

**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**  
**FAKULTA PRÍRODNÝCH VIED A INFORMATIKY**

**POROVNANIE TECHNOLOGIÍ V JAZYKU JAVA**  
**PRI TVORBE WEBOVEJ APLIKÁCIE**  
**BAKALÁRSKA PRÁCA**

**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**  
**FAKULTA PRÍRODNÝCH VIED A INFORMATIKY**

**POROVNANIE TECHNOLOGIÍ V JAZYKU JAVA PRI**  
**TVORBE WEBOEJ APLIKÁCIE**  
**BAKALÁRSKA PRÁCA**

Študijný odbor:	18. Informatika
Študijný program:	Aplikovaná informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Mgr. Dávid Držík



Univerzita Konštantína Filozofa v Nitre  
Fakulta prírodných vied a informatiky

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Jakub Antala  
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)  
**Študijný odbor:** informatika  
**Typ záverečnej práce:** Bakalárska práca  
**Jazyk záverečnej práce:** slovenský  
**Sekundárny jazyk:** anglický

**Názov:** Porovnanie technológií v jazyku Java pri tvorbe webovej aplikácie

**Anotácia:** V súčasnej digitálnej ére je tvorba webových aplikácií kľúčovým aspektom pre mnohé organizácie. Jazyk Java zostáva jedným z najpopulárnejších jazykov pre vývoj webových aplikácií. Medzi možnými technológiami pre vývoj webových aplikácií patria tradičné JavaServer Pages (JSP) a novšie aj Java Spring (vrátane Spring Boot).

### Cieľ práce:

Cieľom práce je porovnať a analyzovať tieto dve technológie (JSP a Java Spring), pri vývoji webovej aplikácie plánovača udalostí a rezervácií pre podnikový sektor. V teoretickej časti sa autor zameria na analýzu a porovnanie týchto dvoch technológií z hľadiska kvality kódu, výkonu, bezpečnosti a podpory komunity. Na základe získaných výsledkov bude v práci uvedené odporúčanie pre výber vhodnej technológie pre konkrétny projekt.

Praktická časť práce bude zahŕňať návrh, implementáciu a testovanie aplikácie s dôrazom na jednoduchú registráciu udalostí a priateľské prostredie pre prezeranie a registráciu na tieto udalosti.

### Charakter práce:

Vývoj softvéru – popis riešeného problému, návrh softvéru (modely, ..), metodika vývoja, implementácia, popis vytvoreného softvéru, testovanie.

### Predmetové prerekvizity:

Jazyky webu (1., Bc.)

Databázové systémy (1., Bc.)

Objektové technológie (2., Bc.)

Webové aplikácie na platforme Java (3., Bc.)

### Najdôležitejšie kompetentnosti získané spracovaním témy:

- má základné znalosti webového programovania a je schopný vytvoriť vlastnú serverovú aplikáciu,
- dokáže definovať požiadavky na vývoj softvéru a vykonáva kontrolu požadovaných funkčností a vlastností pri tvorbe softvéru,
- má schopnosť navrhovať vhodné softvérové technológie pre konkrétne úlohy, vrátane návrhu riešení na optimalizáciu a efektívnejšie využívanie výpočtových zdrojov,
- implementuje vhodné metódy a techniky pre vývoj softvéru a vykonáva testovanie vyvíjaných softvérových riešení.



Univerzita Konštantína Filozofa v Nitre  
Fakulta prírodných vied a informatiky

---

**Školiteľ:** Mgr. Dávid Držík  
**Oponent:** Mgr. František Forgáč, PhD.  
**Katedra:** KI - Katedra informatiky  
**Dátum zadania:** 09.10.2023

**Dátum schválenia:** 07.03.2025

RNDr. Ján Skalka, PhD., v. r.  
vedúci/a katedry

## **POĎAKOVANIE**

Na tomto mieste by som rád využil možnosť vyjadriť vďaku môjmu školiteľovi, pánovi Mgr. Dávidovi Držíkovi, za odborné vedenie, cenné rady a konštruktívne pripomienky, ktoré mi počas vypracovania tejto bakalárskej práce výrazne pomohli. Vážim si jeho trpezlivosť, podporu a ochotu venovať mi svoj voľný čas aj v náročnejších fázach riešenia tejto práce. Jeho usmernenia a skúsenosti výrazne prispeli k lepšiemu pochopeniu problematiky a kvalitnejšiemu spracovaniu výsledkov.

# ABSTRAKT

ANTALA, Jakub: Porovnanie technológií v jazyku Java pri tvorbe webovej aplikácie. [Bakalárska práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied a informatiky. Školiteľ: Mgr. Dávid Držík. Stupeň odbornej kvalifikácie: Bakalár odboru Aplikovaná informatika. Nitra: FPVaI, 2025. .... 55 s.

V tejto bakalárskej práci sme sa zamerali na porovnanie dvoch prístupov k vývoju webových aplikácií v jazyku Java, konkrétne JavaServer Pages (JSP) a frameworku Spring Boot. Cieľom práce bolo navrhnúť, implementovať a otestovať aplikácie plánovača udalostí a rezervácií pre podnikový sektor, pričom bola každá aplikácia vytvorená samostatne pre danú technológiu. Teoretická časť práce sa venovala analýze oboch technológií z hľadiska architektúry, spôsobu vývoja a možností zabezpečenia. V praktickej časti sme detailne popísali návrh databázy, implementáciu jednotlivých funkcií aplikácie a samotný vývoj v oboch technológiách. Obe verzie aplikácie sme následne podrobili testovaniu, kde sme sa zamerali na kvalitu kódu, výkonnosť, bezpečnosť a podporu zo strany vývojárskej komunity. Výsledky testovania ukázali, že aplikácia vyvinutá pomocou Spring Boot dosahovala vyššiu mieru prehľadnosti kódu a lepšiu štruktúru, zatiaľ čo JSP verzia preukázala stabilnejší výkon. Z hľadiska bezpečnosti a rozširiteľnosti však Spring Boot ponúka modernejší a efektívnejší prístup. Na základe získaných výsledkov sme formulovali odporúčania pre výber vhodnej technológie pri tvorbe webových aplikácií v závislosti od rozsahu projektu a požiadaviek.

**Kľúčové slová:** Informatika, JavaServer Pages, Java Spring Boot, Webové aplikácie, Plánovanie udalostí.

## ABSTRACT

Antala, Jakub: Comparison of Java Technologies in Web Application Development. [Bachelor Thesis]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences and Informatics. Supervisor: Mgr. Dávid Držík. Degree of Qualification: Bachelor of Applied Informatics. Nitra: FNSaI, 2025. .... 55 p.

In this bachelor thesis, we focused on comparing two approaches to web application development in the Java programming language, specifically JavaServer Pages (JSP) and the Spring Boot framework. The aim of the thesis was to design, implement, and test event and reservation planner applications for the corporate sector, with each application being developed independently using the respective technology. The theoretical part of the thesis focused on analyzing both technologies in terms of architecture, development approach, and security capabilities. In the practical part, we described the database design, the implementation of individual application functionalities, and the actual development process in both technologies. Both versions of the application were subsequently tested with a focus on code quality, performance, security, and support from the developer community. The testing results showed that the application developed with Spring Boot offered better code readability and a more structured design, while the JSP version demonstrated more stable performance. From a security and scalability perspective, however, Spring Boot provides a more modern and efficient approach. Based on the results obtained, we formulated recommendations for choosing the appropriate technology for web application development depending on the project scope and requirements.

**Keywords:** Informatics, JavaServer Pages, Java Spring Boot, Web applications, Event planning

# OBSAH

<b>Úvod.....</b>	<b>10</b>
<b>1 Analýza súčasného stavu .....</b>	<b>11</b>
1.1 Prehľad a história jazyka Java .....	11
1.1.1 Hlavné verzie a ich vylepšenia.....	11
1.2 Význam Java v súčasnom IT sektore.....	12
1.3 Ekosystém Java a vývojové nástroje.....	12
1.3.1 Java Development Kit .....	13
1.3.2 Podporný nástroj Hibernate v prostredí Java .....	14
1.4 JavaServer Pages.....	14
1.4.1 Úvod do technológie JavaServer Pages.....	14
1.4.2 Zabezpečenie v JavaServer Pages .....	15
1.4.3 Optimalizácia výkonu pomocou delenia JSP .....	16
1.4.4 JavaServer Faces .....	17
1.5 Java Spring framework.....	17
1.5.1 História Java Spring .....	17
1.5.2 Úvod do Java Spring .....	18
1.5.3 Bezpečnosť v Java Spring.....	18
1.6 Java Spring Boot.....	19
1.6.1 Kľúčové vlastnosti nástroja Spring Boot.....	19
1.6.2 Actuator a Micrometer .....	20
1.6.3 Bezpečnosť v Spring Boot.....	20
1.7 Experiment pri tvorbe eshopu.....	21
<b>2 Ciele záverečnej práce .....</b>	<b>23</b>
<b>3 Návrh a metodika .....</b>	<b>24</b>
3.1.1 Architektúra systému.....	25
3.1.2 Návrh Databázy.....	25
3.1.3 Funkcionalita aplikácie.....	27
3.1.4 Používateľské rozhranie .....	28
3.2 Vývoj Aplikácie v JavaServer Pages.....	29
3.2.1 Vývojové prostredie .....	29
3.2.2 Implementácia prihlasovania v JSP.....	30
3.2.3 Implementácia prístupu k funkcionalitám aplikácie.....	32
3.2.4 Výpis HTML obsahu na stránku .....	33
3.2.5 Výpis eventov .....	34



