Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Databázová aplikace pro řízení vědeckých a příbuzných projektů

Pavel Klečanský

Bakalářská práce
2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel Klečanský**
Osobní číslo:	**I18268**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Databázová aplikace pro řízení vědeckých a příbuzných projektů**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je tvorba webové aplikace umožňující řízení různých vědeckých a příbuzných projektů. V teoretické části budou popsány použité technologie spojené s řešeným tématem bakalářské práce a důvod jejich výběru. Výstupem praktické části bude webová aplikace (frontend+backend), která bude umožňovat plnou správu projektů, včetně správy jejich fází, členů, rozpočtu, úkolů, harmnogramu, institucí, apod. Aplikace bude splňovat požadavky na bezpečný provoz. Součástí práce bude detailní popis tvorby praktické části včetně analýzy zadání, návrhu systému, popis implementace a způsob ověření výsledků. V příloze práce bude uveden postup pro nasazení praktického výstupu včetně uvedení nutných systémových a provozních prostředků pro správný běh aplikace.

Rozsah pracovní zprávy: **40**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

WALLS, Craig. Spring in Action. Fifth Edition. New York: Manning Publications, 2018. ISBN 9781617294945.
SHARMA, Sourabh. Modern API Development with Spring and Spring Boot: Design highly scalable and maintainable APIs with REST, gRPC, GraphQL, and the reactive paradigm. Birmingham: Packt Publishing Limited, 2021. ISBN 9781800562479.

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 29. 04. 2022

Pavel Klečanský

PODĚKOVÁNÍ

Rád bych poděkoval vedoucí této práce, Ing. Monice Borkovcové Ph.D., za vedení práce a její užitečné rady při vypracovávání práce. Dále děkuji také mé rodině a přátelům za podporu po dobu studia.

ANOTACE

Bakalářská práce se zabývá implementací webové aplikace pro řízení různých vědeckých a příbuzných projektů. V teoretické části práce popisuje technologie použité při vývoji serverové a klientské části, do větších detailů je v této části popsán ekosystém okolo Springu. Dále také práce představuje ostatní webové frameworky, které Java nabízí. Praktická část práce se zaobírá návrhem aplikace a poté popisem implementace klientské i serverové části. Součástí praktické části je také popis testování aplikace s představením následných výsledků. Výstupem práce je webová aplikace tvořená klientskou a serverovou částí, která bude umožňovat plnou správu projektů včetně správy jejich fází, členů, rozpočtu, úkolů, institucí apod.

KLÍČOVÁ SLOVA

databázová aplikace, Java, Spring Boot, Spring, Svelte, řízení projektů, PostgreSQL

TITLE

Database application for managing scientific and similar projects

ANNOTATION

This bachelor thesis is focused on implementing web application for managing scientific and similar projects. The theoretical part of the thesis describes the technologies used in the development of the server and client part, the ecosystem around Spring is described in more detail in this part. Furthermore, the work presents other web frameworks that Java offers. The practical part of the thesis deals with the design of the application and then the description of the implementation of the client and server part. Part of the practical part is also a description of testing the application with the presentation of subsequent results. The output of the thesis is a web application consisting of client and server part, which will allow full management of projects, including management of their phases, members, budget, tasks, institutions, etc.

KEYWORDS

database application, Java, Spring Boot, Spring, Svelte, project management, PostgreSQL

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

JPA	Java Persistence API
API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
SQL	Structured Query Language
J2EE	Java 2 Enterprise Edition
JVM	Java Virtual Machine
MVC	Model-View-Controller
JAX-RS	Jakarta RESTful Web Services
RDMBS	Relational Database Management Systems¨
JSR	Java Community Process
SPA	Single-page application
POJO	Plain old Java object

ÚVOD

Téměř všechny pracovní činnosti v dnešní době jsou rozděleny do projektů ať už malých či velkých. Tyto projekty zřídkakdy tvoří pouze jediná osoba, a proto může být nepoužívání jakýchkoliv nástrojů pro řízení projektů při plnění úkolů velmi neefektivní. Proto se očekává, že při plánování a provádění projektu bude použit nástroj pro řízení projektu jako například Jira nebo Trello. Ale na rozdíl od projektů, na které jsou zmíněné nástroje navržené, potřebují vědecké projekty specifické funkce, které tyto nástroje nepodporují.

Cílem bakalářské práce je vytvořit funkční webovou aplikaci pro řízení různých vědeckých a příbuzných projektů. Aplikace bude umožňovat komplexní správu projektů a jejich úkolů. Kromě správy samotných projektů bude aplikace poskytovat možnosti správy fází, rozpočtu, harmonogramu, správu týmů nebo správu výstupů. Webová aplikace bude rozdělena na dvě části, a to klientskou a serverovou část. Serverová část bude naprogramována pomocí jazyka Java a běží nad platformou Spring, která zprostředkovává komunikaci s relační částí PostgreSQL za pomoci Hibernate a poskytuje koncové body v podobě REST API. Klientská část bude naprogramována za pomocí nástroje Svelte, který komunikuje se serverovou částí za pomoci již zmíněného REST API. Aplikace nelze využít pro nepřihlášeného uživatele, pro používání aplikace se musí uživatel přihlásit nebo registrovat. Při registraci musí uživatel zadat sedmimístný kód, který generuje administrátor. Po přihlášení se uživatel dostane ke svým projektům.

První blok psané práce se zabývá teoretickým popisem technologií použitých při vývoji aplikace. Vzhledem k architektuře výstupní aplikace bude velký důraz kladen na je v této části kladen na popis technologií v Spring ekosystému, který hraje klíčovou roli v implementaci serverové části aplikace. Nebude chybět ani popis ostatních webových frameworků postavených na JVM, které by mohly být použity místo Springu. Kromě technologií na serverové straně aplikace práce bude obsahovat i přehled technologií využitých při tvorbě klientské aplikace.

Stěžejním blokem práce je praktická část, kde bude v první kapitole představena analýza celého projektu od funkčních a nefunkčních požadavků až po návrh řešení. Následující dvě kapitoly budou popisovat samotnou implementaci – jak serverovou část, tak i klientskou. Poslední kapitola bude zaměřena na testování aplikace a představuje testovací scénáře, které mohou být pro testování použity.

1 SPRING

1.1 Historie Spring Frameworku

Spring Framework vznikl na základě knihy Expert One-on-One: J2EE Design and Development od Roda Johnsona. Hlavním důvodem vzniku Spring Frameworku byla složitost raných specifikací J2EE. V březnu roku 2004 vyšla první hlavní verze Springu, a to verze 1.0, která už v sobě obsahovala základní kámen celého Springu, dependency injection. Od vydání první verze se Spring vyvíjel velkou rychlostí a za pouhé dva roky vyšla verze 2.0, která přinesla řadu funkcí jako například rozšiřitelnou konfiguraci pomocí XML, podporu pro Javu 5 či podporu pro dynamický jazyk Groovy. Verze 3.0 vyšla v prosinci roku 2009 a přinesla s sebou další nové funkce. Například podporu pro vestavěné databáze jako jsou HSQL nebo H2 a také podporu pro Java EE 6. Přesně o čtyři roky později od třetí verze byla vypuštěna verze čtvrtá, které přinesla řadu velkých změn, a to podporu pro Java 8, Java EE 7 a WebSockety. Aktuální verze Spring Frameworku je 5.3.15 a na červenec roku 2022 je plánované vydaní verze 6. [39]

1.2 Co je vlastně Spring ?

Každá netriviální aplikace je sestavená z mnoha komponent, kde každý jednotlivý komponent zastává specifickou funkci a společně vytvářejí určitou činnost. Když se tato aplikace spustí, musí být jednotlivé komponenty nějakým způsobem vytvořeny a spojeny. A toto přesně ve své podstatě Spring umožňuje. Spring nabízí kontejner, často označovaný jako Spring application context, který vytváří a spravuje komponenty v aplikaci. Tyto komponenty, téže označované jako beans, jsou v kontejneru propojeny dohromady a tím vytvářejí aplikaci. [1][2]

Samotné propojování komponent dohromady je založené na návrhovém vzoru známém jako dependency injection. Raději než vytvářet a spravovat životný cyklus komponentů ručně, závisí aplikace s dependency injection na kontejneru, který vytváří a spravuje všechny komponenty a vkládá je do jednotlivých beans které je potřebují. Tato činnost je typicky prováděna přes argumenty konstruktoru objektů. Kromě těchto základních funkcí nabízí Spring celé portfolio souvisejících knihoven, webový framework, různé možnosti perzistence dat, security framework, integrace s jinými systémy, sledování běhu systému, podporu mikroslužeb, reaktivní programovací model a mnoho dalších funkcí. Některé tyto funkce jsou v další části představeny. [1]

1.3 Moduly Spring Frameworku

Spring Framework je rozdělen na několik odlišných modulů pro konkrétní sadu funkcionalit, které pracují více či méně nezávisle na ostatních. Díky tomu je systém velmi flexibilní a umožnuje vývojářům vybrat pouze ty moduly, jež jsou vyžadovány pro aplikaci. Kupříkladu může vývojář použít pouze modul Spring Web a na zbytek aplikace použít knihovny které nepatří do Spring ekosystému. [41]

1.3.1 Core

Core modul představuje elementární část celého Spring Frameworku. Na tomto modulu jsou závislé všechny ostatní moduly. Spring Core poskytuje řadu různých funkcí jako například podporu pro internacionalizaci pomocí i18n, validaci nebo typovou konverzi. Ale hlavní vlastností Spring Core je dependency injection, díky které Spring řídí vytváření, konfigurování a spravování jednotlivých komponentů. [40][41]

1.3.2 Data Access

V dnešní době, kdy se internet stal neodmyslitelnou součástí každodenního života, téměř všechny aplikace nějakým způsobem interagují s daty. Data jsou jednou z nejdůležitějších součástí aplikace a čím jednodušší je vytvářet aplikace, které mohou pracovat s daty, tím lepší. Toto má přesně za úkol Spring Data Access modul. Díky tomuto modulu není potřeba psát tradiční JDBC kód, na který můžeme být zvyklí z jiných frameworků, ale máme k dispozici komponenty, vytvořené za účelem odstranění nutnosti manipulace s databázovým připojením a dalšími věcmi přímo v našem kódu. Ale to neznamená že kód, který uvolňuje prostředky, není potřeba. Jde jen o to, že ho vývojář nemusí sám ručně psát, protože už je napsán uvnitř tříd, které framework poskytuje. [40]

1.3.3 Integration

Téměř žádná aplikace nežije v izolaci, a proto potřebuje nějaký způsob, jak sdílet data nebo pracovat s ostatními aplikacemi. Výhradně k této činnosti byl vytvořen Spring Integration. Spring Integration umožňuje komunikovat s jinými systémy, kupříkladu pomocí webových služeb, Java Message Service nebo Java Management Extensions. [40]

1.3.4 Web

Spring Web je framework, který poskytuje možnost zpracovávat webové požadavky. Spring umožnuje tyto požadavky zpracovávat dvěma způsoby, a to buď pomocí Spring Web MVC, nebo Spring WebFlux. Spring Web MVC je originální webový framework, který byl se

Springem od jeho vytvoření a je postavený nad Servlet API. Spring WebFlux je nový moderní webový framework přidaný do Spring Frameworku od verze 5.0, který používá reaktivní programování a díky tomu dokáže zpracovávat více požadavků než původní framework. [40]

1.3.5 Testing

Testování je nedílnou součástí vývoje podnikového softwaru. Proto není divu, že se Spring Framework snaží ulehčit programátorům práci v této oblasti. Spring Framework jako takový se zaměřuje na dva druhy testování, a to na unit testy a integrační testy. Speciálních funkcionalit Spring nenabízí velké množství. Je to z důvodu jeho postavení nad dependency injection. Tím umožňuje do jednotlivých komponentů v testech jednoduše vkládat mock objekty a díky tomu řádově usnadňuje testování. Kromě explicitní podpory pro integrační testy Spring nabízí testování webových kontrolerů nebo testování přístupu k datům a mnoho dalších funkcí. [40]

1.3.6 AOP

Aspektově orientované programování doplňuje objektově orientované programování tím, že poskytuje další způsob uvažování o struktuře programu. Klíčovou jednotkou modularity v OOP je třída, zatímco v AOP je jednotkou modularity aspekt. AOP pomáhá přidáním dalšího chování k již existujícímu kódu, aniž by se upravoval samotný kód. Ve Springu se AOP může používat k mnoha různým činnostem, jako například logování, bezpečnost, caching a mnoho dalšího. [40][41]

1.4 Projekty Springu

Spring není pouze Spring Framework, jde o celý ekosystém pokrývající různé projekty, které pomáhají vývojářům tvořit aplikace.

1.4.1 Spring Boot

Spring Boot je pouze rozšířením již existujícího a rozsáhlého Spring Frameworku, má ale některé specifické funkce, které usnadňují vytváření aplikací ve Spring ekosystému. Toto rozšíření zahrnují například předkonfigurované balíčky závislostí, které pokrývají typické případy užití jako například balíček Spring Web umožňující vytváření webových aplikací založených na Springu s minimální konfigurací. Další výhodou Spring Boot je, že pomáhá vytvářet vývojářům aplikace, které jak se říká, prostě běží. Konkrétně umožňuje vytvářet samostatné aplikace které běží samy o sobě, aniž by se spoléhaly na externí webové servery, a to pomocí vložení webového serveru, jako například Tomcat nebo Netty, přímo do aplikace.

