěšně.

Tato webová aplikace má vytvořeno několik testů pro serverovou část. Ty vždy ověřují správnost programovaných funkcionalit. Pro tento projekt je rozdělena struktura testování do třech částí. Ty vždy ověřují část aplikace dle jména balíčku. Balíčky jsou následující:

- Repository: Obstarává testování jednotlivých repozitářů. Tyto testy samy o sobě nejsou nezbytné, jelikož metody z těchto testů jsou využívány v testech z následujících kategorií. Na druhou stranu lze naprogramováním těchto testů vyloučit problém s repozitářem za předpokladu, že test proběhne úspěšně. Případně se dají komplikovanější testy optimalizovat tak, že se pouze nasimuluje práce s repozitářem. Z níže vyobrazené ukázky lze detekovat založení nové entity, která je následně uložena. Správnost výsledku je uživateli sdělena po ukončení samotného testu.

```
@Test
public void createCompany(){
   CompanyEntity company = CompanyEntity.builder()
         .name("Testovací firma")
         .description("Pouze pro testování.")
         .email("testuju@gamil.com")
         .tel("776 224 331")
         .build();

   CompanyEntity savedCompany = companyRepository.save(company);

   Assertions.assertThat(savedCompany).isNotNull();
   Assertions.assertThat(savedCompany.getId()).isGreaterThan(0);
   Assertions.assertThat(savedCompany.getEmail())
          .isEqualTo(company.getEmail()
   );
}
```

Ukázka kódu 18: Test kódu s využitím databáze, vlastní zpracování

- Service: Průběh testování služeb funguje na podobném principu, kdy se definuje objekt, který se má odeslat. Pošle se skrze službu (service), která vrací odpověď ve formě DTO s uloženými daty. Úspěšnost testu je potvrzena, když se data vzájemně shodují.
- Controller: Pro kontrolér je zvoleno řešení bez využití databáze. Znamená to, že je pouze definováno, co provést, když je prováděna operace s databází. Využívá se takzvané anotace @Mock, která umožňuje „mockovat“, respektive simulovat, danou třídu spolu s jejím očekávaným chováním. Nevýhodou je, že v případě složitějšího testu, který kontroluje více funkcí naráz, může být poměrně komplikované takový test vytvořit. Na druhou stranu jde o velmi výhodný krok z hlediska efektivity a rychlosti provedení daného testu. V případě, kdy server má více jak sto takových testů, je tato změna velmi výrazná. Pro přehlednost je opět zvolen test pro přidání firmy.

```
@ExtendWith(MockitoExtension.class)
public class CompanyServiceTest {
   @Mock
   private CompanyRepository companyRepository;
   @InjectMocks
   private CompanyService companyService;
   @Test
   public void createCompany(){
      CompanyEntity company = CompanyEntity.builder()
            .name("Testovací firma")
            .description("Pouze pro testování.")
            .email("testuju@gamil.com")
            .tel("776 224 331")
            .build();

      CompanyDTO companyDTO = CompanyDTO.builder()
            .name("Testovací firma")
            .description("Pouze pro testování.")
            .email("testuju@gamil.com")
            .tel("776 224 331")
            .build();

when(companyRepository.save(Mockito.any(CompanyEntity.class))).thenReturn(compan
y);
      CompanyDTO savedCompany = companyService.createCompany(companyDTO);

      Assertions.assertThat(savedCompany).isNotNull();
   }
}
```

Ukázka kódu 19: Test kódu s využitím @Mock, vlastní zpracování

4.5.2 Uživatelská interakce

Testování lidmi je stejně nezbytné jako testování softwarem. Ve většině případů uživatelské testování odhalí chyby, nad kterými programátor do té doby ani

nepřemýšlel. Pokud tester takovou chybu najde, zavede se na server test nový, který bude chybu simulovat.