3.2.6	Implementácia rezervácie eventov .....	35
3.2.7	Operácie nad Eventami.....	37
3.2.8	Hotely a hodnotenia .....	38
3.3	Vývoj aplikácie v Java Spring boot.....	39
3.3.1	Vývojové prostredie .....	39
3.3.2	Implementácia prihlasovania v Java Spring Boot .....	40
3.3.3	Implementácia prístupu k funkcionalitám aplikácie.....	41
3.3.4	Thymleaf v Spring Boot .....	42
3.3.5	Výpis eventov pomocou Java Spring Boot.....	42
3.3.6	Zobrazenie detailov eventu .....	42
3.3.7	Implementácia rezervácie eventov v Spring Boot.....	43
3.3.8	Spravovanie eventov pomocou Spring Boot.....	44
3.3.9	Implementácia Hotelov a hodnotení v Spring Boot .....	45
<b>4</b>	<b>Výsledky.....</b>	<b>46</b>
4.1	Priebeh testovania aplikácií .....	46
4.2	Porovnanie technológií.....	46
4.2.1	Kvalita kódu.....	46
4.2.2	Testovanie výkonu .....	47
4.2.3	Testovanie bezpečnosti.....	48
4.2.4	Podpora komunity .....	49
	<b>Záver.....</b>	<b>51</b>
	<b>Zoznam bibliografických odkazov.....</b>	<b>53</b>
	<b>Zoznam príloh .....</b>	<b>55</b>

# ÚVOD

V súčasnosti predstavujú webové aplikácie kľúčový nástroj pre množstvo podnikov či verejných inštitúcií. Vývoj webových aplikácií je preto stále aktuálnou témou, ktorej sa venuje veľká časť vývojárskej komunity. Technológia Java patrí medzi najčastejšie používané programovacie jazyky pre tvorbu webových aplikácií. Medzi najpoužívanejšie patria JavaServer Pages v kombinácii so servletmi a moderný framework Spring Boot, ktorý sa postupom času častejšie uplatňuje práve pri vývoji komplexnejších webových aplikácií.

Pri výbere témy tejto bakalárskej práce sme sa rozhodli zamerať práve na porovnanie týchto dvoch možností vývoja aplikácií. Rozoberali sme, ako sa líšia z pohľadu implementácie a čo môže vývojárovi priniesť prístup cez JSP a servlety v porovnaní s moderným Spring Boot riešením. Ako námet na aplikáciu sme si zvolili plánovač udalostí s možnosťou registrácie používateľov, správy eventov a hodnotení hotelov. Tento typ aplikácie nám umožnil vytvoriť viacero funkčných častí ako je autentifikácia, autorizácia, CRUD operácie, práca s databázou a generovanie dynamického obsahu.

Cieľom práce je vytvoriť rovnakú aplikáciu pomocou oboch technológií a následne ich porovnať z pohľadu kvality kódu, výkonnosti, bezpečnosti a podpory vývojárskej komunity. Každú verziu aplikácie sme naprogramovali samostatne, pričom sme sa snažili zachovať rovnakú funkcionálnosť. Po dokončení vývoja nasledovalo testovanie aplikácií.

Pri vývoji sme čerpali z oficiálnej dokumentácie, rôznych technických blogov a diskusných fór, pričom hlavným informačným zdrojom bola komunita vývojárov a praktické príklady iných riešení. Tieto zdroje nám výrazne pomohli pri riešení problémov, ktoré vznikali počas implementácie.

Práca je rozdelená do viacerých kapitol. Prvá kapitola je venovaná teoretickému prehľadu technológií, ktoré sme v práci využili. Druhá kapitola je venovaná cieľom práce. V tretej kapitole podrobne rozoberáme návrh aplikácie a samotnú implementáciu v oboch technológiách. V záverečnej časti sa venujeme testovaniu, vyhodnoteniu výsledkov a porovnaniu oboch riešení.

# 1 ANALÝZA SÚČASNÉHO STAVU

Táto kapitola sa zaoberá súčasným stavom technológie Java, jej vývojom a využitím v oblasti vývoja webových aplikácií. Java je jedným z najpoužívanějších a najrozšírenejších programovacích jazykov, ktorý sa využíva v rôznych oblastiach vývoja softvéru, od desktopových, cez webové až po mobilné aplikácie. S rastúcimi požiadavkami na dynamické webové aplikácie a spracovanie veľkých objemov dát sa vyvinuli technologické nástroje ako JavaServer Pages (JSP) a Spring Framework, ktoré poskytujú efektívne a škálovateľné riešenia pre moderné aplikácie. Cieľom tejto analýzy je preskúmať tieto technológie, predstaviť ich základné charakteristiky a aplikácie v praxi a poskytnúť komplexnejší prehľad o možnostiach, ktoré ponúkajú pri vývoji webových aplikácií.

## 1.1 PREHLAD A HISTÓRIA JAZYKA JAVA

Java bola vyvinutá spoločnosťou Sun Microsystems (Sun Microsystems, 1982) a uvedená na trh v roku 1995. Na čele jej vývoju bol James Gosling, ktorý spolu so svojim tímom vytvoril jazyk pôvodne pod názvom *Oak*. Java bola navrhnutá s cieľom byť jednoduchým, objektovo-orientovaným, bezpečným a platformovo nezávislým jazykom. Tento jazyk umožnil aplikáciám bežať na akomkoľvek zariadení bez ohľadu na operačný systém. Vznikol preto pojem „*Napiš raz, spusti kdekoľvek*“ (anglicky Write once, run anywhere), ktorý znamenal, že napísaný kód bolo možné spustiť na platforme, ktorá podporovala Java Virtual Machine (Cosmina, 2022).

### 1.1.1 Hlavné verzie a ich vylepšenia

Java získala popularitu vďaka svojej presnosti a stabilite, čo jej pomohlo získať uplatnenie v podnikových aplikáciách. V nasledujúcich rokoch po jej vývoji prešla viacerými významnými aktualizáciami, pričom každá z nich priniesla nové funkcie a vylepšenia.

- Java 2 (1998): Zavedenie konceptu Java 2 platformy (J2EE), ktorého zameraním boli podnikové aplikácie.
- Java 5 (2004): Pridanie for-each cyklus a anotácie, čo zjednodušilo a vylepšilo čitateľnosť a zároveň bezpečnosť kódu.

- Java 8 (2014): Zlepšenie pracovania s funkcionálnym programovaním vďaka zavedeniu Lambda výrazov a stream-u API. Mnoho rokov bola Java 8 stabilnou verziou, ktorá bola široko používaná vo všetkých oblastiach vývoja.
- Java 9 a novšie verzie: Pridanie modulárneho systému pre lepšie organizovanie a správu veľkých aplikácií.

Po tom, ako spoločnosť Oracle (Oracle Corporation, 1977) v roku 2010 získala Sun Microsystems (Sun Microsystems, 1982), prevzala taktiež aj správu nad vývojom Javy. Od tohto roku pokračuje Oracle v aktualizáciách a ďalšom rozvoji tejto platformy (Cosmina, 2022).

## 1.2 VÝZNAM JAVA V SÚČASNOM IT SEKTORE

Java sa uplatňuje v širokom spektre IT odvetví, vďaka čomu patrí medzi najpoužívanejšie programovacie jazyky na trhu. Podnikové aplikácie sú oblasťou, kde Java EE (Enterprise Edition) zohráva kľúčovú úlohu. Java EE je platforma určená na tvorbu veľkých, bezpečných a škálovateľných aplikácií s viacvrstvovou architektúrou, ktorá umožňuje robustné riadenie transakcií. Webové systémy taktiež využívajú Javu EE na distribuované a viacvrstvové spracovanie. Komponenty klientskeho rozhrania bežia na strane klienta, zatiaľ čo webové a aplikačné vrstvy sú spracované na Java EE serveri. Tento prístup je ideálny pre webové aplikácie prepojené s podnikovo-informačnými systémami (Ardis et al., 2015).

Java sa široko využíva aj v mobilných aplikáciách pre Android, kde základom platformy je Java SE (Standard Edition). Vďaka nezávislosti na platforme a objektovo-orientovaným funkciám predstavuje Java silný základ pre väčšinu aplikácií na Android. Rovnako má Java významné uplatnenie v oblasti Internet vecí (anglicky Internet of Things, IoT), kde jej schopnosť pracovať s rôznymi zariadeniami zabezpečuje efektívnu správu a spracovanie dát v reálnom čase v rámci IoT sietí. Stabilita, všestrannosť a bezpečnosť robia z Javy kľúčový nástroj pre moderné riešenia spracovania komplexných dát (Ben Evans, 2015).