Díky těmto vlastnostem je Spring Boot nejen ideálním nástrojem pro tvorbu mikroslužeb, ale i standartních monolitických aplikacích. [34][35]

1.4.2 Spring Cloud

Spring Cloud poskytuje vývojářům nástroje pro rychlé vytváření některých běžných mechanismů v distribuovaných systémech. Jde například o configuration management, service discovery či intelligent routing. Koordinace distribuovaných systému vede často k používání již známých a ověřených nástrojů a pomocí Spring Cloud mohou vývojáři rychle postavit služby a aplikace, které implementují tyto nástroje. Takové řešení fungují dobře v jakémkoli distribuovaném prostředí, datových centrech na fyzických zařízeních nebo na spravovaných platformách, jako je Cloud Foundry. [36]

1.4.3 Spring Security

Spring Security je framework, který se zaměřuje na poskytnutí mechanismů pro autentifikaci a autorizaci v Spring aplikacích. Vývoj byl zahájen v roce 2003 jako open source projekt pod názvem „Acegi Security“ než byl oficiálně zařazen do Spring projektů. Kromě již zmíněné autentifikace a autorizace lze Spring Security nakonfigurovat tak, aby chránila aplikaci před běžnými útoky jako je například CSRF, XSS, útok hrubou silou nebo Man in the middle útok. Jednou z nejlepších věcí, kterou může člověk udělat, pokud nemá zkušenosti se zabezpečováním aplikace, je zjistit si, zda platforma, kterou používá, má nějaký security framework. A přesně to Spring má v podobě Spring Security, tudíž při vytváření aplikací za pomocí Springu, je tento framework ideální možností, jelikož je spolehlivý, používaný velkým množstvím aplikací a má skvělou integraci s naší aplikací. [2][33]

1.4.4 Další

Kromě již zmíněných projektů existuje ještě mnoho dalších projektů, které pomáhají vývojářům vytvářet nové aplikace pomocí Spring ekosystému. Jedním z takových projektů je Spring for GraphQL poskytující podporu Spring aplikacím k vytváření GraphQL API. Dalším podstatným projektem je Spring Data, který poskytuje známý programovací model založený na Springu pro přístup k datům. Spring má pod sebou více než dvacet projektů, zde jsou představeny jen ty nejpoužívanější. [37][38]

2 DALŠÍ FRAMEWORKY POSTAVENÉ NAD JAVOU

Tato část seznamuje čtenáře s dalšími JVM webovými frameworky, které mohou být použity jako náhrada za Spring.

2.1 Quarkus

Quarkus je full-stack, Kubernetes-native Java framework vytvořený pro JVM. Tento framework je speciálně optimalizovaný pro kontejnerizaci a díky tomu umožnuje být efektivní platformou pro serverless, cloud nebo Kubernetes prostředí. Quarkus je navržen tak, aby pracoval s populárními Java standarty, frameworky a knihovnami, jako je například Eclipse MicroProfile, Spring, Apache Kafka, JAX-RS, JPA, Camel a mnoho dalších. Framework byl vytvořen společností Red Hat s cílem jednoduchého používání a s funkčností s malou nebo žádnou konfigurací. [3]

2.2 Play

Play je webový framework pro Javu a Scalu poskytující komponenty a API pro moderní vývoj webových aplikací. Jednou z velkých výhod Play framework pro výkonově náročné aplikace je, že stojí nad platformou Akka, která používá actor model pro vícevláknové programování. Díky tomu je Play ideální webový framework pro psaní vysoce distribuovaných systému. [6][7]

2.3 Micronaut Framework

Micronaut je aplikační framework pro JVM zaměřený primárně na tvorbu mikroslužeb a cloud-native aplikací. Framework si bere inspiraci z tradičních frameworků jako Spring nebo Grails. Tím nabízí vývojářům známé prostředí pro vývoj, ale s kratšími startovacími časy aplikace a menší spotřebou paměti. Díky tomu je možné používat Micronaut na místech, kde by tradiční frameworky nemusely být tím nejlepším řešením pro tvorbu, jako například serverless funkce, IOT zařízení nebo CLI aplikace. [5]

2.4 Javalin

Javalin je na rozdíl od ostatních zmíněných frameworků velmi minimalistický. Díky této vlastnosti by ho mnoho programátorů mohla považovat spíše za knihovnu než framework. Javalin podporuje programování jak v jazyce Java, tak i v jazyce Kotlin. Jeho hlavním cílem je jednoduchost, skvělý vývojářský zážitek a prvotřídní interoperabilita mezi Javou a Kotlinem. Hlavním rozdílem Javalinu oproti ostatním zmíněným frameworkům je, že nepoužívá pro nastavení aplikace žádné anotace, žádnou reflexi a nepotřebuje žádnou speciální strukturu aplikace. [4]

```
public class HelloWorld {
    public static void main(String[] args) {
        Javalin app = Javalin.create().start(7000);
        app.get("/", ctx -> ctx.result("Hello World"));
    }
}
```

Zdrojový kód 1: Hello World aplikace pomocí Javalin (zdroj vlastní)

3 POUŽITÉ TECHNOLOGIE SPOLU S SPRING FRAMEWORKEM

3.1 Java

Název Java se běžně používá k označení platformy Java, což je sada nástrojů umožňujících snadný vývoj aplikací napříč platformami. Také je Javou označován programovací jazyk, který se využívá k vývoji programů pro tuto platformu. Jednou z hlavních věcí, která odlišuje Javu jako platformu od mnoha jiných technologií, je to, že je navržena tak, aby kód napsaný nad touto platformou mohl být spuštěn na jakémkoli systému, na kterém může běžet Java Virtual Machine. Tento koncept „Write once, run anywhere“ byl používán jako slogan k propagaci multiplatformních schopností Javy. [32]

Samotný programovací jazyk Java je objektově orientovaný a syntakticky velmi podobný C++. Ale na rozdíl od některých jiných jazyků, které obsahují třídy, ale nevyžadují jejich používání, jsou programy v Javě vždy navrhovány s objektově orientovaným designem. I když jsou jazyk Java a Java Virtual Machine úzce propojeny, je Java Virtual Machine navržená takovým způsobem, že je v ní možné spouštět kód i z jiných jazyků navržených přímo pro ni, a to díky takzvanému bajtkódu, do kterého jsou jednotlivé jazyky přeloženy a ten je poté spuštěn na samotném virtuálním stroji. Takovými jazyky jsou například Groovy, Scala, Kotlin nebo JRuby. Když už se zde zmiňuji o různých jazycích, které běží nad JVM, mohlo by leckoho napadnout, že do této kategorie spadá i JavaScript. Tak to ale vůbec není. JavaScript s Javou kromě části jména nemá mnoho společného. [32]

Java je jedním z nejpoužívanějších programovacích jazyků na světě, a proto ji můžeme najít na mnoha místech. Android je operační systém, který pohání miliony mobilních zařízení po celém světě. Používá jazyk spolu s vlastní sadou knihoven jako základ pro mobilní aplikace vytvořené pro tuto platformu. Javu můžeme najít i na našem desktopovém počítači. Dokonce jedna z nejpopulárnějších světových her Minecraft je napsaná pomocí Javy. I když se Java v dnešní době nevyskytuje běžně jako frontend pro webové aplikace, stala se velmi populárním jazykem pro psaní backendu pomocí frameworků jako je Spring nebo aplikačních serverů, jako například WildFly nebo Tomcat. [32]

3.2 Lombok

Jednou z často vytýkaných vlastností Javy je, že je takzvaně velmi upovídaná. To znamená, že i pro velmi jednoduchý úkol napíšete mnoho řádků kódu. Navíc Java zahrnuje velké množství opakujícího se kódu, jako jsou například gettery a settery, což vede k obrovskému množství

boilerplate kódu[1], který nepřidává žádnou hodnotu byznysové části aplikace, a přesně tyto problémy pomáhá Lombok řešit. [8]

Lombok je knihovna, která poskytuje několik anotací, jejichž cílem je vyhnout se psaní opakujícího se kódu a boilerplate kódu. Pouhým přidáním několika anotací se kupříkladu můžeme vyhnout psaní defaultního konstruktoru, metod toString(), equals() a hashCode(). Všechny tyto akce probíhají za překladu, kdy knihovna vloží bytecode, který představuje požadovaný kód do .class souboru. Díky tomu že je Lombok podporován ve všech hlavních vývojových prostředí pro Javu jako například Eclipse, IntelliJ IDEA a NetBeans má programátor stejný vývojářský zážitek jako kdyby kód psal sám. [8]

3.3 IntelliJ IDEA

IntelliJ IDEA je vývojové prostředí pro JVM jazyky navržený tak, aby maximalizovalo produktivitu vývojářů. Jednou z hlavních výhod, které vývojové prostředí umožnuje je, že provádí rutinní a opakující se úkoly za nás. Například vývojářům poskytuje chytré doplňování kódu, statickou analýzu kódu a jednoduché refaktorování a tím umožňuje soustředit se na psaní kvalitního a čistého kódu. IntelliJ je multiplatformní a poskytuje stejný zážitek na Windows, macOS nebo Linuxu. [27]

První verze byla vydána v lednu roku 2001 firmou JetBrains. Bylo to první vývojové prostředí pro Javu, které obsahovalo pokročilou navigaci v kódu a možnost kód refaktorovat. V roce 2010 bylo IntelliJ vyhlášeno nejlepším programovacím nástrojem dostupným pro Javu. A o čtyři roky později vydal Google vlastní open source vývojové prostředí pro tvorbu Android aplikací postavené nad IntelliJ IDEA. [28]

3.4 Gradle

Gradle je nástroj pro automatizaci sestavování programů stejně jako Maven nebo Ant. Umožňuje psaní build skriptů, které můžou běžet v jakémkoli prostředí na náš požadavek. Gradle čerpá z poznatků získaných z jiných oblíbených nástrojů a posouvá jejich nápady na další úroveň. Schopnost Gradlu spravovat závislosti není omezena pouze na externí knihovny. Gradle poskytuje podporu pro definování a organizování více projektů a také modelování závislostí mezi nimi. Tato vlastnost je velmi důležitá primárně pro velké projekty schopné pojmout až stovky dílčích projektů, které spolu musí být propojeny. [24]

[1] kód, který se opakuje na více místech s malou nebo žádnou změnou

Build skripty nejsou v Gradlu psány pomocí XML jako u většiny nástrojů pro Javu, ale pomocí Groovy nebo Kotlin DSL. Díky tomu jsou skripty čitelnější a mají mnohem menší velikost. Kromě Javy podporuje Gradle i další jazyky, a to například Kotlin, Scala, C/C++ nebo Groovy. Gradle je také úzce propojen s ostatními vývojářskými nástroji jako Eclipse, IntelliJ, NetBeans nebo Visual Studio Code.[25][26]

3.5 JPA

JPA neboli Java Persistence API je framework založení na POJO určení k ukládání a získávání objektů z databáze. JPA sám o sobě nemůže provádět žádnou ze svých činností, protože je pouze specifikací. Pokud chceme JPA použít v nějaké z našich aplikací, musíme využít nějaké z implementací tohoto standartu. Jednou z nejznámějších a nejpoužívanějších je Hibernate, ale můžeme nalézt i další jako například EclipseLink, DataNucleus nebo OpenJPA. [17]

JPA se poprvé objevil v Java EE 5 v roce 2006 jako součást JSR 220. Další verze na sebe nenechala dlouho čekat a o tři roky později vyšlo JPA 2.0, které obsahovalo mnoho nových funkcí, primárně ty, které byly nejvíce žádané komunitou. Mezi ně patřily například další možnosti mapovaní, rozšíření Java Persistence Query Language nebo přidání Java Criteria API, jenž umožňovalo vytváření vlastních dynamických dotazů v programu. V roce 2013 vyšla verze 2.1, která přidala několik exotičtějších funkcí, jako například podporu pro uložené procedury nebo mapování konvertorů. Verze 2.2 je nejmenší verze oproti ostatním zmíněným a neobsahuje velké množství nových funkcí. Jedna z hlavních funkcí v této verzi, usnadňující života mnoha programátorů, je podpora Java 8 tříd pro datum a čas. Poslední verze JPA je verze 3.0, která nepřinesla žádné nové funkce, přišla však s přejmenováním na Jakarta Persistence 3.0 a s tím spojeným přejmenováním balíčků z javax.persistence na jakarta.persistence. [17]

3.5.1 Hibernate

Hibernate je open source nástroj umožňující objektově relační mapování pro Javu. Poskytuje framework pro mapování objektově orientovaných doménových modelů do tradičních relačních databází. Hibernate neposkytuje pouze mapování Java tříd do tabulek v databázi, nabízí také možnosti dotazování a načítání dat, čímž může výrazně zkrátit dobu vývoje jinak strávenou ručním zpracováním dat v SQL a JDBC. [18]