Konkrétní příklad může být následující: Tester zjistí, že při zadávání věku zaměstnance lze zadat čas, který ještě nenastal, např. datum narození v roce 2026. Nutně nemusí jít o úmysl, ale o prostou chybu při zadávání dat. Tato nekompletní data by však v budoucích verzích programu mohla způsobovat nepřesnosti např. v grafech, statistikách apod. Postup by byl tedy následující: Napsat test, který simuluje totožný stav testera. Poté vložit, jaký výsledek by měl nastat, např. věk vyšší než 15 let. Následně by tento test měl být zamítnut a může se začít samotný problém řešit. Test spouští znovu a „debuguje se“, dokud se nedocílí úspěšného stavu testu.

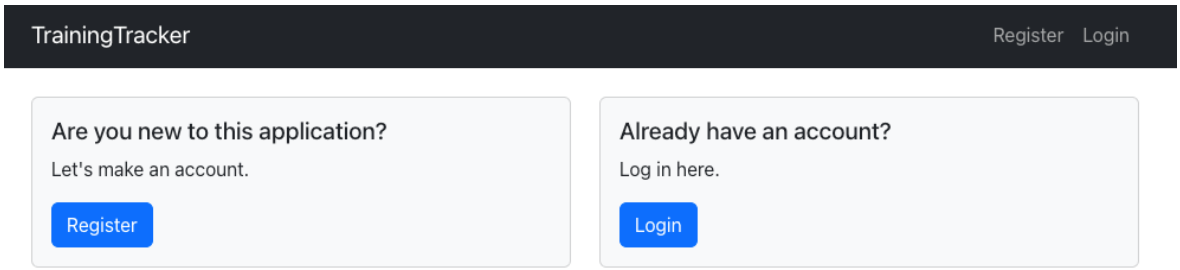
Nutno závěrem zmínit, že takovéto testy se nikdy nemažou, ani po opravení těchto bugů. Testy zde zůstávají a jsou součástí jakéhosi kontrolního balíku testů, které zajistí, že se stejné chyby nebudou nikdy znovu opakovat.

5 Shrnutí a diskuse výsledků

Teoretická část práce seznamuje s teoretickou podstatou a problematikou daného tématu. To přímo vede k nabytí potřebných vědomostí, které byly uplatněny při tvorbě webové aplikace v následující části.

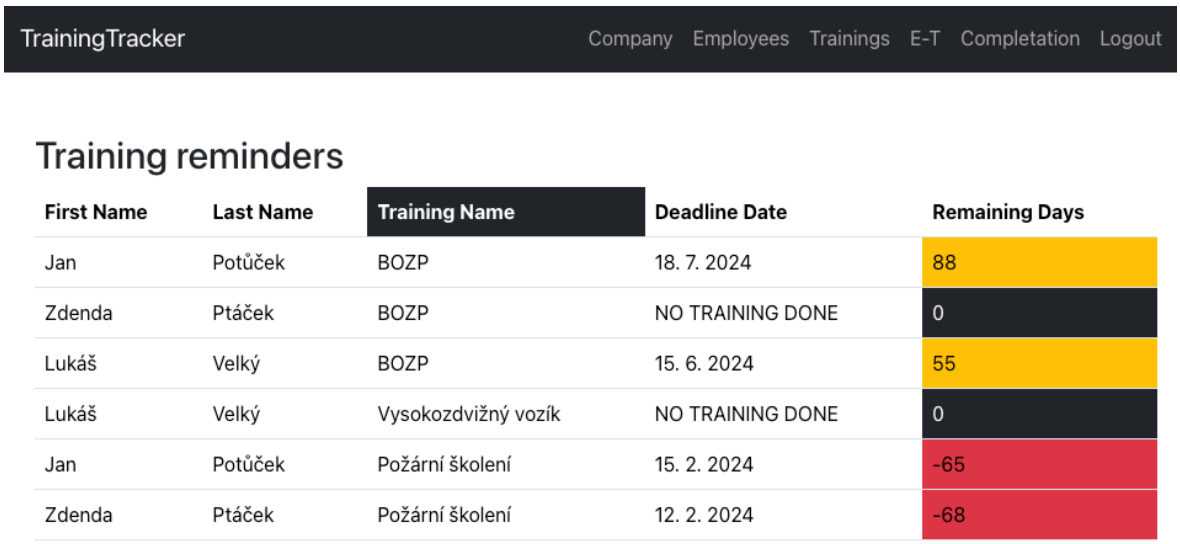
V praktické části byla analyzována samotná aplikace spolu s nastíněním problémů s jejich řešením. Vizualizuje a rozebírá klíčové praktiky a části kódů, bez kterých by takovýto systém bylo obtížné dokončit. Konkrétně se jedná o analýzu problémů při tvorbě aplikace. Mimoto vyobrazuje strukturu databázového serveru spolu s rozvržením toku dat a architektury systému. Zaměřuje se také na konkrétní implementace problému (nejen těch zmíněných v analýze) včetně ukázek zdrojového kódu. Poslední kapitola je věnována testování systému, na němž byla ověřena správnost vybraných funkcionalit.