## 1.3 EKOSYSTÉM JAVA A VÝVOJOVÉ NÁSTROJE

Java patrí medzi najpoužívanejšie programovacie jazyky s komunitou rozšírenou po celom svete. Túto rozsiahlu komunitu tvorí viac než 10 miliónov vývojárov, čo ju radí hneď vedľa webových technológií, ako je JavaScript, a jazykov ako C a C++. Java je často využívaná pri vývoji bezpečných aplikácií, kde zohrávajú kľúčovú úlohu

spoľahlivosť a bezpečnosť. Vzhľadom na to, že je primárne zameraná na podnikové a komerčné využitie, môže byť komunita vývojárov Javy menej viditeľná v oblasti open-source (po slovensky softvér s otvoreným zdrojovým kódom) v porovnaní s inými jazykmi (Ben Evans, 2015).

Napriek tomu má Java významný ekosystém open-source projektov, ktoré podporuje množstvo vývojárov angažujúcich sa vo vývoji aj po pracovnej dobe. Komunitný proces Javy (anglicky Java Community Process, JCP), založený v roku 1998, slúži na formalizáciu a štandardizáciu technológií Javy prostredníctvom špecifikačných žiadostí pre Javu (anglicky Java Specification Requests, JSR). Tieto žiadosti vytvárajú pracovné skupiny, ktorých cieľom je vyvinúť špecifikáciu, testovaciu sadu a referenčnú implementáciu. Tento proces umožňuje Jave pružne reagovať na nové technológie a potreby (Ben Evans, 2015).

Java má tiež ekosystém nezávislých vývojárov, ktorí prispievajú k platforme mimo oficiálnych organizácií. V počiatočných rokoch sa snažili rozšíriť hranice možností platformy, čo viedlo k vzniku nezávislých projektov, ktoré sú aktívne dodnes. Po sprístupnení platformy ako open-source mnohí vývojári naďalej pracujú mimo oficiálnych projektov, ako je OpenJDK, čím Javu obohacujú o nové funkcie a rozšírenia, ktoré môžu byť implementované v budúcich verziách (Ben Evans, 2015).

### **1.3.1 Java Development Kit**

Java Development Kit (JDK) predstavuje základný nástroj na vývoj aplikácií v jazyku Java. Jednou z jeho kľúčových súčastí je kompilátor, ktorý prekladá zdrojový kód do bajtkódu – platformovo nezávislého kódu, ktorý dokáže vykonávať virtuálny stroj Java (anglicky Java Virtual Machine, JVM). Tento proces umožňuje aplikáciám napísaným v Jave bežať na rôznych operačných systémoch bez potreby úprav kódu, čím zabezpečuje vysokú prenositeľnosť softvéru. Ďalšou podstatnou zložkou JDK je samotný JVM, ktorý interpretuje bajtkód a zaručuje konzistentné správanie aplikácií naprieč rôznymi platformami (Gößner et al., 2004).

Okrem kompilátora a virtuálneho stroja Java obsahuje JDK aj rozsiahlu zbierku knižníc a rozhraní pre programovanie aplikácií (anglicky Application Programming Interface, API). Tieto knižnice poskytujú predpripravený kód pre riešenie bežných programátorských úloh, ako sú dátové štruktúry, sieťová komunikácia a grafické používateľské rozhrania (anglicky Graphical User Interfaces, GUIs). Používanie týchto knižníc uľahčuje vývoj aplikácií, pretože znižuje potrebu programátorov písať kód od

základov, čo zvyšuje ich produktivitu a kvalitu výsledného kódu. Vďaka týmto prínosom sa Java Development Kit považuje za nevyhnutný nástroj pre každého vývojára v jazyku Java (Gößner et al., 2004).

### **1.3.2 Podporný nástroj Hibernate v prostredí Java**

Hibernate je jedným z najvýznamnejších nástrojov v prostredí Java, najmä pre podnikové aplikácie. Ide o objektovo-relačný mapovací nástroj (anglicky Object-Relational Mapping, ORM), ktorý umožňuje efektívne mapovať objekty na databázové tabuľky, čím uľahčuje prácu s relačnými databázami v Java aplikáciách. Vďaka Hibernate už nie je potrebné písať komplexný SQL kód na základné operácie, ako sú vloženie, načítanie alebo aktualizácia dát; namiesto toho umožňuje vývojárom priamo pracovať s objektmi Javy. Hibernate automatizuje proces ukladania stavu objektov do databázy, čo výrazne zjednodušuje vývoj veľkých aplikácií. Tento nástroj je často používaný v kombinácii s rámcom Spring, s ktorým umožňuje efektívne riadiť transakcie a spravovať vzťahy medzi entitami. Hibernate tak prispieva k vývoju škálovateľných a ľahko udržiavateľných aplikácií, čo je v podnikovom prostredí kľúčové (Joseph B. et al., 2022).

## **1.4 JAVASERVER PAGES**

JavaServer Pages (JSP) je dôležitá technológia v Java ekosystéme, určená na vývoj dynamických webových stránok, ktorá umožňuje efektívne vkladanie Java kódu priamo do hypertextového značkovacieho jazyka (anglicky HyperText Markup Language, HTML). Vďaka JSP môžu vývojári jednoducho vytvárať webové aplikácie s flexibilným obsahom a robustnou logikou na strane servera, čím sa zvyšuje efektivita a udržiateľnosť moderných webových aplikácií (Juneau, 2014).

### **1.4.1 Úvod do technológie JavaServer Pages**

Technológiu JavaServer Pages (JSP) predstavila spoločnosť Sun Microsystems v roku 1999 (Sun Microsystems, 1982). JSP funguje ako nadstavba nad Java Servlet API, čo umožňuje vývojárom vkladať Java kód priamo do HTML stránok pomocou značiek a skriptletov. Tento prístup výrazne zvýšil produktivitu vývoja webu v Jave, pretože kombinuje možnosti jazyka Java s jednoduchou syntaxou HTML. Flexibilita JSP umožňuje prepájať statický obsah v jazykoch XML a HTML s dynamickými prvkami na báze Javy, čo robí JSP vhodným nástrojom pre vývoj responzívnych webových aplikácií (Juneau, 2014).

S postupným vývojom sa technológia JSP prispôsobila tak, aby jasne oddeľovala obchodnú logiku od prezentačnej vrstvy. Pôvodne bol Java kód vkladaný priamo do HTML pomocou skriptletov, no moderné aplikácie často využívajú JavaBeans a Expression Language (EL), čo umožňuje vytvoriť štruktúru podľa architektúry Model-View-Controller (MVC) pre lepšiu udržateľnosť (Juneau, 2014).

Podľa (Ramirez-Noriega et al., 2020) je MVC návrhový vzor, ktorý rozdeľuje aplikáciu na tri hlavné komponenty: Model, View a Controller. Model spravuje dáta a aplikačnú logiku, View zobrazuje informácie používateľovi a Controller sprostredkúva interakciu medzi modelom a pohľadom. Toto rozdelenie umožňuje vývojárom sústrediť sa na jednotlivé vrstvy aplikácie, čo zvyšuje udržiavateľnosť kódu a jeho prehľadnosť. JSP podporuje viacero základných značiek, ako napríklad `<jsp:useBean>`, `<jsp:setProperty>` a `<jsp:getProperty>`, ktoré umožňujú prácu s JavaBeans a spracovanie dynamických dát. Životný cyklus JSP je podobný životnému cyklu servletov – pri prvom spustení sa JSP súbory preložia do servletových tried, čo zjednodušuje nasadzovanie a aktualizáciu webových aplikácií (Juneau, 2014).

#### **1.4.2 Zabezpečenie v JavaServer Pages**

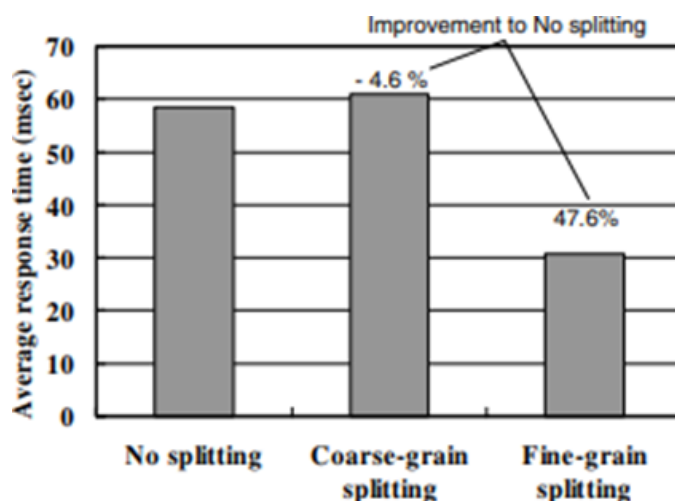
JSP ponúka rôzne bezpečnostné mechanizmy na ochranu webových aplikácií pred potenciálnymi hrozbami a na kontrolovaný prístup k citlivým údajom. Jedným z hlavných bezpečnostných prvkov je deklaratívne zabezpečenie, ktoré umožňuje definovať prístupové obmedzenia a role používateľov pomocou popisovačov alebo anotácií. Tento typ zabezpečenia umožňuje vývojárom presne nastaviť, ktorí používatelia majú prístup k určitým URL alebo funkciám aplikácie, a to bez nutnosti zasahovať do zdrojového kódu. Deklaratívne zabezpečenie je ideálne pre situácie, kde sa požiadavky na bezpečnosť môžu meniť, pretože umožňuje tieto úpravy bez zásahov priamo v kóde aplikácie (Jendrock, 2013).