Vývoj Hibernate odstartoval v roce 2001 Gavin King jako alternativu pro EJB2-style entity beans. Hlavním cílem bylo nabídnout lepší schopnosti persistence, než v té době nabízel EJB2, dále také zmenšit komplexitu a přidat chybějící funkce. Hibernate používal své vlastní mapovací soubory, které byly psány pomocí XML. Od verze Java 5, kde byly do jazyka přidány

anotace, začal Hibernate pro mapování primárně právě je. Ve verzi Hibernate 3.5, vydané v roce 2010, začal být Hibernate plně kompatibilní s JPA 2.0. Poslední stabilní verzí Hibernate je 5.6, která je plně kompatibilní s JPA 2.2 a novou specifikací Jakarta JPA 3.0. [18] [19][20]

Pokud někdo zmiňuje Hibernate, myslí ve většině případech projekt Hibernate ORM, avšak pod záštitou slova Hibernate se skrývá mnohem více projektů, než je tento. Hibernate Search, který poskytuje fulltextové vyhledávaní pro Hibernate, podporuje Apache Lucene, Elesticsearch server nebo OpenSearch server, Hibernate Validator poskytující validaci pro aplikace, který je také referenční implementací Jakarta Bean Validation, Hibernate Reactive, který u vybraných databází, jako PostgresSQL, MySQL, nebo Db2 umožňuje reaktivní komunikaci s databází, nebo Hibernate OGM, který odemyká možnost používat JPA pro NoSQL databáze, v současné době podporuje tři databáze a to Infinispan, MongoDB a Neo4j. [2][21][22][23]

3.6 PostgreSQL

PostgreSQL je open source databáze s dobrou pověstí díky své spolehlivosti, flexibilitě a podpoře technologických standardů. Na rozdíl od jiných RDMBS PostgreSQL podporuje jak nerelační, tak relační datové typy. To z ní dělá jednu z nejuniverzálnějších, stabilních a vyzrálých relačních databází, které jsou dnes k dispozici. [15]

PostgreSQL byl původně vyvinut v roce 1986 jako pokračování relační databáze INGRES. Původní název PostgreSQL byl POSTGRES, jelikož ve svých začátcích projekt nepodporoval SQL, ale používal vlastní dotazovací jazyk POSTQUEL, který byl inspirován jazykem QUEL používaným v již zmíněné databázi INGRES. V roce 1994 byl nahrazen POSTQUEL dotazovacím jazykem SQL, a tím vznikla nová verze s názvem Postgres95. O dva roky později se projekt znovu přejmenoval na PostgreSQL, který se používá dodnes. [15] [16]

Jak už bylo psáno na začátku, PostgreSQL je velmi flexibilní databáze. To také dokazuje to, že je v ní možné psát uživatelsky definované funkce i v jiných jazycích, než je SQL nebo C. PostgreSQL oficiálně podporuje čtyři různé jazyky, a to PL/pgSQL, PL/Tcl, PL/Perl, PL/Python. Díky rozsáhlé komunitě dobrovolníků okolo tohoto projektu vznikla i další uživatelská podpora pro jiné jazyky než oficiální, a to například Java, TypeScript, Kotlin a mnoho dalších. [16]

3.7 Tailwind CSS

Tailwind je utility-first CSS framework speciálně navržený pro rychlé vytváření vlastních uživatelských rozhraní. Na rozdíl od ostatních CSS frameworků, jako je například Bootstrap nebo Bulma, nemá Tailwind žádné předdefinované komponenty. Místo toho funguje na nižší úrovni a poskytuje sadu pomocných tříd, jejichž prostřednictvím můžeme rychle a snadno vytvořit vlastní design. Díky tomu mají designéři větší kreativní volnost a tím mohou vytvářet více originální stránky, které nejsou vůbec podobné ostatním stránkám vytvořením za pomoci stejného frameworku. [9][10]

Přidání Tailwindu do projektu není tak jednoduché jako například u Bootstrapu, kde stačí přidat odkaz na CSS soubor do HTML. Se správným návodem to ale není nikterak složité. Ve složce s projektem se použije příkaz **`npm install -D tailwindcss`**, který přidá závislost na Tailwind, a poté je nutné přidat konfigurační soubor příkazem **`npx tailwindcss init`**.

```
module.exports = {
  content: ["./src/**/*.{html,js}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Zdrojový kód 2: Nastavení konfiguračního souboru pro Tailwind (zdroj vlastní)

Dále v konfiguračním souboru, je potřeba přidat cestu ke všem šablonovým souborům, jak je prezentováno v ukázce zdrojových kódů číslo dva. Poté je vytvořen CSS soubor, kam se importuje Tailwind, jak je předvedeno v ukázce zdrojových kódu číslo tři.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Zdrojový kód 3: Nastavení Tailwind v CSS souboru (zdroj vlastní)

Na závěr postačí spustit tento **`npx tailwindcss -i ./src/input.css -o ./dist/output.css –watch`** příkaz, monitorující šablonové soubory a vytvoří díky nim CSS soubor, na nějž lze odkazovat z HTML souboru, ve kterém je nyní možné používat všechny pomocné třídy nabízené Tailwindem.

3.8 TypeScript

TypeScript je objektově orientovaný programovací jazyk vytvořený společností Microsoft. Pro zjednodušení si lze představit TypeScript jako takový obal nad JavaScriptem, který přidává nové funkcionality. Díky tomu, že je TypeScript pouze obal, jsou veškeré kódy JavaScriptu validními kódy i v TypeScriptu. Z tohoto důvodu je možné ho přidat do všech JavaScriptích projektů bez jakýchkoliv problémů. Jednou z hlavních funkcionalit, kterou TypeScript nabízí, je volitelné statické typování, díky čemuž se stal oblíbeným jazykem pro psaní JavaScriptu ve velkých projektech. [13]

Vytvoření nového TypeScript projektu není nijak složité. Stačí vložit pár příkazů do příkazové řádky a nový projekt je vytvořen. Jako první krok pro instalaci TypeScriptu je zapotřebí přidat do projektu závislost na TypeScript, která se provede příkazem **`npm i typescript --save-dev`**. Jako další krok je nutné inicializovat TypeScript projekt příkazem **`npx tsc –init`**, který přidá do projektu přednastavený konfigurační soubor v názvem tsconfig.json. Nyní už je v projektu všechno potřebné a je možné začít TypeScript používat. Pro převedení TypeScript kódu do JavaScriptu, který lze spustit v prohlížeči nebo v Node.js, stačí použít příkaz **`npx tsc index.ts`**, který daný TypeScriptový soubor zkompiluje nebo můžeme také použít příkaz **`npx tsc -w index.ts`**, který bude sledovat vložený soubor a při jakékoliv změně daný soubor zkompiluje. [14]

3.9 Svelte

Svelte je frontendový nástroj na tvoření webových aplikací. Svelte má velmi odlišný přístup k vytváření webových aplikací od jiných nástrojů jako například React nebo Vue, které vykonávají většinu své práce v prohlížeči za běhu aplikace. Oproti tomu Svelte tyto kroky přesouvá do překladu, který probíhá při sestavování aplikace, a díky tomu se vytváří vysoce optimalizovaná aplikace, kde běží pouze čistý JavaScript. Díky tomuto řešení nevzniká pouze velikostně menší aplikace s lepším výkonem, ale také lepší zážitek pro vývojáře, kteří mají menší zkušenost s ekosystémem moderních nástrojů okolo JavaScriptu. [12]

Vytvoření nového projektu se Svelte je jednoduché. Stačí použít příkaz **`npx degit sveltejs/template my-new-project`**, stahující šablonu, která je poskytnuta vývojáři Svelte. Poté co se projekt z šablony vytvoří, stačí nainstalovat potřebné balíčky pomocí příkazu **`npm install`** a po instalaci je možné spustit aplikaci příkazem **`npm run dev`**, který přeloží Svelte aplikaci a spustí jí na localhostu defaultně na portu 8080. [12]

Svelte je komponentový framework, ve kterém se jednotlivé stránky i celá aplikace skládají z komponentů. Jednotlivé komponenty jsou psány do souboru s koncovkou .svelte pomocí nadstavby HTML. Svelte soubor je rozdělen do tří volitelných částí, a to do skriptovací části, ve které se zapisuje JavaScript, stylové části, kde jsou uložené CSS styly a jako poslední do značkovací části, ve které je napsané samotné HTML. [12]