Výsledný systém tak po implementaci všech teoretický poznatků bylo možné dovést do finálního výsledku, kdy je uživatel schopen se přihlásit a registrovat, odhlásit se, provádět operace s daty (CRUD) pro vybrané tabulky a má k dispozici modul, který upozorní na nutnost provést školení zaměstnanců. GUI aplikace je navrženo následujícím způsobem.



Obrázek 18: Načtení webové aplikace, vlastní zpracování

Uživateli se načte úvodní stránka (rozcestník) pro přihlášeni nebo registraci do systému. Ty jsou obslouženy formulářem, ze kterého se data odesílají na server. Server tato data vyhodnotí, v případě validity přidělí JWT token a uživatel je přesměrován na hlavní stránku pro přihlášené (obrázek č. 19).



Obrázek 19: GUI úvodní strany pro přihlášené, vlastní zpracování

Jak je z obrázku patrné, uživatel má v horní části obrazovky k dispozici navigaci, pomocí které obsluhuje data. Na hlavní obrazovce jsou zobrazena data uživatelů, kterým se termín školení blíží ke konci, kterým skončila platnost nebo nemají žádné platné školení. Tento náhled tak může nejen ovlivnit plánování školení zaměstnanců, ale také včas varovat majitele firmy, aby se nedostal do právních potíží.

Tato práce byla zaměřena na klientskou a serverovou část a tomu náležité technologie i koncepty. Aplikační systém založený na znalostech této práce je připraven ve funkčním stavu pro další testování a nasazení do produkce.

6 Závěry a doporučení

Výsledný systém tvořený v praktické části na základě teoretických poznatků je ve funkčním stavu, je připraven na potenciální využití klienty. Byla splněna zadaná kritéria, kde bylo vytvořeno uživatelské rozhraní, jednoduchá správa a vizualizace uživatelských dat spolu s upozorňovací funkcí, která informuje před končící platností školení. Po nahrání tohoto programu na server klienta, domovský server nebo server třetích stran bude k dispozici online všem uživatelům, kteří k němu mají přístup. Před takovýmto krokem je však nezbytné jednat v souladu se zákony a jinými nařízeními (udělení souhlasu se zpracováním údajů apod.).

Ačkoliv byla naplněna požadovaná kritéria, systém je dále možné rozšiřovat, implementovat nadstavby a nové funkce. Může jít například o již zmiňovanou funkci „Zůstat přihlášen“, pro kterou je připravena struktura databáze. Také lze systém dále rozšířit o analytické nástroje jako jsou grafy či statistiky, které povedou ke zefektivnění firemních procesů. V neposlední řadě lze umožnit správu dalších firemních dat. Celkově se však jedná pouze o doporučení, kam má vývoj softwaru dále směřovat.

7 Seznam použité literatury

[1] MICROSOFT. *What is Java Spring Boot?* Online. @ 2024 Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot. [cit. 2024-03-05].

[2] TECHTARGET CONTRIBUTOR. *inversion of control (IoC)*. Online. Aktualizace červenec 2021. Dostupné z: https://www.theserverside.com/definition/inversion-of-control-IoC. [cit 2024-03-06].

[3] BROADCOM. *Spring Dependency Injection*. Online. @ 2005-2024. Dostupné z: https://spring.academy/guides/spring-depedency-injection-. [cit. 2024-03-06].