Ďalším dôležitým prvkom bezpečnosti v JSP je programatické zabezpečenie, ktoré ponúka vysokú mieru flexibility. Tento typ zabezpečenia je priamo vložený do kódu aplikácie, čo dáva vývojárom možnosť dynamicky riadiť autentifikáciu, autorizáciu a správanie aplikácie podľa konkrétnych podmienok. Ak sa požiadavky na prístup do rôznych častí aplikácie menia, zabezpečenie umožňuje prispôbovať pravidlá prístupu. Tento prístup je ideálny pre zložité aplikácie, ktoré vyžadujú detailnú kontrolu prístupových práv (Oracle, 2013).

Zabezpečenie zdrojov protokolu pre prenos hypertextu (anglicky HyperText Transfer Protocol, HTTP) je ďalším kľúčovým prvkom bezpečnosti v JSP. Pomocou obmedzení na úrovni HTTP môžu vývojári riadiť prístup k určitým HTTP metódam, ako sú napríklad GET, POST, PUT alebo DELETE, na základe používateľských rolí. Tento prístup umožňuje definovať, ktorí používatelia môžu vykonávať špecifické akcie na zadaných URL adresách. Napríklad, na základe rolí používateľov je možné nastaviť, že len administrátori majú prístup k určitým operáciám, čím sa zabráni neoprávnenému prístupu a neautorizovaným akciám. Takéto obmedzenie výrazne zvyšuje celkovú bezpečnosť aplikácie (Goncalves, 2013).

### 1.4.3 Optimalizácia výkonu pomocou delenia JSP

Využitie techniky delenia JSP stránok (anglicky JSP splitting) predstavuje efektívny spôsob, ako zvýšiť výkon JSP v prostrediach s vysokou záťažou. Táto metóda umožňuje rozdeliť stránky JSP na menšie fragmenty, ktoré je možné nezávisle ukladať do vyrovnávacej pamäte, čo optimalizuje spracovanie obsahu na strane servera. Ak sa jednotlivé časti stránky menia v rôznych intervaloch, delenie umožňuje ukladať do pamäte iba nemenné fragmenty, zatiaľ čo dynamické časti sa môžu aktualizovať podľa potreby. Tento prístup nielen skracuje odozvu stránky, ale aj šetrí serverové zdroje tým, že minimalizuje opakované spracovanie statických častí (Nakaike et al., 2004).



Obrázok 1 JSP Splitting<sup>1</sup>

Pre účinnú implementáciu tejto techniky bol vyvinutý nástroj na delenie JSP stránok ako plugin pre prostredie Eclipse IDE, ktorý automatizuje proces delenia a analyzuje dátové závislosti medzi fragmentmi. Tým zabezpečuje konzistentné

<sup>1</sup> Zdroj Obrázok 1: <https://doi.org/10.1109/SAINT.2004.1266106>



fungovanie celej stránky aj po jej rozdelení. Experimenty preukázali výrazné zlepšenie výkonnosti (zobrazené na obrázku nižšie), čo robí z delenia JSP užitočný nástroj pri vývoji škálovateľných webových aplikácií (Nakaike et al., 2004).

#### **1.4.4 JavaServer Faces**

JavaServer Faces (JSF) je silný webový rámec navrhnutý na základe architektúry Model-View-Controller (MVC), ktorá umožňuje jasne oddeliť aplikačnú logiku, zobrazovanie a interakciu s používateľom. Cieľom JSF je zjednodušiť vývoj webových aplikácií v jazyku Java a poskytnúť vývojárom účinné nástroje na správu komplexných používateľských rozhraní. Jednou z kľúčových výhod JSF je rozsiahla knižnica komponentov. Táto knižnica obsahuje dopredu definované prvky používateľského rozhrania, ktoré sú znovu použiteľné, čím výrazne zlepšujú používateľskú skúsenosť.

JSF umožňuje efektívne prepojenie jednotlivých komponentov s časťou servera (anglicky backendom) prostredníctvom spravovaných objektov (anglicky managed beans) a funkcií kontextov a vkladania závislostí (anglicky Contexts and Dependency Injection, CDI), čo uľahčuje prácu s dátami v rámci aplikácie. Súčasťou JSF sú aj spojené nástroje na validáciu a konverziu používateľských vstupov, ktoré poskytujú flexibilitu pri spracovaní dát. Životný cyklus JSF aplikácií vývojárom umožňuje riadenie všetkých fáz – od prijatia požiadavky až po vykreslenie odpovede – čo zabezpečuje vysokú mieru kontroly nad správaním aplikácie a jej fungovaním. Vďaka týmto vlastnostiam je JSF skvelým nástrojom na vývoj podnikového softvéru, kde patria škálovateľnosť, modulárnosť a bezpečnosť medzi kľúčové požiadavky (Li a Sun, 2011).

### **1.5 JAVA SPRING FRAMEWORK**

Java Spring je rámec (anglicky framework) určený na vývoj softvéru, ktorý zjednodušuje správu závislostí a umožňuje flexibilnú prácu s aplikáciami. Pomocou techník ako inverzia riadenia (anglicky Inversion of Control, IoC) a vkladanie závislostí (anglicky Dependency Injection, DI) uľahčuje vytváranie modulárnych a škálovateľných aplikácií, čo z neho robí obľúbený nástroj pre moderné projekty v Jave (Gajewski a Zabierowski, 2019).

#### **1.5.1 História Java Spring**

Vznik Java Spring Frameworku sa viaže na rok 2002 kedy vznikol ako odpoveď na komplexnosť tradičných Java2EE aplikácií. Rod Johnson vo svojej knihe Expert One-on-One J2EE Design and Development (Johnson, 2006) predstavil približne 30 000

riadkov kódu, ktoré popisovali základné koncepty ako kontajner pre inverziu riadenia (anglicky Inversion of Control, IoC) a skorú verziu Spring MVC. Spočiatku váhal, či by mal venovať čas rozvoju open-source projektu, no vďaka povzbudeniu od čitateľov sa rozhodol pokračovať. Práve jeden z jeho čitateľov Yann Caroff navrhol názov „Spring“, ktorý mal symbolizovať nový začiatok po zložitom období Java2EE. V júni roku 2003 bol projekt oficiálne uvedený, čo viedlo k vydaniu Spring 1.0 (Johnson, 2002).

### **1.5.2 Úvod do Java Spring**

Spring Framework bol navrhnutý s dôrazom na flexibilnú a modulárnu architektúru, ktorá umožňuje vývojárom efektívne spravovať aj zložité podnikové aplikácie. Táto architektúra je rozdelená na samostatné moduly, ktorých každý rieši konkrétnu funkciu aplikácie. Vďaka tomu si môžu vývojári vybrať iba tie komponenty, z ktorých potrebujú, a tak zachovať prehľadnosť a vyššiu efektívnosť kódu. Napríklad modul Spring Data JPA poskytuje podporu pre prácu s relačnými databázami a umožňuje jednoducho vykonávať základné operácie či vytvoriť vlastné databázové dotazy. Táto možnosť je obzvlášť užitočná pri práci s veľkými a komplexnými databázami (Acetozi, 2017).

### **1.5.3 Bezpečnosť v Java Spring**

Java Spring Framework kladie veľký dôraz na bezpečnosť aplikácií a poskytuje výkonný modul Spring Security, navrhnutý na ochranu webových aj podnikových aplikácií v prostredí Java. Tento modul umožňuje komplexné riešenie pre autentifikáciu, autorizáciu a správu prístupových práv, pričom jeho flexibilná architektúra pomáha vývojárom prispôsobiť bezpečnostné nastavenia na viacerých úrovniach – od zabezpečenia jednotlivých URL adries až po detailné riadenie prístupu k obchodným objektom pomocou Access Control Lists (ACLs) (Scarioni a Nardone, 2019).

Spring Security podporuje rôzne autentifikačné metódy, ako sú LDAP, OAuth 2.0 a X.509 certifikáty, čo zvyšuje jeho univerzálnosť a umožňuje integráciu s inými systémami a bezpečnostnými protokolmi. Architektúra Spring Security je zároveň navrhnutá tak, aby umožnila rozšírenie a prispôbenie podľa konkrétnych bezpečnostných požiadaviek aplikácie. Princípy Inversion of Control (IoC) a Dependency Injection (DI) uľahčujú integráciu bezpečnostných komponentov do rôznych častí aplikácie, čím zlepšujú ochranu pred neoprávneným prístupom a ďalšími potenciálnymi rizikami. Tieto bezpečnostné vlastnosti robia zo Spring Frameworku robustný nástroj na

ochranu moderných aplikácií, ktorý je flexibilný a prispôsobiteľný pre rôzne typy podnikového aj individuálneho použitia (Scarioni a Nardone, 2019).