```
<script>
    // do této části se píše JavaScript
</script>

<h1>HTML</h1><!-- do této části píšeme HTML -->

<style>
    /* do této části píšeme CSS styly */
</style>
```

Zdrojový kód 4: Ukázka rozložení Svelte komponentu (zdroj vlastní)

3.9.1 Routify

Routify je nástroj pro Svelte umožňující. vytvářet cesty založené na souborovém systému. Například pokud dojde k vytvoření souboru index.svelte, jeho cesta bude / ,nebo pokud je vytvořen soubor pes.svelte jeho cesta bude /pes. Při tvorbě aplikace jsou dvě možnosti, jak definovat cesty, a to buď pomocí konfiguračního souboru, nebo pomocí souborového systému. Každá má své výhody a nevýhody, ale při větších projektech se může stávat, že je konfigurační soubor nepřehledným a tím může stěžovat práci při hledání chyb. [29]

Vytvoření nové aplikace, která využívá Routify, je velmi jednoduché. Stačí použít příkaz **`npx @roxi/routify init`**. Tento příkaz nainstaluje startovací šablonu pro Routify. Šablona obsahuje nakonfigurované Routify pro okamžité použití a také složku example obsahující ukázky použití které se mohou hodit primárně pro začátečníky s tímto nástrojem. Pokud je zapotřebí nový projekt spustit, stačí použít příkaz **`npm run dev`**, který spustí aplikaci na localhostu primárně s portem 5000. [30]

3.10 Vite

Vite je nástroj pro sestavování frontendu. Jeho hlavním cílem je poskytovat rychlejší a jednodušší vývojářský zážitek pro moderní webové projekty. Skládá se ze dvou hlavních částí. První část je dev server, který poskytuje vylepšení funkcí oproti nativním ES modulům, jako

například extrémně rychlý Hot Module Replacement, což umožňuje vyměnit moduly za běhu aplikace. Druhou důležitou částí Vite je sestavovací příkaz, který z našeho kódu pomocí Rollup udělá optimalizovaný balíček a ten lze poté nasadit na produkci. [31]

Začít používat Vite v novém projektu je velmi jednoduché. Stačí použít příkaz **`npm init vite@latest`**. Po spuštění příkaz položí několik dotazů na nastavení – například jméno projektu nebo jaký frontendový framework je žádoucí použít. Vite v tomto případě nabízí například čistý JavaScript, Vue, React, Svelte a další. Po dokončení příkazu je nutné ještě nainstalovat jednotlivé závislosti pomocí příkazu **`npm install`**. Nyní už je projekt plně připraven a lze začít vyvíjet. [31]

4 ANALÝZA PROJEKTU

4.1 Funkční a nefunkční požadavky

Požadavky, které jsou dobře promyšlené a jasně zdokumentované, jsou základem každého úspěšného projektu. Existují dva hlavní typy požadavků, jenž by měly být vytvořeny při práci na projektu, a to funkční a nefunkční požadavky. Pochopení rozdílu mezi těmito dvěma skupinami pomáhá zajistit, že vývojáři dodají produkt, který bude fungovat podle očekávání. Výzkum ukazuje, že 68 % projektu v IT selže. Jedním z hlavních důvodů je špatná nebo žádná definice požadavků na začátku projektu. Pochopení rozdílu mezi funkčními a nefunkčními požadavky pomáhá tomuto předejít. [42]

4.1.1 Funkční požadavky

Funkční požadavky představují „Co musí systém dělat“. Pokud systém nesplňuje funkční požadavek, pak selže. To z důvodu toho, že systém nebude schopen provádět činnosti ke správnému fungování. [42]

Tabulka 1: Funkční požadavky (zdroj vlastní)

F1	Správa projektů	Aplikace bude umožňovat uživateli vytvářet nové projekty a upravovat nebo odstraňovat projekty ve kterých je členem.
F2	Viditelnost projektů	Uživateli budou viditelné pouze projekty, do kterých je zařazen, nebo do kterých je zařazena skupina, do které patří.
F3	Správa úkolů	Aplikace bude umožňovat členovy projektu vytvářet, upravovat a odstraňovat jednotlivé úkoly projektu.
F4	Vlastnosti úkolu	Aplikace bude umožňovat členovy projektu měnit vlastnosti úkolu v projektu.
F5	Role uživatelů	Aplikace bude implementovat uživatelské role super administrátor, administrátor a uživatel.

F6	Autentizace	Uživatel se bude přihlašovat do aplikace za pomoci emailu a hesla.
F7	Hesla	Uživateli bude umožněna změna heslo po přihlášení.
F8	Organizace	Aplikace bude umožňovat administrátorovi vytvářet, upravovat a odstraňovat projekty.
F9	Skupiny	Aplikace bude umožňovat administrátorovi vytvářet, upravovat a odstraňovat skupiny.
F10	Uživatel ve skupinách	Aplikace bude umožňovat přidávat uživatele do skupin a tím zviditelňovat jednotlivé projekty.
F11	Kontakty	Aplikace bude umožňovat uživatelům prohlížet kontakty na jiné uživatele nebo organizace.
F12	Komentáře	Aplikace bude umožňovat uživatelům přidávat komentáře pod jednotlivé projekty nebo úkoly.
F13	Soubory	Aplikace bude umožňovat uživatelům přidávat k jednotlivým projektů soubory.
F14	Design	Aplikace bude responzivní a uživatelsky přívětivá.

4.1.2 Nefunkční požadavky

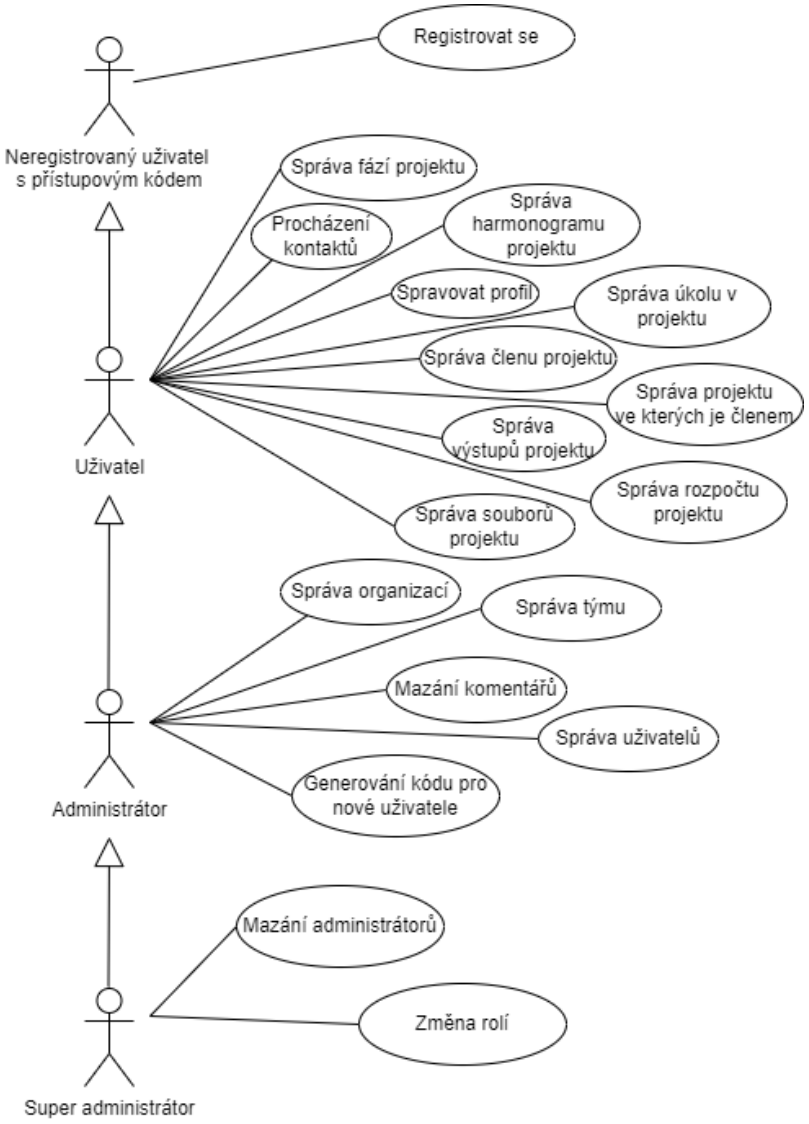
Nefunkční požadavky představují „Jak systém funguje“. Nefunkční požadavky se zaměřují na to, jak systém poskytuje konkrétní funkci. Na první pohled mohou být považovány za méně důležité než funkční požadavky, ale obě skupiny požadav ků mají svou opodstatněnou roli v dobrém systému. [42]

Tabulka 2: Nefunkční požadavky (zdroj vlastní)

N1	Platforma	Aplikace bude vyvíjena nad platformou Spring.

N2	Databáze	Aplikace bude využívat relační část databáze PostgreSQL.
N3	Přístup do databáze	Přístup do databáze bude zprostředkován za pomoci JPA.
N4	Klientská část	Klientská část aplikace bude vyvíjena pomocí nástroje Svelte.
N5	Styly	Styl klientské části bude vytvořen pomocí Tailwind CSS.
N6	SPA	Klientská část bude implementována jako Single-page aplikace.

4.2 Diagram případů užití



Obrázek 1: UML Diagram případů užití (zdroj vlastní)

Diagram případu užití je nástroj, který mapuje interakce mezi uživateli a systémem a ukazuje interakce mezi nimi. Tyto diagramy představují velmi široký pohled na systém. Diagram případu užití zobrazuje modelový scénář, ve kterém jednotlivci interagují se systémem pomocí řady specializovaných symbolů a konektorů.

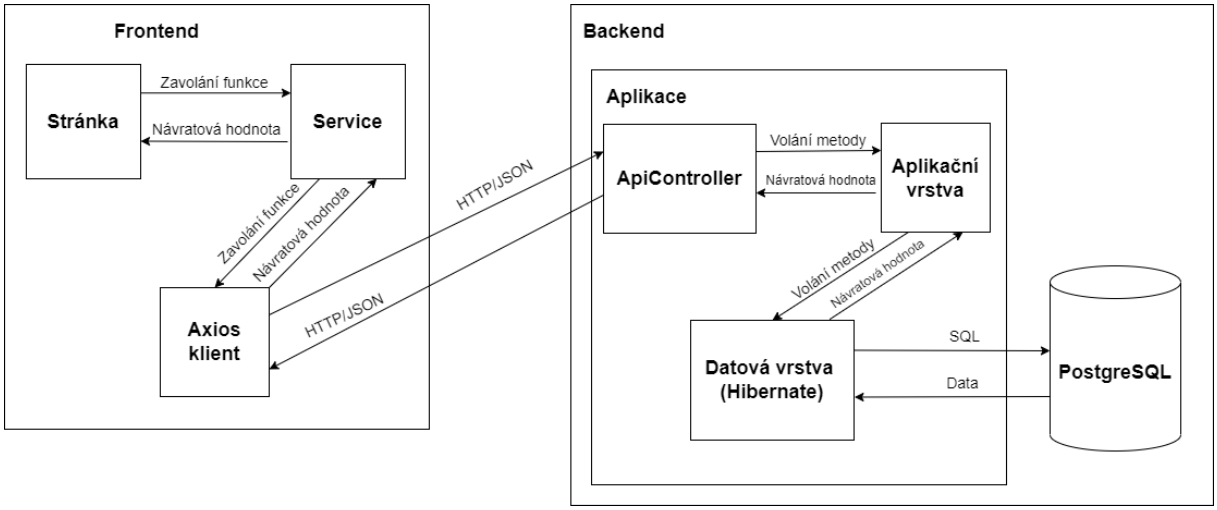
4.3 Návrh řešení

Každý projekt by měl mít před začátkem implementace vytvořen návrh řešení, který představuje vyšší perspektivu, a to představení základních technologií, které bude aplikace používat, základní návrh architektury, a jaké všechny vlastnosti by měla aplikace mít.

4.3.1 Návrh architektury

Aplikace používá klient-server architekturu. Klientskou část aplikace představuje SPA, která je napsána za pomocí Svelte. SPA komunikuje se serverovou částí pomocí http požadavků za pomocí knihovny axios. Autentizace klientské aplikace probíhá pomocí JWT, který se za pomoci axios interceptoru posílá při každém požadavku na server. Klient a server mezi sebou komunikují za pomoci JSON formátu. Při komunikaci jsou použity čtyři http dotazovací metody, každá se speciálním úkolem. Metoda GET je používána pro získání jednoho nebo skupiny záznamů. Dále se využívá metoda POST určená pro tvoření nových řádků, pro jejich úpravu již existujících je určená metoda PUT. Poslední používanou metodou je DELETE, pomocí které dojde k odstranění řádku.

Serverová část aplikace vytvořená za pomocí Springu zpracovává dotazy klientské části a vrací odpovědi. Spring za pomoci Spring Web MVC vytvoří koncové body, se kterými klientská aplikace komunikuje. Jednotlivé koncové body jsou chráněné za pomocí Spring Security, jenž k nim umožnuje přistupovat pouze přihlášeným uživatelům. Serverová aplikace komunikuje s relační databází PostgreSQL za pomoci Hibernate, který pomáhá načítat nebo ukládat záznamy.



Obrázek 2: Základní architektura aplikace (zdroj vlastní)

4.3.2 Funkce aplikace

Hlavní funkcionalitou, kterou aplikace podporuje, je práce s projekty. Projekty představují základní kámen celé aplikace, a proto je s nimi propojená většina funkcí. Další stěžejní vlastností jsou úkoly – každý úkol patří do projektu. Kromě nich může být úkol zařazen do fáze

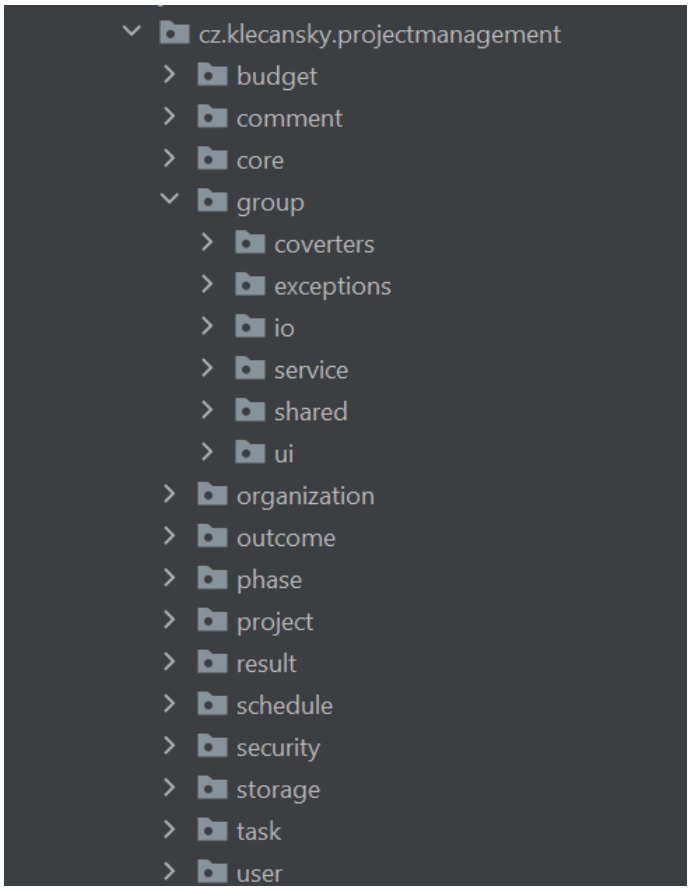
představující ucelenou část životního cyklu projektu. Fáze v sobě obsahuje kromě úkolu také výstupy, které mohou být rozděleny do kategorií. Každý výstup zahrnuje výsledky. Kromě práce s úkoly a s nimi spojenými funkcemi může projekt spravovat svoje členy. Členem projektu může být jak uživatel, tak i tým. Přístup do projektu mají povolen jen členové projektu anebo administrátoři. Do jednotlivých projektů je možno ukládat soubory a ty dále členit do složek. Další funkcionalitou, kterou projekty podporují, je harmonogram, kde si uživatelé mohou plánovat akce. Poslední funkcí, kterou projekty nabízí, je správa rozpočtu. Kromě práce s projekty umožnuje aplikace správu týmů, uživatelů a organizací.

5 IMPLEMENTACE SERVEROVÉ ČÁSTI

V příštích kapitolách bude podrobně popsána implementace serverové části.

5.1 Struktura aplikace

Zdrojové kódy serverové části se nacházejí ve složce backend. Složka backend obsahuje potřebný soubor pro spuštění aplikace pomocí Gradle, a to **build.gradle.kts,** dále obsahuje soubor **backend.dockerfile** používaný pro vytvoření docker image. Samotný zdrojový kód aplikace se nachází ve složce **src,** která používá standardizovanou strukturu projektu Maven, a to **src/main/java**. Složka **java** obsahuje balíčky, které jsou fyzicky reprezentovány složkami. Základní struktura balíčku představuje reverzní doménu s názvem projektu na konci. V této aplikaci je struktura následující: **cz.klecansky.projectmanagement**. Jednotlivé funkcionality aplikace jsou rozdělené do balíčků podle vlastnosti, takže například správa uživatelů je v balíčku **user** nebo správa projektů je v balíčku **project**. Tyto jednotlivé balíčky mají další podbalíčky, které sdružují třídy se stejnými rysy, kupříkladu balíček **exceptions** v sobě obsahuje uživatelsky definované výjimky nebo balíček **io,** který v sobě zahrnuje třídy pro komunikaci s databází.



Obrázek 3: Struktura serverové části aplikace (zdroj vlastní)

5.2 JPA

Veškerá komunikace s databází PostgreSQL byla v aplikaci prováděna za pomocí JPA. Jako implementace JPA byl použit Hibernate s nadstavbou Spring Data JPA.

5.2.1 Návrh entit

Jednotlivé entity se vytvářejí pomocí POJO, což představuje běžné Java objekty. Pro vytvoření funkční entity je zapotřebí třídě přidat anotaci @Entity, která dá JPA na vědomí, že tato třída je entita. Jako další záležitost, kterou je nutné udělat pro fungující entitu, je přiřadit jí nějaký jedinečný identifikátor, který může být buď číslo, nebo UUID. Všechny entity v aplikaci používají UUID jako svůj identifikátor. Výhodou používání UUID jako identifikátoru je, že můžou být generovány v aplikaci, díky čemuž zná aplikace identifikátor ještě předtím, než by byl standartně automaticky vygenerován v databázi. Mezi další výhody patří to, že UUID může být použito jako identifikátor jednotlivých objektů přímo v url bez snížení zabezpečení.