[4] BROADOM. *Introduction to the Spring IoC Container and Beans.* Online. @ 2005-2024. Dostupné z: https://docs.spring.io/spring-framework/reference/core/beans/introduction.html. [cit. 2024-03-06].

[5] BROADOM. *10. Authentication*. Online. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.2.10.RELEASE/reference/html/jc-authentication.html. [cit. 2024-04-19].

[6] HARTINGER, David. *MVC architektura.* Online. Aktualizace 2. září 2020. Dostupné z: https://www.itnetwork.cz/mvc-architektura-navrhovy-vzor. [cit. 2023-10-14].

[7] AWS. *What is RESTful API?* Online. Dostupné z: https://aws.amazon.com/what-is/restful-api/. [cit. 2024-02-13].

[8] GOURLEY, David, Brian TOTTY, Marjorie SAYER, Sailu REDDY a Anshu AGGARWAL. *HTTP: The Definitive Guide (Definitive Guides)*. O'Reilly Media, Inc., 2002. ISBN 978-1-56592-509-0.

[9] BROADCOM. *Mapping Requests.* Online. @ 2005-2024. Dostupné z: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html. [cit. 2024-02-16].

[10] HTTP DEV. *302 Found*. Online. Aktualizace 2. srpna 2023. Dostupné z: https://http.dev/302. [cit. 2024-03-02].

[11] MDN. *HTTP response status codes*. Online. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status [cit. 2024-03-02].

[12] BROADCOM. *Spring Quickstart Guide.* Online. @ 2005-2024. Dostupné z: https://spring.io/quickstart. [cit. 2024-02-11].

[13] BROADCOM. *Spring Initializr*. Online. @ 2005-2024. Dostupné z:

https://start.spring.io/. [cit. 2024-03-23].

[14] TURNQUIST, L. Greg. *Learning Spring Boot 3*, 3rd ed. Packt Publishing Ltd, 2022. ISBN 978-1-80323-330-7.

[15] GOOGLE. *What Is A Relational Database?* Online. Dostupné z: https://cloud.google.com/learn/what-is-a-relational-database. [cit. 2024-03-27].

[16] DULDULAO, B. Devlin, Seiji R. VILLAFRANCA. *Spring Boot and Angular*. *Hands-on full stack web development with Java, Spring, and Angular.* Packt Publishing Ltd., 2022. ISBN: 978-1-80324-321-4.

[17] VISHAMBER, Lal. *Understanding Data Transfer Objects (DTO) in Spring Boo*t. Online. 14.12.2023. Dostupné z: https://medium.com/@vishamberlal/understanding-data-transfer-objects-dto-in-spring-boot-ac06b575a1d5. [cit. 2024-01-12].

[18] BROADCOM. *Building an Application with Spring Boot*. Online. @ 2005-2024. Dostupné z: https://spring.io/guides/gs/spring-boot. [cit. 2024-02-09].

[19] BROADCOM. *Spring Data JPA*. Online. @ 2005-2024. Dostupné z: https://spring.io/projects/spring-data-jpa. [cit. 2024-02-17].

[20] BROADCOM. *Accessing Data with JPA*. Online. @ 2005-2024. Dostupné z: https://spring.io/guides/gs/accessing-data-jpa. [cit. 2024-02-12].

[21] JONATHAN. *Hibernate – OneToOne, OneToMany, ManyToOne and ManyToMany*. Online. 2.4.2020. Dostupné z: https://dev.to/jhonifaber/hibernate-onetoone-onetomany-manytoone-and-manytomany-8ba. [cit. 2024-02-11].

[22] BROADCOM. *Accessing data with MySQL*. Online. @ 2005-2024. Dostupné z: https://spring.io/guides/gs/accessing-data-mysql. [cit. 2024-02-13].

[23] KIMBERLIN, Michael. *Reducing Boilerplate Code with Project Lombok*. Online. 2010. Dostupné z: https://objectcomputing.com/resources/publications/sett/january-2010-reducing-boilerplate-code-with-project-lombok. [cit. 2024-01-25].