## **1.6 JAVA SPRING BOOT**

Spring Boot je navrhnutý tak, aby vývoj aplikácií v Jave zjednodušil minimalizovaním potrebnej konfigurácie. Vývojári sa tak môžu sústrediť priamo na biznis logiku a funkcionality aplikácie, zatiaľ čo technické detaily konfigurácie a nasadenia sú automatizované. Vďaka vstavaným prednastaveniam a štartovacím modulom (starters) je možné vytvárať aplikácie pripravené na produkciu bez rozsiahlych úprav. Tento prístup robí Spring Boot ideálnym nástrojom pre moderné aplikácie, ktoré vyžadujú rýchlosť a flexibilitu, ako sú cloudové alebo mikroservisné architektúry (Gutierrez, 2019).

### **1.6.1 Kľúčové vlastnosti nástroja Spring Boot**

Spring Boot prináša viacero kľúčových funkcií, ktoré zásadne zjednodušujú vývoj a nasadenie aplikácií v Jave. Jednou z hlavných výhod je možnosť vytvárať samostatne spustiteľné aplikácie s integrovanými servermi, ako sú Tomcat, Netty alebo Jetty. Vďaka tejto funkcionalite nie je potrebné konfigurovať a nasadzovať externý server, čo uľahčuje nasadenie aplikácie a zároveň zvyšuje jej presnosť a spoľahlivosť (Gutierrez, 2019).

Ústrednú úlohu pri spustení aplikácie zohráva trieda `SpringApplication`, ktorá poskytuje jednoduché nástroje na inicializáciu a konfiguráciu. Tento prístup uľahčuje vývojárom sústrediť sa viac na samotnú logiku aplikácie než na technické nastavenie prostredia a servera. Spring Boot zároveň ponúka flexibilné možnosti konfigurácie pomocou súborov `.properties` alebo `.yaml`, čo umožňuje ľahké prispôbenie aplikácie rôznym prostrediam. Každé prostredie môže mať vlastné nastavenia, čo zjednodušuje údržbu a zlepšuje prispôsobivosť bez nutnosti zasahovať priamo do kódu (Gutierrez, 2019).

Ďalšou výhodou Spring Bootu je eliminácia potreby XML konfigurácií. Pomocou anotácií a prednastavených hodnôt výrazne redukuje komplexnosť prípravy prostredia, čím zjednodušuje údržbu kódu a zvyšuje jeho čitateľnosť. Navyše, Spring Boot obsahuje aj príkazové rozhranie (Command-Line Interface, CLI), ktoré umožňuje rýchlu tvorbu a spustenie jednoduchých prototypov prostredníctvom skriptov v Groovy alebo priamo v Jave. Toto rozhranie je skvelým nástrojom na rýchle overenie konceptov (Gutierrez, 2019).

### 1.6.2 Actuator a Micrometer

Spring Boot obsahuje modul Actuator, ktorý poskytuje účinné nástroje na monitorovanie a správu aplikácií. Tento modul sprístupňuje kľúčové metriky a diagnostické údaje, ktoré vývojárom umožňujú detailne sledovať stav a výkon aplikácie, ako aj jej vnútorné procesy. Vďaka integrácii s platformou Micrometer je možné metriky zaznamenávať nezávisle od platformy a prepojiť ich s rôznymi nástrojmi na monitorovanie, ako sú Prometheus, Grafana alebo New Relic. Actuator tak ponúka jednoduchý spôsob, ako monitorovať výkon aplikácie a stav služieb cez REST API, čo výrazne uľahčuje prepojenie s externými monitorovacími systémami (Gutierrez, 2019).

Táto funkcionálna je obzvlášť cenná pre mikroservisné aplikácie, ktoré si vyžadujú podrobné sledovanie, vysokú dostupnosť a flexibilitnú správu. Actuator umožňuje vývojárom zabudovať monitorovacie nástroje priamo do aplikácie už počas vývoja, čím sa znižuje potreba ďalšej konfigurácie alebo použitia externých nástrojov. Modul poskytuje prehľad o využití centrálnej procesorovej jednotky (anglicky Central Processing Unit, CPU) a pamäte, sleduje stav vlákien (anglicky threads) a monitoruje oneskorenie (anglicky latency). Týmto spôsobom Actuator výrazne prispieva k stabilite a výkonnosti aplikácií postavených na Spring Boote (Gutierrez, 2019).

### 1.6.3 Bezpečnosť v Spring Boot

Podobne ako Java Spring, aj Spring Boot obsahuje robustnú podporu pre zabezpečenie aplikácií vďaka modulu Spring Security. Tento modul pomáha vývojárom jednoducho implementovať autentifikáciu a autorizáciu, pričom poskytuje ochranu proti bežným útokom, ako sú Cross-Site Request Forgery (CSRF) a Cross-Origin Resource Sharing (CORS). Ponúka tiež možnosť autentifikácie cez rôzne mechanizmy, napríklad:

- LDAP - umožňuje autentifikáciu a správu používateľov cez adresovú službu,
- OAuth - poskytuje bezpečný spôsob autorizácie medzi aplikáciami bez zdieľania hesiel,
- a HTTP Basic - ktorý zabezpečuje prístup cez základnú autentifikáciu pomocou používateľského mena a hesla zaslaných v každej požiadavke.

Základné zabezpečenie je možné dosiahnuť pridaním závislosti spring-boot-starter-security, ktorá automaticky aplikuje jednoduchú autentifikáciu. Spring Boot zároveň umožňuje pokročilé prispôsobenie bezpečnostných nastavení. Vývojári môžu konfigurovať používateľské účty a oprávnenia prostredníctvom súboru

application.properties alebo vytvoriť vlastné konfigurácie pomocou triedy *WebSecurityConfigurerAdapter* (Gutierrez, 2018).

Pre zložitejšie autentifikačné prípady, kde je potrebná integrácia s existujúcou databázou, Spring Boot podporuje aj autentifikáciu pomocou JDBC. Táto možnosť umožňuje zabezpečenie prístupu do aplikácie prepojením s inými autentifikačnými službami. Ako príklad možno uviesť implementáciu OAuth2 autorizácie, ktorá umožňuje prihlásenie cez poskytovateľov ako Google alebo GitHub. Tento prístup zvyšuje úroveň ochrany a súčasne zjednodušuje správu používateľov v moderných webových aplikáciách (Gutierrez, 2018).

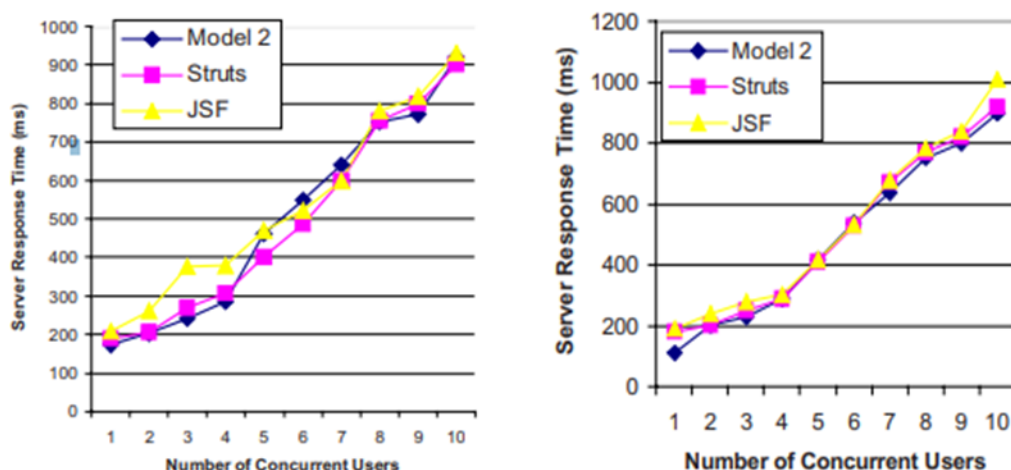
## 1.7 EXPERIMENT PRI TVORBE ESHOPU

V rámci experimentu autori porovnávali tri dizajnové modely – Java Model 2, Struts a JavaServer Faces – na príklade aplikácie online obchodu BuyDirect, ktorá umožňuje vyhľadávanie produktov, prezeranie produktov a dokončenie nákupu s transakciou v databáze. Cieľom bolo posúdiť jednoduchosť vývoja a výkon každého modelu. Hodnotenie jednoduchosti vývoja sa uskutočnilo porovnaním počtu riadkov kódu a počtu tried v aplikáciách. Výsledky ukázali (môžeme vidieť v tabuľke nižšie), že Model 2 vyžadoval najväčší počet riadkov a tried, čo znamená vyššiu zložitosť vývoja, zatiaľ čo JSF mal najnižšie hodnoty v oboch ukazovateľoch (Yu et al., 2004).