```java
@Entity
@Table(name = "comments")
@Getter
@Setter
public class CommentEntity {

    @Id
    private UUID id;

    @Column(nullable = false, length = 1000)
    private String text;

    @Column(nullable = false)
    private Instant createdDate;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "users_id", nullable = false)
    private UserEntity user;
```

Obrázek 4: Implementace entity komentáře (zdroj vlastní)

5.2.2 Návrh repositářů

Aplikace implementuje repositář skoro pro každou entitu kromě pár výjimek, jako například entitu komentáře. Implementace repositářů je velmi jednoduchá – stačí vytvořit nový interface,

který dědí JpaRepository s generickými parametry. První generický parametr je samotná entita, kterou má repositář používat, a jako druhý parametr je použit datový typ identifikátoru, který používá entita. Po tomto nastavení JPA automaticky vygeneruje základní CRUD metody. Mezi tyto metody patří například metoda findAll, která vrátí všechny instance dané entity v databázi, dále je zde metoda deleteById, která odstraní záznam podle identifikátoru.

```
@Repository
public interface BudgetRepository extends JpaRepository<BudgetEntity, UUID> {
}
```

Obrázek 5: Implementace repositáře rozpočtu (zdroj vlastní)

5.3 Tvorba API

API je na serveru vytvořeno za pomoci Spring Web MVC. Všechny koncové body aplikace se nacházejí za adresou api. K této adrese se poté připojují názvy jednotlivých služeb, které aplikace nabízí – například adresa koncového bodu api/projects zastřešuje práci s projekty. Pro komunikaci s API mezi klientskou aplikací a serverem se využívají čtyři http metody, a to GET sloužící k získávání jednoho nebo více zdrojů. Také se používá metoda POST, která je primárně určená pro tvoření nových vět, nebo metoda PUT, jež upravuje již vytvořené. Poslední používanou metodou je DELETE, která slouží k mazání jednotlivých záznamů. Všechny koncové body kromě těch, které používají metodu GET, vrací speciální třídu SuccessResponse obsahující samotná data a zprávu, která se zobrazí uživatelům v klientské aplikaci. Pokud se na serveru vyskytne nějaký problém, vrátí se speciální třída ErrorResponse obsahující zprávu určenou pro zobrazení uživatelům. Kromě informace určené pro zobrazení obsahuje třída i další důležitá data o vzniklém problému.

```
@DeleteMapping(path = "{id}")
@PreAuthorize("isAuthenticated()")
public ResponseEntity<SuccessResponse> deleteProject(@PathVariable UUID id) {
    projectService.delete(id);
    return ResponseEntity.ok(SuccessResponse.builder().message("Project was successfully deleted.").build());
}
```

Obrázek 6: Implementace koncového bodu pro odstranění projektu (zdroj vlastní)

5.4 Mapování objektů

Aplikace používá třívrstvou architekturu a díky tomu má každá vrstva vlastní objektový model. Prezentační vrstva, která zobrazuje informace pro uživatele, používá příponu Response. Aplikační vrstva, která se stará o logiku aplikace, používá příponu Command a datová vrstva,

která tvoří rozhraní s databází, používá příponu Entity. Při používání aplikace je zapotřebí mezi jednotlivými vrstvami mapovat, protože není cílem uživatelům zobrazovat veškeré informace, které o daném objektu jsou v aplikační vrstvě z důvodu, že by se v objektu mohli nacházet citlivé informace, které nemají být uživatelům posílány. Proces mapování jednotlivých objektů na jiné může být velmi zdlouhavý a repetitivní, pokud je dělán manuálně samotným programátorem. Ale tento způsob není nezbytně nutný, protože Java nabízí řadu knihoven, které tento problém řeší. V této aplikaci se používá knihovna ModelMapper.

```
public ProjectCommand projectEntityToProjectCommand(ProjectEntity projectEntity) {
    return modelMapper.map(projectEntity, ProjectCommand.class);
}
```

Obrázek 7: Ukázka kódu mapování ProjectEntity na ProjectCommand (zdroj vlastní)

5.5 Notifikace

Aplikace umožnuje posílat uživatelům notifikace ve formě e-mailu. Pro posílání e-mailu se v programu používá JavaMailSender, který byl nakonfigurován k použití SMTP serveru. Texty e-mailových notifikací jsou v HTML formátu pro větší přehlednost a možnosti zvýraznění důležitých informací. Aplikace podporuje několik notifikací, které mohou být poslány uživateli. První notifikace, na kterou uživatel narazí, je potvrzení e-mailové adresy. Tato notifikace obsahuje odkaz, který po kliknutí potvrdí uživatelovu e-mailovou adresu. S další notifikací se uživatel bude potkávat jen zřídka, protože se jedná o notifikace, které uživateli přijde na e-mail, když zapomene heslo. Poslední a zároveň nejčastější notifikace upozorňuje, kdy byli uživatel nebo skupina přiřazeni jako řešitelé do úkolu.

```
public void sendAssignToTaskEmail(TaskCommand taskCommand) {
    MimeMessage mimeMessage = javaMailSender.createMimeMessage();
    try {
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, encoding: "utf-8");
        helper.setFrom(FROM_EMAIL);
        helper.setTo(taskCommand.getAssigned().getEmail());
        helper.setSubject(ASSIGNED_TO_TASK_SUBJECT);
        helper.setText(getAssignToTaskEmailHTMLText(taskCommand), html: true);
        javaMailSender.send(mimeMessage);
    } catch (Exception e) {
        throw new SmtpErrorException("Application cannot send email.");
    }
```

Obrázek 8: Zdrojový kód metody pro notifikaci uživatelů (zdroj vlastní)

5.6 Uživatelské role

Každý registrovaný uživatel má přiřazenou roli. Podle uživatelských rolí aplikace rozhoduje, jestli má uživatel právo přistupovat k danému prostředku. Aplikace podporuje tři druhy rolí, a to uživatel, administrátor a super administrátor. Uživatel s rolí uživatele může v aplikaci spravovat projekty, do kterých byl přiřazen jako člen nebo je součástí týmu, jenž je členem projektu. Administrátor na rozdíl od normálního uživatele má přístup ke všem projektům v aplikaci a dále může spravovat uživatele a týmy. Super administrátor má stejná práva jako běžný administrátor, ale navíc má právo odstraňovat ostatní administrátory či může jednotlivým uživatelům měnit jejich role.

Tabulka 3: Oprávnění rolí (zdroj vlastní)

Oprávnění	Uživatel	Administrátor	Super administrátor
Správa všech projektů		✓	✓
Správa projektů, ve kterých je členem	✓	✓	✓
Správa úkolů v projektu	✓	✓	✓
Správa fází projektu	✓	✓	✓
Správa výstupů projektu	✓	✓	✓
Správa harmonogramu	✓	✓	✓
Správa organizací		✓	✓
Změna rolí			✓
Správa týmů		✓	✓
Mazání administrátorů			✓
Správa členů projektu	✓	✓	✓

Správa rozpočtu projektu	✓	✓	✓

5.7 Zabezpečení

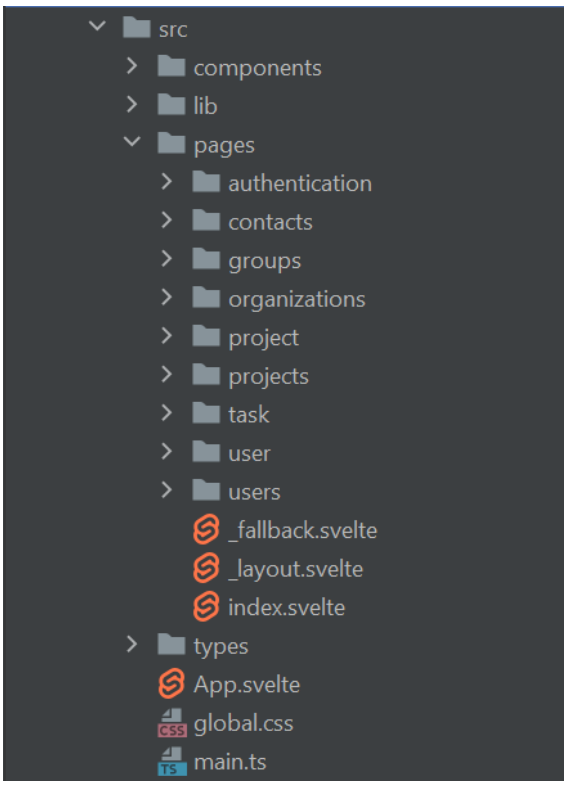
Serverová aplikace je zabezpečena za pomoci Spring Security. Všechny koncové body jsou přístupné pouze přihlášeným uživatelům kromě pár výjimek, jako například koncový bod pro přihlášení uživatele nebo registraci. Přihlášený uživatel vlastní JWT, který při každém dotazu na koncový bod posílá v http hlavičce Authorization. Díky tomuto tokenu server rozpozná o jakého uživatele se jedná a zpřístupní mu povolené zdroje.

6 IMPLEMENTACE KLIENTSKÉ ČÁSTI

V příštích kapitolách bude podrobně popsána implementace klientské části.

6.1 Struktura aplikace

Zdrojové kódy klientské části aplikace se nalézají ve složce frontend. Složka frontend obsahuje potřebný soubor pro spuštění klientské aplikace, a to **package.json** obsahující všechny závislosti, které klientská aplikace vyžaduje a také definuje npm skripty pomocí kterých je samotná aplikace spouštěna. Dále jsou ve složce konfigurační soubory pro nástroje, které frontend používá – kupříkladu **tsconfig.json** nebo **vite.config.js**. Samotný zdrojový kód aplikace se nachází ve složce **src,** která obsahuje čtyři podsložky rozdělující kód podle jejich účelu, a to na **components**, **lib**, **pages** a **types**. Složka **components** obsahuje komponenty, které jsou používány v jednotlivých stránkách. Ve složce **lib** jsou k nalezení služby, které mohou jednotlivé stránky nebo komponenty využívat a jejímž cílem je ve většině případů komunikace s API serverové aplikace. Jednotlivé stránky, které bude uživatel navštěvovat, se nacházejí ve složce **pages**. Poslední zmíněná složka **types** obsahuje uživatelsky definovatelné typy za pomoci TypeScriptu.



Obrázek 9: Struktura aplikace klientská část (zdroj vlastní)

6.2 Připojení k serverovému API

Klientská část aplikace komunikuje se serverovou částí pomocí knihovny axios, která umožnuje posílat http dotazy a přijímat http odpovědi. Ve starších verzích aplikace se pro tento účel používalo nativní fetch api, které má vestavěné každý moderní prohlížeč, ale kvůli absenci podpory http interceptoru jsem přešel na používaní již zmíněné knihovny axios. Všechny dotazy na server se provádějí přes pomocnou funkci **apiRequest** přijímající jako vstup **RequestOptions** objekt, ve kterém lze nastavit, na jaký koncový bod bude dotaz posílat, jakou metodu lze použít či jaké bude tělo daného dotazu. Pro zpřehlednění kódu a pro zamezení opakování se tato funkce používá primárně v takzvaných službách a ne přímo na stránkách. Služby představují abstrakci nad koncovými body serverového api, kupříkladu pokud je nutné ze serveru zjistit všechny projekty, není nutné znát, na jaký koncový bod má být dotaz poslán nebo jakou metodu použít. Stačí zavolat funkci **getAllProjects,** která za nás všechny tyto věci zařídí.