[24] BROADCOM. *Spring Security*. Online. @ 2005-2024. Dostupné z: https://spring.io/projects/spring-security. [cit. 2023-11-04].

[25] MICROSOFT. *Authentication vs. authorization*. Online. 20.3.2024. Dostupné z: https://learn.microsoft.com/en-us/entra/identity-platform/authentication-vs-authorization. [cit. 2024-04-22].

[26] FIDAL, Mathew. *Session-Based vs. Token-Based Authentication: Which is better?* Online. 23.12.2023. Dostupné z: https://dev.to/fidalmathew/session-based-vs-token-

based-authentication-which-is-better-227o. [cit. 2024-03-18].

[27] POSTMAN. *What is postman?* Online. @ 2024. Dostupné z: https://www.postman.com. [cit. 2023-10-19].

[28] POSTMAN. *Send parameters and body data with API requests in Postman*. Online. @ 2024. Dostupné z: https://learning.postman.com/docs/sending-requests/create-requests/parameters/. [cit. 2023-10-21].

[29] SANIYA. *What is XAMPP*. Online. 30.5.2023. Dostupné z: https://cloudfoundation.com/blog/what-is-xampp/. [cit. 2023-10-28].

[30] META. *React*. Online. @ 2024. Dostupné z: https://react.dev/. [cit. 2024-01-21].

[31] VIKRAM, Joshi. *Seven Reasons Why A Website's Front-End And Back-End Should Be Kept Separate*. Online. 19.7.2018. Dostupné z: https://www.forbes.com/sites/forbestechcouncil/2018/07/19/seven-reasons-why-a-websites-front-end-and-back-end-should-be-kept-separate/. [cit. 2024-01-27].

[32] GEEKSFORGEEKS. *Difference between Virtual DOM and Real DOM*. Online. Aktualizace 13. října 2023. Dostupné z: https://www.geeksforgeeks.org/difference-between-virtual-dom-and-real-dom/. [cit. 2024-01-21].

[33] VAILSHERY, S. Lionel. *Most used web frameworks among developers, as of 2023.* Online. 19.1.2024. Dostupné z: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. [cit. 2024-03-02].

[34] SCHWARZMÜLLER, Maximilian. *React Key Concepts. Consolidate your knowledge of React's core features.* Packt Publishing Ltd, 2022. ISBN 978-1-80323-450-2.

[35] AUTH0 BY OKTA. *Introduction to JSON Web Tokens*. Online. Dostupné z: https://jwt.io/introduction. [cit. 2024-11-07].

8 Přílohy

Příloha č. 1: zip soubor obsahující zdrojový kód pro klientskou a serverovou část

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Eduard Spousta**

Studium: I2000415

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Vývoj webové aplikace pro správu a evidenci školení**

Název bakalářské práce AJ: Development of web application to manage training records

Cíl, metody, literatura, předpoklady:

Cíl:

Cílem praktické části práce je vytvoření informačního systému pro správu a evidenci školení zaměstnanců. Řešení je zaměřeno na společnosti bez speciálních programů pro personalistiku. Nahradí lokální evidence v tabulkových editorech. Bude splňovat požadavky zaměstnavatelů na evidování zaměstnanců, definovat jednotlivá školení a zaznamenávat jejich provedení. Navržený informační systém bude připomínat nutný termín pro včasné absolvování školení. Řešení bude dostupné online, což umožní rychlý a efektivní přístup k datům.

Osnova:

- Úvod
- Cíl práce
- Teoretická východiska
- Praktická část
- Shrnutí výsledků
- Závěry a doporučení
- Seznam použité literatury

Walls, C. (2016). Spring Boot in Action. Velká Británie: Manning. ISBN: 9781617292545, 1617292540

Cosmina, I., Harrop, R., Schaefer, C., Ho, C. (2017). Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools. Spojené státy americké: Apress. ISBN: 9781484228081, 1484228081

Mardan, A. (2017). React Quickly: Painless Web Apps with React, JSX, Redux, and GraphQL. Spojené státy americké: Manning. ISBN: 9781638353966, 1638353964

Zadávající pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Jaroslav Langer

Datum zadání závěrečné práce: 15.10.2021