*Tabuľka 1 Počet Tried a riadkov kódu pre aplikácie*

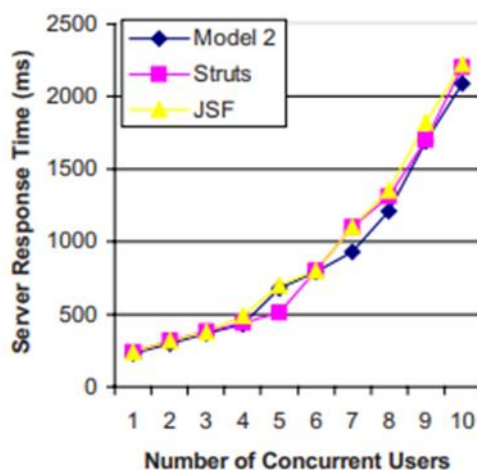
		<b>Model 2</b>	<b>Struts</b>	<b>JSF</b>
<b>Servlet</b>	#Classes	1	0	0
	#Lines	74	0	0
<b>Bean</b>	#Classes	9	9	9
	#Lines	348	348	348
<b>JSP</b>	#Classes	9	9	9
	#Lines	809	733	534
<b>Others</b>	#Classes	12	10	3
	#Lines	590	430	271
<b>Total</b>	#Classes	31	28	21
	#Lines	1821	1511	1153

Výkonnostné testy sa realizovali pomocou nástroja JMeter, ktorý simuloval rôzny počet používateľov od 1 do 10. Pri operácií vyhľadávania (Obrázok 2, vľavo) a prezerania produktov (Obrázok 2, vpravo) dosiahli Model 2 a Struts podobný výkon (môžeme vidieť na obrázkoch nižšie), zatiaľ čo JSF vykazoval mierne vyšší čas odozvy kvôli náročnosti spracovania komponentov (Yu et al., 2004).



Obrázok 2 Operácia vyhľadávania a prezerania

Pri nákupnej operácii, ktorá zahŕňala transakciu v databáze, Model 2 opäť vykazoval najlepšie výsledky (Obrázok 3), zatiaľ čo JSF zaostával z dôvodu vyššieho preťaženia pri rastúcom počte používateľov.



Obrázok 3 Operácia nákupu<sup>2</sup>

Tento experiment ukázal, že hoci Model 2 poskytuje najlepší výkon, je zložitejší na implementáciu, zatiaľ čo JavaServer Faces zjednodušuje vývoj s miernym poklesom výkonu pri vyššom zaťažení (Yu et al., 2004).

<sup>2</sup> Zdroj Obrázok 2, Obrázok 3: <https://doi.org/10.1007/s11761-023-00364-w>

## 2 CIELE ZÁVEREČNEJ PRÁCE

Cieľom bakalárskej práce je porovnať a analyzovať dve technológie JavaServer Pages (JSP) a Java Spring, pri vývoji webovej aplikácie plánovača udalostí a rezervácií pre podnikový sektor. Súčasťou cieľa je aj návrh databázy v MySQL, ktorá bude slúžiť ako úložisko pre používateľské údaje, eventy, hotely, rezervácie a hodnotenia. Ďalším cieľom práce je otestovať a porovnať obidve riešenia z pohľadu výkonnosti, bezpečnosti, kvality kódu a dostupnosti dokumentácie a komunity. Výsledkom porovnania bude odporúčanie, v akých prípadoch je vhodné zvoliť jednotlivé technológie pri vývoji webových aplikácií v jazyku Java.

### **Podciele:**

- analyzovať hlavné vlastnosti a rozdiely medzi technológiami JavaServer Pages a Java Spring,
- navrhnuť architektúru webovej aplikácie plánovača udalostí a rezervácií,
- definovať všetky požiadavky aplikácie,
- vyvinúť aplikáciu v JavaServer Pages a Java Spring,
- otestovať aplikácie z hľadiska výkonnosti, kvality kódu a bezpečnosti,
- vyhodnotiť výsledky testovania oboch technológií,
- na základe výsledkov testovania odporučiť vhodnú technológiu.

### 3 NÁVRH A METODIKA

Pre dosiahnutie cieľa práce sme najprv realizovali analýzu dvoch technológií v Java – JavaServer Pages a Java Spring. Tieto technológie boli analyzované na základe ich hlavných vlastností, rozdielov, bezpečnosti a aplikovateľnosti v oblasti vývoja podnikových webových aplikácií. Analýza zahŕňala štúdium dostupných zdrojov, dokumentácie a štúdií, so zameraním na aspekty ako kvalita kódu, bezpečnosť, výkon a podpora komunity.

Na základe tejto analýzy bola navrhnutá architektúra webovej aplikácie v JSP a Java Spring s dôrazom na funkčnosť a používateľskú prívetivosť. V tejto fáze bola definovaná štruktúra aplikácie, jej komponenty a ich vzájomné vzťahy, ako aj kľúčové funkcie pre podnikový sektor. Boli stanovené funkčné požiadavky aplikácie:

- spravovanie udalostí,
- autentifikácia používateľov,
- prezeranie udalostí,
- spravovanie používateľských rolí,
- spravovanie profilov používateľov,
- prezeranie hotelov,
- hodnotenia hotelov.

Stanovené boli aj nefunkčné požiadavky, medzi ktoré patria:

- bezpečnosť,
- výkon,
- použiteľnosť.

V ďalšej fáze sme sa zamerali na samotný vývoj aplikácií v dvoch verziách – jednu v JavaServer Pages a druhú v Java Spring. Pri implementácii boli aplikované poznatky zhrnuté v predchádzajúcich fázach.

Následne bolo vykonané testovanie oboch verzií aplikácie. Toto testovanie zahŕňalo analýzu výkonnosti, kvality kódu a bezpečnosti v reálnych podmienkach.

Na záver boli vyhodnotené výsledky testovania, čím bol získaný komplexný prehľad o výhodách a nevýhodách oboch technológií v kontexte podnikového využitia. Na základe získaných údajov bolo následne formulované odporúčanie pre výber vhodnej technológie pre plánovač udalostí a rezervácií v podnikovom sektore.



### 3.1.1 Architektúra systému

Pri návrhu systému sme sa rozhodli pre viacvrstvovú architektúru, ktorá zabezpečuje oddelenie jednotlivých zodpovedností a uľahčuje údržbu a ďalší rozvoj aplikácie. Táto architektúra je spoločná pre obe verzie implementácie – ako tú vytvorenú pomocou JSP a servletov, tak aj verziu vytvorenú v rámci frameworku Spring Boot.

Aplikácia je navrhnutá ako webová aplikácia s klientom v prehliadači a backendovým serverom, ktorý spracováva požiadavky. Klientská časť využíva štandardné HTML, CSS (vrátane Bootstrapu) a JavaScript v minimálnej miere, zatiaľ čo serverová časť je zodpovedná za spracovanie logiky a komunikáciu s databázou.

Z hľadiska logickej štruktúry aplikácie sme uplatnili nasledujúce vrstvy:

- Prezentačná vrstva – zabezpečuje zobrazenie údajov používateľovi. V prípade JSP verzie ide o HTML generované pomocou PrintWriter v servletoch, zatiaľ čo v Spring Boot verzii sa používajú šablóny Thymeleaf.
- Aplikačná logika (kontroléry / servlety) – tieto komponenty prijímajú požiadavky od používateľov, vyhodnocujú ich, volajú potrebné databázové operácie a vracajú odpovede. Práve tu sa realizuje kontrola rolí, správa session a rozhodovanie o tom, ktoré dáta budú zobrazené.
- Dátová vrstva – zodpovedá za komunikáciu s databázou. Vo verzii so servletmi túto vrstvu tvoria SQL dotazy priamo integrované v kóde servletov. V prípade Spring Boot verzie je táto vrstva oddelená pomocou @Repository rozhraní a Java Persistence API (JPA).