```
export async function apiRequest(options: RequestOptions): ApiResponse {
    const {
        endpointName,
        body = null,
        method = "GET",
        responseType = "json"
    } = options
    try {
        const res = await axios(
            endpoint(endpointName),
            config: {
                method,
                data: method !== "GET" ? body : null,
                responseType
            }
        )
        let data = await res.data;
        return [data, null];
    } catch (error) {
        ifExpiredOrInvalidJWTTokenSignout(error.response.data);
        return [null, error.response.data];
    }
}
```

Obrázek 10: Zdrojový kód funkce apiRequest (zdroj vlastní)

6.3 Autentizace uživatele

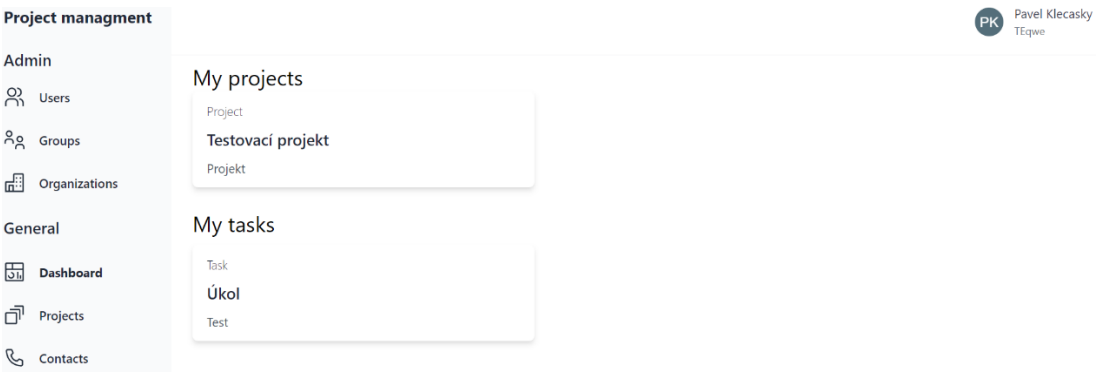
Uživatel se autentizuje pomocí přihlášení. Přihlášení probíhá pomocí e-mailové adresy a hesla která se posílají na server. Poté je ověřena jejich správnost a pokud jsou údaje validní, je nazpět odeslán JSON Web Token a informace o uživateli. Tato data se poté uloží do Local Storage prohlížeče. Pro další dotazy již přihlášený uživatel použije interceptor, který nabízí http klient axios. Do axios je zaregistrována nová funkce, která modifikuje každý dotaz, jenž axios provede. Tato funkce upravuje hlavičku dotazu tak, že pokud existuje v Local Storage JSON Web Token, je přidán do hlavičky **Authorization,** díky které server dokáže rozpoznat o jakého uživatele se jedná.