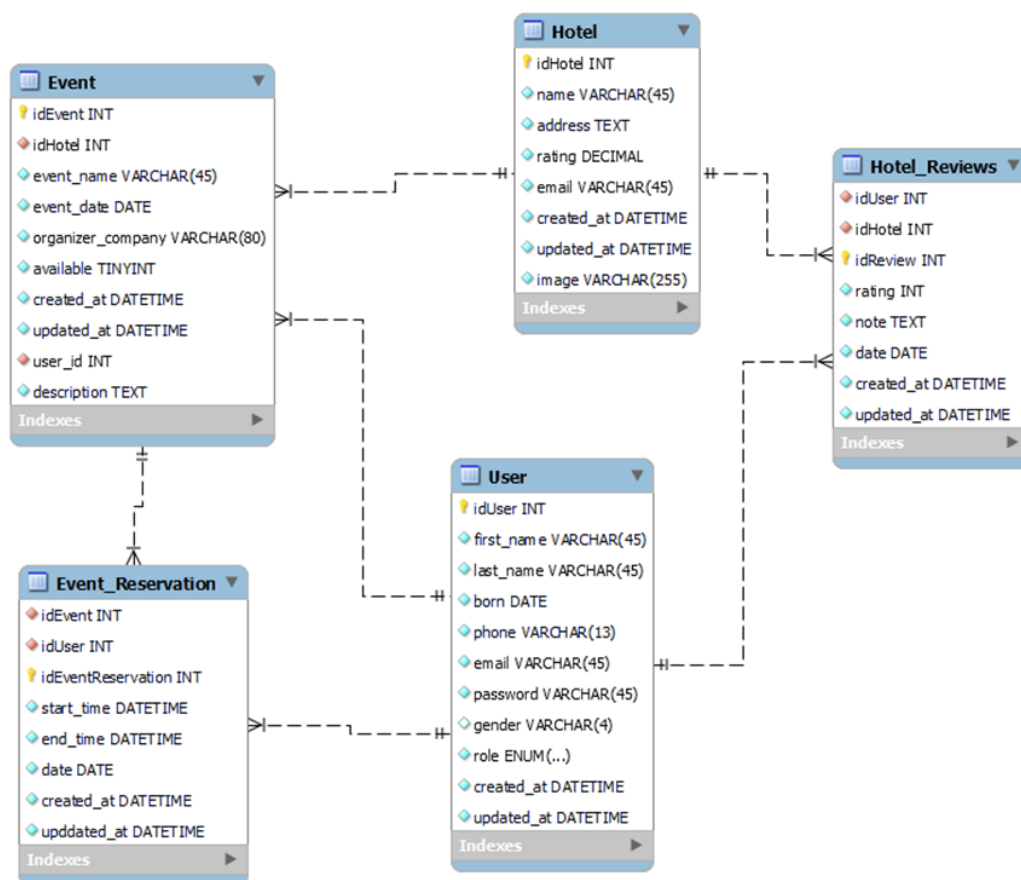
Rozhodnutie udržať túto architektonickú štruktúru konzistentnú medzi oboma verziami aplikácie nám umožnilo jednoduchšie porovnanie oboch technológií. Zároveň to uľahčilo implementáciu, keďže niektoré časti logiky bolo možné preniesť bez zásadných zmien.

### 3.1.2 Návrh Databázy

Pre účely vytvárania webovej aplikácie plánovača udalostí a rezervácií sme, vytvorili entitno-relačný databázový návrh, ktorý slúži ako centrálné úložisko všetkých údajov systému. Pri návrhu sme kladli dôraz na prehľadnosť, normalizáciu dát a jednoduchú rozšíriteľnosť, pričom sme vychádzali z požiadaviek kladených na aplikáciu z pohľadu používateľov, udalostí, rezervácií, hotelov a ich hodnotení. Ako databázové riešenie sme zvolili MySQL, nakoľko ide o stabilný a široko podporovaný databázový systém, ktorý zároveň umožňuje jednoduchú integráciu s backendovým

rozhraním vytvoreným v Jave, a je tak vhodnou voľbou pre podobné projekty. Databáza pozostáva z nasledujúcich hlavných tabuliek:

- user – uchováva informácie o registrovaných používateľoch vrátane mena, priezviska, emailu, hesla (uloženého vo forme hash), dátumu narodenia, telefónneho čísla a roly používateľa (guest, staff, admin).
- event – obsahuje údaje o jednotlivých udalostiach ako názov, dátum konania, popis, organizátor, dostupnosť miest, cudzí kľúč ID hotela, ktorý udalosť hostí a cudzí kľúč ID používateľa, ktorý udalosť vytvoril.
- event\_reservation – predstavuje prepojenie medzi používateľmi a udalosťami, ktoré si rezervovali. Okrem cudzieho kľúča ID používateľa a cudzieho kľúča ID udalosti obsahuje aj dátum, čas začiatku a konca rezervovanej udalosti.
- hotel – obsahuje informácie o hoteloch ako názov, adresa, hodnotenie a obrázok.
- hotel\_reviews – uchováva hodnotenia hotelov vytvorené používateľmi. Každé hodnotenie obsahuje textovú poznámku, číselné skóre od 0 do 5, dátum a ID hodnoteného hotela aj autora hodnotenia.



Obrázok 4 ER Diagram

Vzťahy medzi entitami sú definované pomocou cudzích kľúčov, čím sme zabezpečili integritu dát v celej databáze. Každý používateľ môže vytvoriť viacero udalostí, zarezervovať si viaceré z nich a ohodnotiť ľubovoľný hotel. Zároveň každá udalosť je naviazaná na konkrétny hotel a je spravovaná jedným používateľom. Takto navrhnutá databáza nám poskytla spoľahlivý základ pre realizáciu aplikačnej logiky a jej ďalšie rozširovanie.

### 3.1.3 Funkcionalita aplikácie

Funkcionalita aplikácie bola navrhnutá s ohľadom na požiadavky podnikovej sféry, kde je potrebné zabezpečiť efektívnu správu udalostí, možnosť ich prehľadného zobrazenia a jednoduchého prihlásenia sa na vybrané eventy. Základom systému je autentifikácia používateľov, pričom každý používateľ má pridelenú rolu určujúcu jeho autorizáciu v rámci aplikácie. Roly sú definované ako *guest*, *staff* a *admin*, pričom každá z nich má prístup k inému rozsahu funkcionalít.

Všetci používatelia majú možnosť prezerat' zoznam dostupných udalostí, pričom každá udalosť je zobrazená vo forme karty s názvom, dátumom, miestom konania a stručným popisom. Po kliknutí na detail udalosti sa zobrazia rozšírené informácie, ako sú organizačné údaje, popis podujatia, dostupnosť miest a možnosti interakcie. V prípade, že je udalosť dostupná, môže si ju používateľ zarezervovať. Rezervovaná udalosť sa následne zobrazí v osobitnom zozname „*Moje rezervácie*“, z ktorého ju môže používateľ aj zrušiť.

Používateľ s rolou *staff* má možnosť pridávať nové udalosti cez formulár, pričom každá udalosť musí byť naviazaná na konkrétny hotel. Okrem toho má oprávnenie upravovať alebo vymazať udalosti, ktoré sám vytvoril.

Používateľ s rolou *admin* má širšie oprávnenia – môže upravovať alebo vymazať akúkoľvek udalosť a zároveň má prístup aj k správe používateľov. V tejto časti aplikácie môže *admin* meniť roly ostatným používateľom prostredníctvom jednoduchého rozhrania.

Okrem správy udalostí aplikácia umožňuje aj interakciu s hotelmi. Používatelia si môžu prezerat' zoznam hotelov a po kliknutí na konkrétny hotel sa im zobrazia jeho detaily vrátane hodnotení od iných používateľov. Registrovaný používateľ má možnosť pridať vlastné hodnotenie hotela vo forme textovej poznámky a číselného skóre od 0 do 5. Tieto hodnotenia sú následne zobrazované v detailoch hotela a prispievajú k celkovému

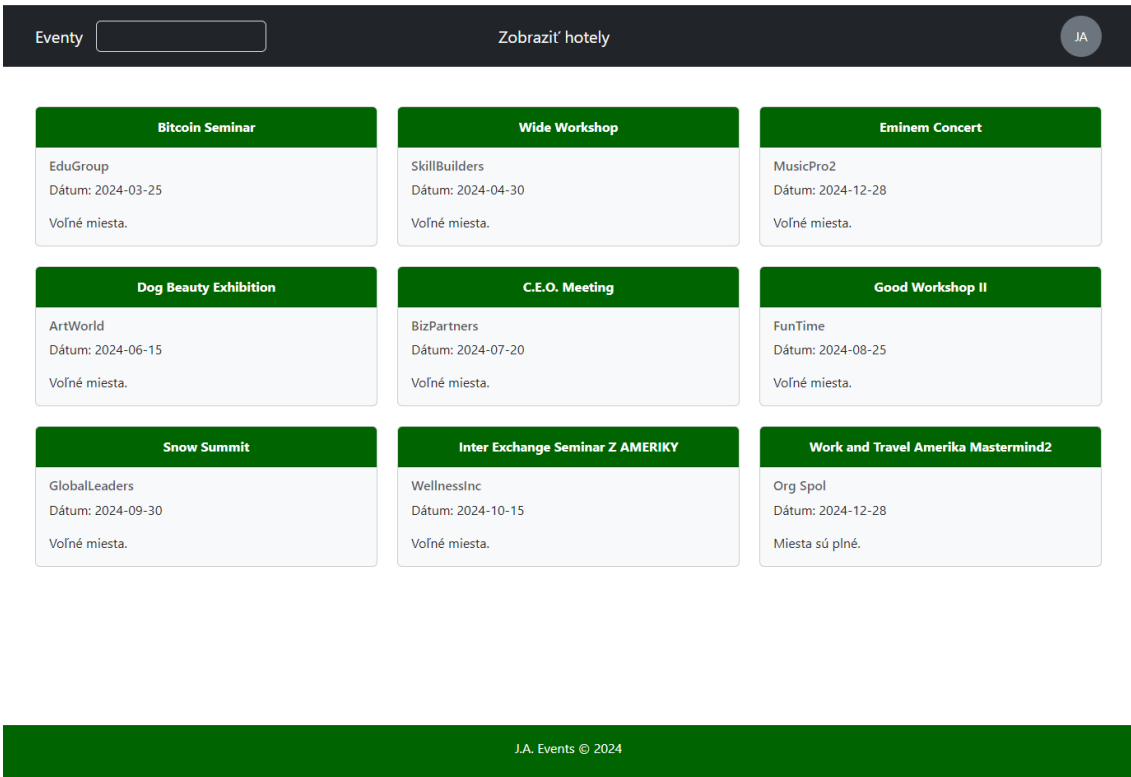
hodnoteniu kvality ubytovania, ktoré môže ovplyvniť rozhodovanie ďalších používateľov.