```
axios.interceptors.request.use( onFulfilled: function (config : AxiosRequestConfig ) {
    config.headers = {
        ...(getAuthorizationHeader() ? {"Authorization": getAuthorizationHeader()} : {}),
        ...config.headers
    }
    return config;
}, onRejected: function (error) {
    return Promise.reject(error);
});
```

Obrázek 11: Zdrojový kód interceptoru (zdroj vlastní)

6.4 Navigace na webové aplikaci

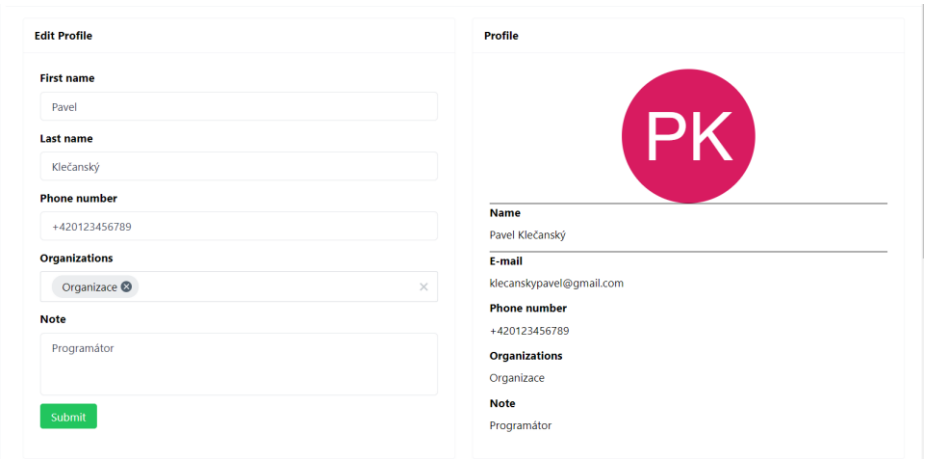
Na všech stránkách se v pravém horním rohu aplikace zobrazuje informace o přihlášeném uživateli. Po kliknutí se zobrazí kontextové menu se dvěma položkami, a to pro zobrazení profilu uživatele nebo odhlášení. Levá strana aplikace obsahuje primární menu, které bude uživatel používat pro navigaci v aplikaci. Menu je rozděleno na dvě části: obecnou, která se zobrazuje všem uživatelům, nebo administrátorskou, ke které má přístup pouze administrátor.



Obrázek 12: Úvodní stránka aplikace (zdroj vlastní)

6.5 Profil uživatele

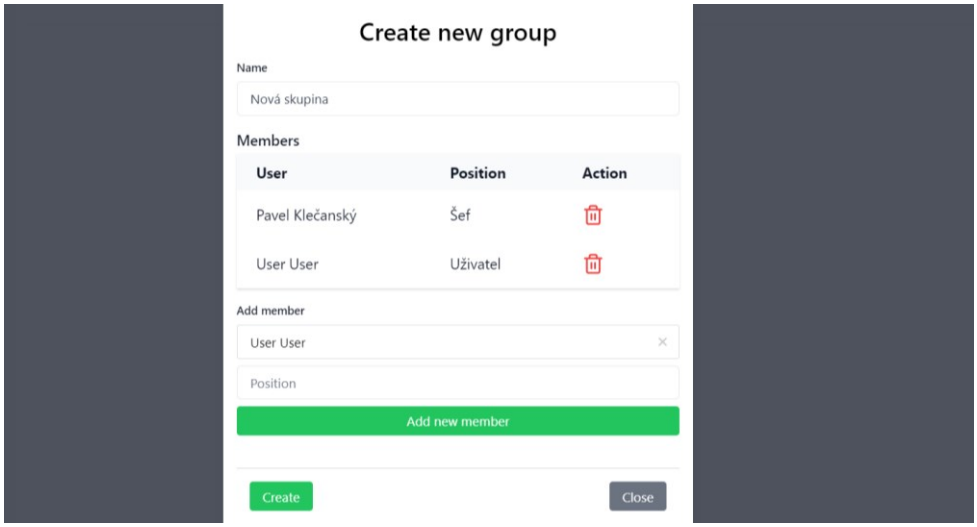
Profil uživatele slouží k zobrazení informací uživatele, úpravě těchto informací a změně hesla. Profil zobrazuje informace o přihlášeném uživateli, a to například jméno, email, telefonní číslo nebo do jaké organizace patří. Stránka také umožnuje uživateli upravovat většinu již zmíněných informací.



Obrázek 13: Profil uživatele (zdroj vlastní)

6.6 Týmy

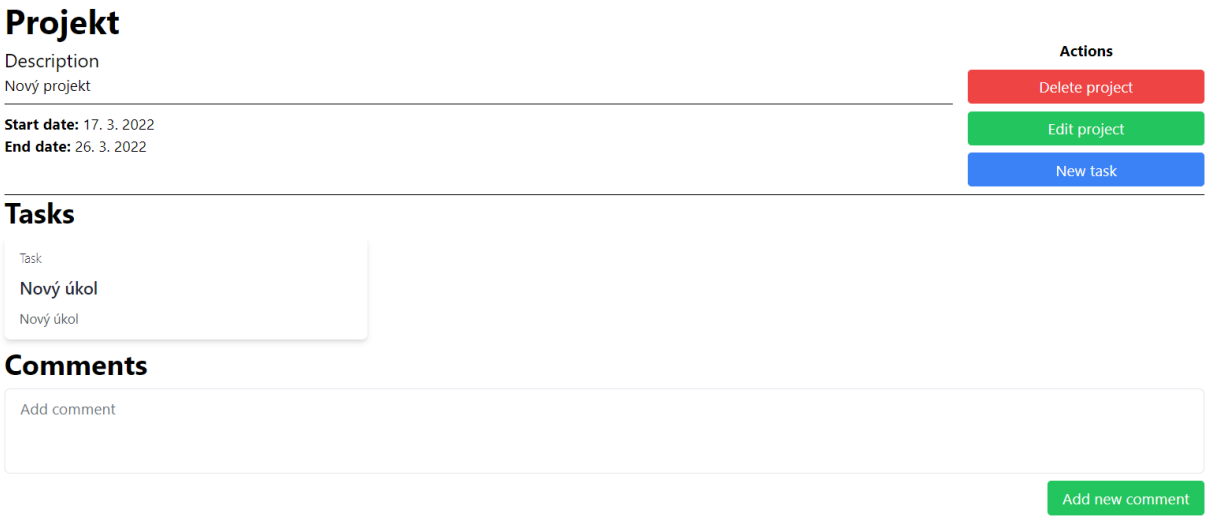
Administrátor může v aplikaci vytvářet týmy různých uživatelů. Každý tým má svůj název a členy. Členové jsou uživatelé s přidělenou pozicí v týmu. Pokud je uživatel přiřazen do nějakého týmu, dostane automaticky přístup k projektům, ve kterých je tým zařazen jako člen. Dále bude také dostávat oznámení při zařazení týmu jako řešitele některého z úkolů.



Obrázek 14: Vytváření nového týmu (zdroj vlastní)

6.7 Projekty

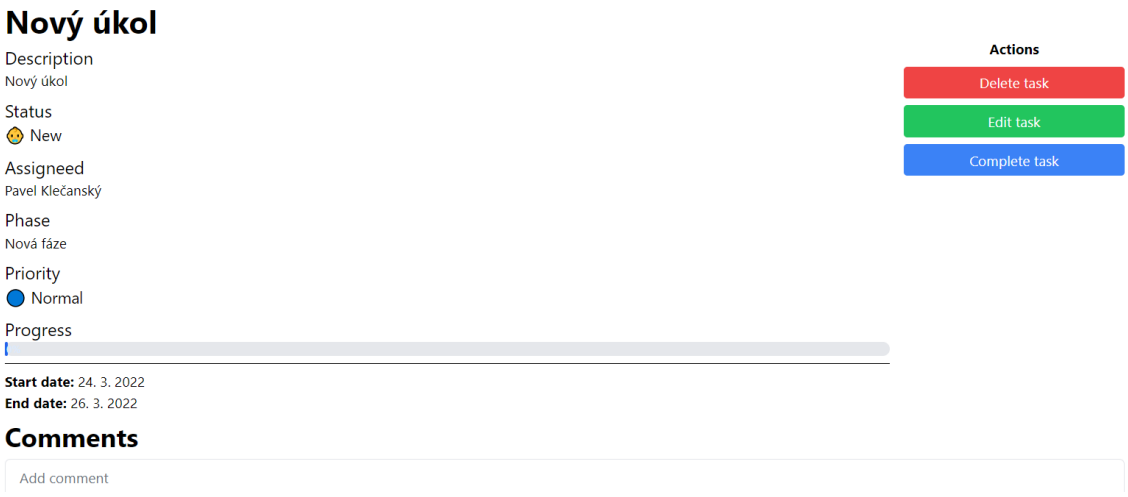
Projekty jsou primární částí celé aplikace. Projekty se nacházejí na stránce projektů, pokud je uživatel administrátor zobrazí se mu na této stránce všechny vytvořené projekty v aplikaci, ale pokud uživatel není administrátorem, zobrazí se mu pouze ty, ve kterých je členem nebo pokud patří do skupiny, která je členem. K detailu projektu je možné se dostat po kliknutí na zobrazený projekt. Stránka detailu obsahuje v horní části menu, pomocí kterého se člověk může přesouvat na jednotlivé podstránky související s projektem, jako například harmonogram projektu, rozpočet projektu nebo členové projektu. Pod menu se na levé straně nachází detail projektu a na pravé straně tlačítka pro smazání projektu, editaci projektu a vytvoření nového úkolu. Pod touto částí stránky jsou zobrazené již vytvořené úkoly a pod nimi se nachází diskuzní část, kde uživatelé mohou pomocí komentářů nad projektem diskutovat.



Obrázek 15: Detail projektu (zdroj vlastní)

6.7.1 Úkoly projektu

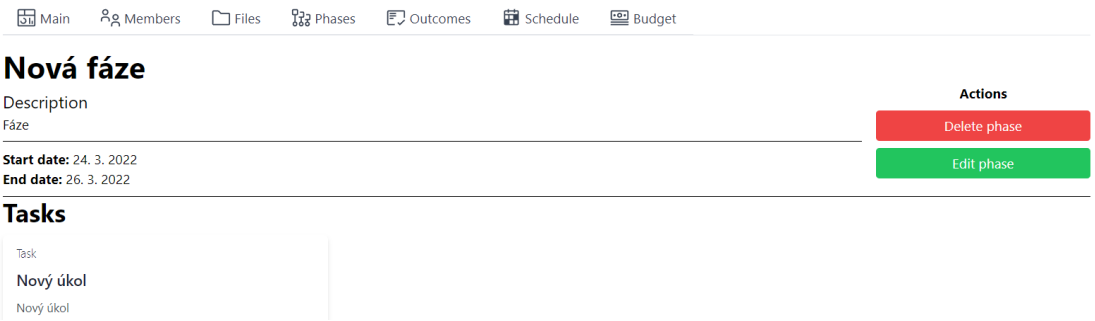
Úkoly jsou stěžejní částí každého projektu. Proto je možné vytvořit nový úkol již na úvodní stránce projektu. Tlačítko pro vytvoření nového úkolu se nachází na pravé straně stránky. Po kliknutí na dané tlačítko se zobrazí modální okno s větším množstvím formulářů, mezi něž patří například formulář pro vybrání statusu, fáze nebo pro jakého uživatele či skupinu bude úkol přiřazen. Úvodní stránka rovněž obsahuje již vytvořené úkoly, které po kliknutí přesměrují uživatele na stránku s podrobnostmi k danému úkolu. Stránka s podrobnostmi o úkolu v levé části zobrazuje detaily úkolu, jako například jeho prioritu nebo postup plnění. V pravé části se nachází tlačítka pro smazání úkolu, úpravu nebo dokončení. Pod detaily úkolu mohou uživatele diskutovat pomocí komentářů.



Obrázek 16: Detail úkolu (zdroj vlastní)

6.7.2 Fáze projektu

Jednou z podstránek u projektů je fáze projektu. Po zobrazení této stránky uvidí uživatel již vytvořené fáze a v pravém horním rohu se nachází tlačítko na vytvoření nové fáze, které zobrazí modální okno s formuláři pro vytvoření nové fáze. Na jednotlivé již vytvořené fáze, může uživatel kliknout, čímž bude přesunut na stránku s podrobnostmi k dané fázi. Stránka zobrazuje detaily o fázi, a to název, popis, počáteční datum a koncové datum. Dále také obsahuje dvě tlačítka na editaci fáze a na smazání fáze. Pokud byl nějaký úkol přidán do fáze, zobrazí se pod zmíněnými detaily.

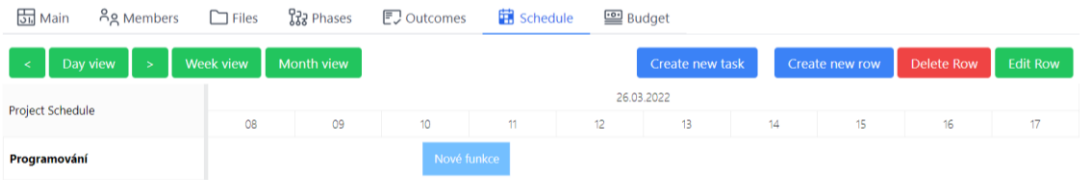


Obrázek 17: Detail fáze (zdroj vlastní)

6.7.3 Harmonogram projektu

Každý projekt má svůj harmonogram, který se nachází za položkou Schedule v projektovém menu. Na tvorbu harmonogramu byla použita knihovna svelte-gantt poskytující Ganttův diagram, který muže být vložen přímo do aplikace. Díky této knihovně se řádově zjednodušilo vytvoření harmonogramu. Stránka obsahuje samotný diagram a nad ním dvě skupiny tlačítek – jednu pro úpravu obsahu harmonogramu a druhou na změnu zobrazení. Úprava harmonogramu

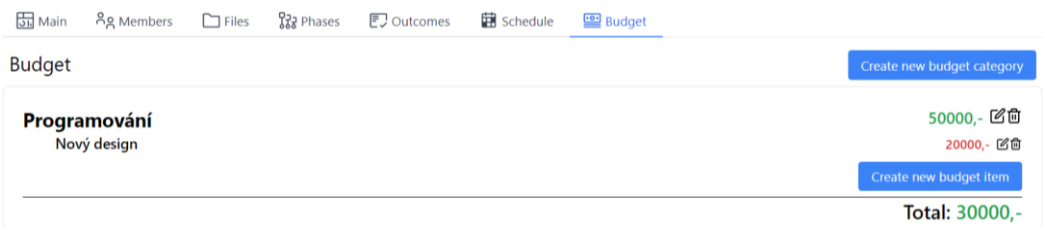
probíhá primárně pomocí zmíněné skupiny tlačítek, která obsahuje tlačítka například pro vytváření nového řádku, vytvoření nového úkolu nebo smazání řádku. Jednotlivé úkoly je možné upravovat a mazat přímo v diagramu. Stačí na daný úkol kliknout a zobrazí se kontextové menu, ve kterém bude možné vybrat, jestli chceme úkol odstranit nebo editovat. Druhou skupinou tlačítek jsou, jak již bylo zmíněno, tlačítka, které upravují zobrazování harmonogramu. Díky těmto tlačítkům můžeme upravovat, v jakém rozmezí času se bude harmonogram zobrazovat. Tato skupina obsahuje tlačítka jako například Day view, které zobrazí úkoly na tento den, Week view, které zobrazí úkoly na tento týden, nebo Month view, které zobrazí úkoly na tento měsíc.



Obrázek 18: Harmonogram projektu (zdroj vlastní)

6.7.4 Rozpočet projektu

U jednotlivých projektů je možné spravovat jejich rozpočet. Každý rozpočet se skládá ze dvou základní struktur, a to kategorie rozpočtu a v ní jednotlivé položky. Na stránce rozpočtu se v pravé horní části nachází tlačítko pro vytváření kategorie, které po kliknutí zobrazí modální okno s možností vyplnění názvu dané kategorie a jejím rozpočtem. Pokud jsou již nějaké kategorie rozpočtu vytvořené, zobrazí se na jako oddělené bloky. Jednotlivé bloky jsou rozdělené na tři části. První část je hlavička, kde jsou informace o kategorii, a to název a její rozpočet. Dále také obsahuje ikony pro editaci a odstranění kategorie. Druhou částí je tělo, které obsahuje list položek dané kategorie a tlačítko pro vytváření nových položek. Poslední částí je patička, zobrazující zbývající rozpočet v kategorii po odečtení všech položek. Při úpravě nebo vytváření nový položek je validováno, aby výdaje nepřesáhly rozpočet kategorie.



Obrázek 19: Rozpočet projektu (zdroj vlastní)

7 TESTOVÁNÍ APLIKACE

Lidé jsou náchylní k chybám kvůli nepozornosti, nesprávným předpokladům, nedbalosti nebo nedostatečné znalosti systému. Právě tato lidská vlastnost činí software zranitelným vůči chybám a defektům. Chceme-li těmto problémům zabránit, je nutné software řádně otestovat. Tradičně se testování softwaru provádělo v jedné fázi, a to až poté, co byla dokončena implementace nebo kódování. [43]

7.1 Testovací scénáře

Testovací scénář je v podstatě dokumentace případů užití. Jinými slovy, popisuje akci, kterou může uživatel provést s webovou aplikací. Testovací scénáře jsou vytvořeny tak, aby zajistily, že všechny jednotlivé funkce nabízené webovou aplikací fungují podle očekávání. Chceme-li vytvořit přesné testovací scénáře, je nejlepší shromáždit informace od klientů, zúčastněných stran a vývojářů. To napomáhá k efektivnímu pokrytí všechny možných uživatelských scénářů a komplexnímu testování celé aplikace. Testovací scénáře jsou vyžadovány k ověření celého systému z pohledu uživatelů. Při jejich vytváření se testeři musí vžít do situace uživatelů, aby měli jasno v tom, jaké reálné scénáře bude muset software po zveřejnění zvládnout. [44]

Pro přehled oprávnění jednotlivých rolí viz Tabulka 3.

7.1.1 Práce s projektem a úkoly

Role použita pro scénář: Uživatel

- Ověřte, že uživatel může pomocí menu přejít na stránku Projects.
- Ověřte, že daná stránka obsahuje tlačítko na vytvoření nového projektu.
- Ověřte, že tlačítko po kliknutí zobrazí modální okno, které obsahuje formuláře na vytvoření nového projektu.
- Ověřte, že pro vytvoření nového projektu je potřeba vyplnění všech polí formuláře.
- Ověřte, že při úspěšném vytvoření projektu aplikace zobrazí oznámení a projekt bude viditelný.
- Ověřte, že kliknutí na název vytvořeného projektu přesune uživatele na stránku projektu, která obsahuje podrobnější informace o samotném projektu.
- Ověřte, že se po kliknutí na tlačítko Edit project zobrazí modální okno, které obsahuje předvyplněná pole formuláře na editování projektu.

- Ověřte, že po změně nějaké vlastnosti projektu a zmáčknutí tlačítka Edit se změna zobrazila na stránce projektu.
- Ověřte, že se po kliknutí na tlačítko New task zobrazí modální okno, které obsahuje formuláře na vytvoření nového úkolu.
- Ověřte, že se po vyplnění všech potřebných formulářů a kliknutí na tlačítko Create vytvoří nový úkol, který bude viditelný na stránce projektu.
- Ověřte, že po kliknutí na tlačítko Delete project bude projekt odstraněn a uživatel bude přesměrován na stránku projektů.

Závěr testování: Při tvorbě projektu i úkoly chybí u jednotlivých polí formuláře nápověda, která by se mohla zobrazit například po najetí myši na název pole formuláře.

Tento testovací scénář lze odzkoušet pomocí Google Forms na této adrese: https://forms.gle/Vdk9ptJ8RCT4AreS9

7.1.2 Práce s fázemi a výstupy

Role použita pro scénář: Uživatel, při práci na tomto scénáři vyberte projekt s názvem Testovací.

- Ověřte, že menu na stránce projektu obsahuje políčko Phases.
- Ověřte, že po kliknutím na políčko Phases bude uživatel přesměrován na stránku fází.
- Ověřte, že stránka fází obsahuje tlačítko Create new phase na vytvoření nové fáze.
- Ověřte, že se po kliknutí na tlačítka Create new phase zobrazí modální okno, které obsahuje formuláře pro vytvoření nové fáze.
- Ověřte, že se pro vyplnění všech potřebných formulářů a kliknutí na tlačítko Create vytvoří nová fáze, která bude viditelná na stránce fází.
- Ověřte, že menu na stránce projektu obsahuje políčko Outcomes.
- Ověřte, že po kliknutí na políčko Outcomes bude uživatel přesměrován na stránku výstupů.
- Ověřte, že stránka výstupů obsahuje tlačítko Create new outcome na vytvoření nového výstupu a tlačítko Create new outcome category pro vytvoření nové kategorie výstupů.

- Ověřte, že se po kliknutí na tlačítka Create new outcome category zobrazí modální okno, které obsahuje formuláře pro vytvoření nové kategorie výstupů.
- Ověřte, že se pro vyplnění všech potřebných formulářů a kliknutí na tlačítko Create vytvoří nová kategorie výstupů, která bude viditelná na stránce výstupů.
- Ověřte, že se po kliknutí na tlačítko Create new outcome zobrazí modální okno, které obsahuje formuláře pro vytvoření nového výstupu, primárně ověřte, že formulář Phase umožnuje vybrat dříve vytvořenou fázi a formulář Outcome Category umožnuje také vybrat dříve vytvořenou kategorii výstupů.
- Ověřte, že se pro vyplnění všech potřebných formulářů a kliknutí na tlačítko Create vytvoří nový výstup, který bude viditelná na stránce výstupů.

Závěr testování: Při práci s fázemi a výstupy projektu chybí u jednotlivých polí formuláře nápověda, která by se mohla zobrazit například po najetí myši na název pole formuláře. Při tvorbě a správě výstupů uživatelům chybí zobrazení podrobnějších informací o fázích, které projekt má a do kterých výstup spadá. Obdobně je požadovaná změna i při tvorbě fází, kdy není zobrazeno, kdy již začínají dříve vytvořené fáze projektu a ještě na vyšší úrovni při tvorbě fází chybí zobrazení délky trvání projektu v podobě datumů od a do.

Tento testovací scénář lze odzkoušet pomocí Google Forms na této adrese: https://forms.gle/eDZifTEDaoPqEMnt8

7.1.3 Práce s harmonogramem

Role použita pro scénář: Uživatel, při práci na tomto scénáři vyberte projekt s názvem Testovací.

- Ověřte, že menu na stránce projektu obsahuje políčko Schedule.
- Ověřte, že po kliknutí na políčko Schedule bude uživatel přesměrován na stránku harmonogramu.
- Ověřte, že stránka harmonogramu obsahuje tlačítka Create new task, Create new row, Delete Row a Edit Row. Dále také navigační tlačítka Day view, Week view a Month view.
- Ověřte, že před vytvořením řádku do harmonogramu se při pokusu o vytvoření nového úkolu zobrazí oznámení o nutnosti vytvoření řádku.

- Ověřte, že se po kliknutí na tlačítka Create new row zobrazí modální okno, které obsahuje formulář pro vytvoření nového řádku.
- Ověřte, že se po vyplnění formuláře s názvem řádku a kliknutí na tlačítko Create vytvoří nový řádek, který bude viditelný na stránce harmonogramu.
- Ověřte, že se po kliknutí na tlačítka Create new task zobrazí modální okno, které obsahuje formuláře pro vytvoření nového úkolu. Primárně ověřte, že formulář Row umožnuje vybrat dříve vytvoření řádek.
- Ověřte, že se po vyplnění všech potřebných formulářů a kliknutí na tlačítko Create vytvoří nový úkol, který bude viditelný na stránce harmonogramu.
- Ověřte, že jednotlivá navigační tlačítka fungují – tlačítko Day view zobrazuje denní pohled, tlačítko Week view týdenní pohled a tlačítko Month view měsíční pohled.

Závěr testování: Při práci s harmonogramem a vytváření úkolu do harmonogramu by se měl zobrazit kalendář i s možností vložení času. V přehledu navigačních tlačítek chybí zobrazení Year view. Při práci s harmonogramem chybí u jednotlivých polí formuláře nápověda, která by se mohla zobrazit například po najetí myši na název pole formuláře. Při tvorbě plánu není možné převzít již vytvořené fáze a úkoly z projektu, což je nepříjemné, a i když v plánu bývají jen hlavní fáze a úkoly, bylo by vhodné, aby bylo možné některé do plánu vybrat.

Tento testovací scénář lze odzkoušet pomocí Google Forms na této adrese: https://forms.gle/znekTs4HUtgFzMEP8

7.2 Závěr testování

Po testování byly implementovány tyto připomínky: chybějící tlačítko zobrazení Year view, nápovědy u polí formulářů, informace o délce projektu při tvorbě fáze, informace o fázích při správě výstupů.

Při tvorbě plánu není možné převzít již vytvořené fáze a úkoly z projektu. Tato část harmonogramu nebyla implementována z časových důvodů, ovšem je možné rozšíření bakalářské práce s řešením: harmonogram by byl upraven takovým způsobem, aby podporoval zobrazování i již vytvořené fáze a úkoly. A to tím způsobem že by úkoly harmonogramu, úkoly z projektu a fáze implementovali rozhraní které by obsahovalo všechny informace které harmonogram potřebuje pro zobrazení úkolu.

ZÁVĚR

Bakalářská práce vznikla s cílem vytvořit webovou aplikaci umožňující řízení různých vědeckých a příbuzných projektů. V první části práce, která se věnuje teorii, dochází k podrobnému seznámení se Spring ekosystémem, ve kterém byla implementovaná celá serverová část aplikace. Pro možné srovnání jsou v práci popsány další webové frameworky, které nám Java nabízí. Poslední část teoretického bloku obsahuje podrobný popis dalších technologií, které nepatří do Spring ekosystému, ale byly v práci použity ať už na serveru nebo pro vytvoření klientské webové aplikace.

V rámci praktické části je nejprve představena analýzu projektu, kde jsou vypsány funkční a nefunkční požadavky kladené na systém nebo diagram případů užití. Implementace praktického výstupu práce je popsána v kapitolách 6 Implementace serverové části a 7 Implementace klientské části. Součástí těchto oddílů je podrobný výklad celého procesu implementace tohoto výstupu včetně samotné struktury, mapování objektů, stanovení uživatelských rolí a zabezpečení na straně serveru, ale také způsob autentizace uživatele a navigace ve webové aplikace na klientské části aplikace.

Pro ověření kvality navrženého a realizované aplikace pro řízení vědeckých a příbuzných projektů bylo provedeno testování na připravených scénářích. Cílem tohoto postupu byla verifikace skutečné využitelnosti aplikace. V rámci tohoto šetření bylo navrženo několik drobných úprav, které byly implementovány a výstup lze tak považovat za ověřitelný v praxi, a tak vytyčený cíl bakalářské práce byl splněn. Při práci s harmonogramem nebyly implementována jediná funkcionalita, a to převzetí již vytvořených fází a úkoly z projektu do harmonogramu, byl ale navržen postup pro budoucí možnou implementaci.

Rozsáhlý vývoj výstupní aplikace mi rozšířil znalosti a převážně jsem se zdokonalil ve využití frameworku Spring, který jsem využíval v pracovní oblasti, ale nikdy jsem v něm netvořil aplikaci od analýzy až k samotné realizaci. Moje portfolia bylo při tvorbě bakalářské práce rozšířeno o pokročilé možnosti Svelte a Tailwindcss, které jsem použil pro vytvoření klientské části aplikace. Kromě jednotlivých využívaných technologií jsem získal zkušenost s prací na větším projektu s omezeným množstvím času, i tato zkušenost mi bude určitě velmi prospěšná v budoucí tvorbě aplikací.

POUŽITÁ LITERATURA

[1] WALLS, Craig. *Spring in Action*. Fifth Edition. New York: Manning Publications, 2018. ISBN 9781617294945.

[2] SHARMA, Sourabh. *Modern API Development with Spring and Spring Boot: Design highly scalable and maintainable APIs with REST, gRPC, GraphQL, and the reactive paradigm*. Birmingham: Packt Publishing Limited, 2021. ISBN 9781800562479.

[3] What is Quarkus? In: *Red Hat* [online]. 13.4.2020 [cit. 8.12.2021]. Dostupné z: https://www.redhat.com/en/topics/cloud-native-apps/what-is-quarkus.

[4] ÅSE, David. Javalin: A Simple, Modern Web Server Framework. In: *Java Magazine* [online]. 15.5.2019 [cit. 9.12.2021]. Dostupné z: https://blogs.oracle.com/javamagazine/post/javalin-a-simple-modern-web-server-framework.

[5] KLEIN, Zachary. Micronaut: A Java Framework for the Future, Now. In: *Object Computing* [online]. 7.2018 [cit. 9.12.2021]. Dostupné z: https://objectcomputing.com/resources/publications/sett/july-2018-micronaut-framework-for-the-future.

[6] What is Play? In: *Play Framework* [online]. 2021 [cit. 13.12.2021]. Dostupné z: https://www.playframework.com/documentation/2.8.11/Introduction.

[7] Introducing Play 2. In: *Play Framework* [online]. 2021 [cit. 13.12.2021]. Dostupné z: https://www.playframework.com/documentation/2.8.11/Philosophy.

[8] ZANINI, Antonello. A Complete Guide to Lombok. In: *Auth0* [online]. 29.7.2021 [cit. 22.01.2022]. Dostupné z: https://auth0.com/blog/a-complete-guide-to-lombok/

[9] ESCHWEILER, Sebastian. Tailwind CSS For Absolute Beginners. In: *CodingTheSmartWay.com* [online]. 8.3.2020 [cit. 23.01.2022]. Dostupné z: https://codingthesmartway.com/tailwind-css-for-absolute-beginners

[10] EKWUNO, Obinna. Tailwind CSS vs. Bootstrap: Is it time to ditch UI kits? In: *LogRocket* [online]. 5.7.2021 [cit. 23.01.2022]. Dostupné z: https://blog.logrocket.com/tailwind-css-vs-bootstrap-ui-kits/

[11] Get started with Tailwind CSS. In: *Tailwind Labs Inc* [online]. 2022 [cit. 23.1.2022]. Dostupné z: https://tailwindcss.com/docs/installation

[12] Getting started with Svelte. In: *MDN Web Docs* [online]. 22.01.2022 [cit. 23.1.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started

[13] What is TypeScript? In: *Educative* [online]. 12.03.2019 [cit. 23.1.2022]. Dostupné z: https://www.educative.io/edpresso/what-is-typescript

[14] How To Set Up a New TypeScript Project. In: *DigitalOcean* [online]. 19.10.2020 [cit. 24.1.2022]. Dostupné z: https://www.digitalocean.com/community/tutorials/typescript-new-project

[15] What is PostgreSQL? In: *IBM* [online]. 6.8.2019 [cit. 25.1.2022]. Dostupné z: https://www.ibm.com/cloud/learn/postgresql

[16] What Is PostgreSQL? In: *OpenLogic* [online]. 12.8.2021 [cit. 25.1.2022]. Dostupné z: https://www.openlogic.com/blog/what-is-postgresql

[17] KEITH, Mike, SCHINCARIOL, Merrick a NARDONE, Massimo. *Pro JPA 2 in Java EE 8.* Third Edition. New York: Apress, 2018. ISBN 978-1-4842-3420-4.

[18] GUPTA, Lokesh. Hibernate Hello World Example. In: *HowToDoInJava* [online]. 25.1.2022 [cit. 27.1.2022]. Dostupné z: https://howtodoinjava.com/hibernate/hibernate-hello-world-application/

[19] EBERSOLE, Steve. Hibernate 3.5.0-Final release. In: *Hibernate* [online]. 1.4.2010 [cit. 27.1.2022]. Dostupné z: https://in.relation.to/2010/04/01/hibernate-350-final-release/

[20] Hibernate ORM 5.6 series. In: *Hibernate* [online]. 26.1.2022 [cit. 27.1.2022]. Dostupné z: https://hibernate.org/orm/releases/5.6/

[21] Hibernate OGM 5.4.1.Final: Reference Guide. In: *JBoss.org* [online]. 18.12.2018 [cit. 27.1.2022]. Dostupné z: https://docs.jboss.org/hibernate/stable/ogm/reference/en-US/html_single/

[22] Hibernate Validator 7.0.2.Final – Jakarta Bean Validation Reference Implementation: Reference Guide. In: *JBoss.org* [online]. 14.12.2021 [cit. 27.1.2022]. Dostupné z: https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/

[23] Hibernate Search 6.1.0.Final: Reference Documentation. In: *JBoss.org* [online]. 25.1.2022 [cit. 27.1.2022]. Dostupné z: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html_single/#getting-started-compatibility

[24] GUPTA, Lokesh. Gradle Tutorial – Installation and Hello World Example. In: *HowToDoInJava* [online]. 26.12.2020 [cit. 27.1.2022]. Dostupné z: https://howtodoinjava.com/gradle/gradle-tutorial-installation-and-hello-world-example/

[25] Gradle User Manual. In: *Gradle* [online]. ©2022 [cit. 27.1.2022]. Dostupné z: https://docs.gradle.org/current/userguide/userguide.html

[26] What is Gradle? In: *Gradle* [online]. ©2022 [cit. 27.1.2022]. Dostupné z: https://docs.gradle.org/current/userguide/what_is_gradle.html

[27] IntelliJ IDEA overview. In: *JetBrains* [online]. 26.8.2021 [cit. 27.1.2022]. Dostupné z: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

[28] IntelliJ IDEA Tutorial. In: *JavaTpoint* [online]. ©2021 [cit. 27.1.2022]. Dostupné z: https://www.javatpoint.com/intellij-idea-tutorial

[29] LEIJA, Ruben. How to add routing to Svelte SPA with Routify. In: *Linguine Code* [online]. 2020 [cit. 27.1.2022]. Dostupné z: https://linguinecode.com/post/how-to-add-routes-to-svelte-spa-with-routify

[30] Creating an app. In: *Routify* [online]. 28.1.2021 [cit. 27.1.2022]. Dostupné z: https://www.routify.dev/guide/installation

[31] Getting Started. In: *Vite* [online]. 18.01.2022 [cit. 28.1.2022]. Dostupné z: https://vitejs.dev/guide/

[32] What is Java? In: *Opensource.com* [online]. ©2022 [cit. 28.1.2022]. Dostupné z: https://opensource.com/resources/java

[33] ERINC, Yigit Kemal. Spring Security Overview. In: *Auth0* [online]. 15.4.2021 [cit. 28.01.2022]. Dostupné z: https://auth0.com/blog/spring-security-overview/

[34] Java Spring Boot. In: *IBM* [online]. 25.3.2020 [cit. 29.1.2022]. Dostupné z: https://www.ibm.com/cloud/learn/java-spring-boot

[35] BOS, Spencer. Java Basics: What Is Spring Boot? In: JRebel [online]. 5.8.2020 [cit. 29.1.2022]. Dostupné z: https://www.jrebel.com/blog/what-is-spring-boot

[36] Spring Cloud. In: *Spring* [online]. 2021 [cit. 30.1.2022]. Dostupné z: https://spring.io/projects/spring-cloud

[37] Spring for GraphQL. In: *Spring* [online]. ©2022 [cit. 30.1.2022]. Dostupné z: https://spring.io/projects/spring-graphql

[38] Spring Data. In: *Spring* [online]. 2021 [cit. 30.1.2022]. Dostupné z: https://spring.io/projects/spring-data

[39] History of Spring Framework and Spring Boot. In: *Quick Programming Tips* [online]. 26.3.2017 [cit. 30.1.2022]. Dostupné z: https://www.quickprogrammingtips.com/spring-boot/history-of-spring-framework-and-spring-boot.html

[40] Spring Framework Documentation. In: *Spring* [online]. 17.2.2022 [cit 19.2.2022]. Dostupné z: https://docs.spring.io/spring-framework/docs/5.3.16/reference/html/

[41] RAJPUT, Dinesh. Spring 5 Design Patterns. Birmingham: Packt Publishing. 2017. ISBN 9781788299459.

[42] GORBACHENKO, Pavel. What are Functional and Non-Functional Requirements and How to Document These. In: *Enkonix* [online]. 9.4.2021 [cit 22.2.2022]. Dostupné z: https://enkonix.com/blog/functional-requirements-vs-non-functional/

[43] KULDEEP, Rana. What is Software Testing? In: *ArtOfTesting* [online]. 29.5.2021 [cit. 22.03.2022]. Dostupné z: https://artoftesting.com/what-is-software-testing

[44] BOSE, Shreya. How to create Test Scenarios with Examples. In: *BrowserStack* [online]. 10.5.2021 [cit. 22.03.2022]. Dostupné z: https://www.browserstack.com/guide/how-to-create-test-scenarios

[45] Use Case Diagram: Definition and Examples. In: *Indeed* [online]. 22.4.2021 [cit. 22.03.2022]. Dostupné z: https://www.indeed.com/career-advice/career-development/use-case-diagram

PŘÍLOHY

PŘÍLOHA A – DATABÁZOVÁ APLIKACE PRO ŘÍZENÍ VĚDECKÝCH A PŘÍBUZNÝCH PROJEKTŮ

Obsah přílohy práce:

- Zdrojové kódy klientské i serverové části webové aplikace
- Databáze DDL soubor
- Tento dokument