Navrhnutá funkcionálna vytvára základ pre intuitívnu a bezpečnú interakciu používateľa s aplikáciou. Všetky akcie sú dostupné v závislosti od oprávnení používateľa, pričom rozhranie dynamicky reaguje na to, či je používateľ prihlásený a akú rolu má pridelenú. Celý systém je navrhnutý modulárne, čo umožňuje jeho jednoduché rozšírenie o ďalšie funkcie v budúcnosti.

### 3.1.4 Používateľské rozhranie

Pri návrhu používateľského rozhrania aplikácie sme sa usilovali o vytvorenie prehľadného, responzívneho a vizuálne jednotného prostredia, ktoré bude intuitívne pre všetky typy používateľov bez ohľadu na ich rolu. Dôležitým cieľom bolo zabezpečiť, aby jednotlivé prvky aplikácie – ako napríklad výpis udalostí, prihlasovanie, registrácia či správa obsahu – boli jednoducho dostupné a zrozumiteľné.

Vďaka tomu že sa v našej práci nezaobráame samotným dizajnom ale funkčnosťou aplikácie, sme využili framework Bootstrap 5, ktorý nám umožnil rýchle vytváranie prvkov používateľského rozhrania, ako sú karty, formuláre, navigácia či tlačidlá.



Obrázok 5 Hlavná stránka aplikácie

Na obrázku (Obrázok 5) je zobrazené základné rozloženie stránky pozostáva z hlavičky (header), tela (body) a päty (footer), pričom všetky časti sú vizuálne zjednotené

pomocou manuálne definovaných CSS štýlov. Hlavička obsahuje navigáciu a profilové menu, ktoré sa dynamicky mení v závislosti od prihláseného používateľa a jeho roly. Táto vlastnosť zabezpečuje zobrazenie relevantných odkazov, napríklad na pridávanie udalostí alebo správu používateľských účtov.

Dôležitým prvkom rozhrania je aj zobrazovanie informácií vo forme „kartičiek“, ktoré používame pri výpise udalostí, hotelov a hodnotení. Každý typ dát má svoju štruktúru a štýl, čím sa zvyšuje prehľadnosť aplikácie.

Pri formulároch sme dbali na zrozumiteľné popisy, zreteľné rozloženie polí a zabezpečenie validácie vstupov, aby sa minimalizovalo riziko chýb pri zadávaní údajov. V prípade chýb alebo úspešných operácií sme zabezpečili zobrazovanie spätnej väzby priamo v rozhraní pomocou farebných hlások (napr. zelené pre úspech, červené pre chybu).

Všetky HTML súbory využívajú rovnaké základné štýly. Tieto štýly boli integrované priamo do súborov, aby sa zabezpečila jednotnosť vzhľadu bez závislosti od externých štýlových súborov.

## **3.2 VÝVOJ APLIKÁCIE V JAVASERVER PAGES**

V tejto podkapitole popisujeme jednotlivé kroky vývoja aplikácie pomocou technológie JSP.

### **3.2.1 Vývojové prostredie**

Pri vývoji aplikácie v JavaServer Pages (JSP) sme použili vývojové prostredie Eclipse IDE for Enterprise Java Developers, ktoré poskytuje podporu pre tvorbu Java EE projektov, vrátane JSP a servletov.

Pre lokálne nasadenie a testovanie aplikácie sme použili Webový server Tomcat 10.1. Tento nástroj nám slúžil ako webový kontajner, ktorý je schopný spracovávať požiadavky cez HTTP protokol, nasadzovať servletové komponenty a vykonávať JSP skripty. Tomcat sme zvolili vďaka jeho jednoduchšej konfigurovateľnosti a kompatibilitě s najnovšími verziami Javy.

Na komunikáciu medzi servletmi a databázou MySQL sme využili JDBC (Java Database Connectivity). Spojenie s databázou sa vytváralo v metóde *init()*, ktorá sa volá pri inicializácii servletu. Pripojenie využíva MySQL konektor a štandardné rozhranie DriverManager (**Chyba! Nenašiel sa žiaden zdroj odkazov.**).

```

public void init() throws ServletException {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        con =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/hotel_
events", "root", "");
    } catch (Exception e) {
        error = "Nepodarilo sa pripojiť k databáze: " +
e.getMessage();
    }
}

```

Obrázok 6 Pripojenie k databáze

Pre štýlový a responzívny dizajn webového rozhrania sme používali knižnicu Bootstrap 5, doplnenú o vlastné CSS úpravy. Pri spracovaní hesiel sme využili knižnicu *org.mindrot.jbcrypt.BCrypt*, ktorá nám umožnila bezpečne ukladať heslá v zahashovanej podobe a tým zvyšovať celkovú úroveň bezpečnosti systému. Samotné hashovanie hesiel sme implementovali v *registraciaServlet.java*.

```

import org.mindrot.jbcrypt.BCrypt;

String heslo = request.getParameter("heslo");
String hashHeslo = BCrypt.hashpw(heslo, BCrypt.gensalt());
String sql = "INSERT INTO user (first_name, last_name, born,
phone, email, password, gender, role, created_at, updated_at)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement ps = con.prepareStatement(sql);

ps.setString(6, hashHeslo);

```

Obrázok 7 Hashovanie hesla

Pri registrácii používateľa sa heslo hashovalo ešte pred uložením do databázy a pri prihlásení sa overovalo porovnaním hashu s hodnotou v databáze (Obrázok 7).

### 3.2.2 Implementácia prihlasovania v JSP

Proces prihlasovania sme implementovali pomocou servletu, ktorý spracúva údaje z prihlasovacieho formulára. Hlavným cieľom tejto časti je overiť identitu používateľa a vytvoriť vhodné prepojenie s relevantnými údajmi pre neskoršie použitie v aplikácii. Prihlasovací formulár odosiela údaje cez metódu *POST* na servlet, ktorý získa prihlasovacie údaje z formulára (Obrázok 8).

```
String email = request.getParameter("email");
String heslo = request.getParameter("password");
```

Obrázok 8 Získavanie parametrov

Tieto hodnoty predstavujú vstupy z formulára a sú základom pre overenie identity. Ako ďalší krok vyhľadá používateľa v databáze. Databáza sa prehľadáva podľa unikátneho e-mailu, ktorý slúži ako prihlasovacie meno a overí sa správnosť hesla pomocou *BCrypt*.

```
PreparedStatement ps = con.prepareStatement("SELECT * FROM user
WHERE email = ?");
ps.setString(1, email);
ResultSet rs = ps.executeQuery();
if (rs.next()) {
    String hashedPwd = rs.getString("password");
    if (BCrypt.checkpw(pwd, hashedPwd)) {
        HttpSession session = request.getSession();
        session.setAttribute("ID", rs.getInt("id"));
        session.setAttribute("login", rs.getString("email"));
        session.setAttribute("first_name",
rs.getString("first_name"));
        session.setAttribute("last_name",
rs.getString("last_name"));
        session.setAttribute("born", rs.getDate("born"));
        session.setAttribute("phone", rs.getString("phone"));
        session.setAttribute("email", rs.getString("email"));
        session.setAttribute("role", rs.getString("role"));
        response.sendRedirect("mainServlet");
        return;
    }
}
```

Obrázok 9 vyhľadávanie v databáze a prihlásenie

Na obrázku (Obrázok 9) pomocou *rs.next()* overujeme, či SQL dotaz našiel aspoň jeden záznam, ktorý reprezentuje používateľa s daným e-mailom. Ak sa žiaden záznam nenájde, zobrazí sa nám chybové hlásenie: „Používateľ s týmto emailom neexistuje.“. Následne sa z databázy získa hashované heslo (*hashedPwd*). Funkcia *BCrypt.checkpw(...)* porovná heslo zadané vo formulári (*pwd*) s hashom z databázy. Ak sú heslá zhodné, pokračuje sa do tela podmienky, inak sa zobrazí hlásenie: „Nesprávne heslo.“. Telo podmienky slúži na vytvorenie používateľskej relácie (*session*). Tento *session* objekt si uloží kľúčové údaje o používateľovi. Tieto údaje sa potom následne používajú počas celej *session* relácie. Po zapísaní údajov sa používateľ presmeruje na hlavnú stránku aplikácie, pričom *return* ukončí vykonávanie servletu. Ak sa používateľ nenašiel alebo zle zadal

heslo, zobrazí sa formulár s odpovedajúcou chybovou hláškou a vracia ich na pôvodný formulár vďaka `zobrazLoginFormular(...)`.

### 3.2.3 Implementácia prístupu k funkcionalitám aplikácie

Pre implementáciu prístupu používateľa k jednotlivým funkcionalitám aplikácie sme si vytvorili pomocnú triedu *RoleHelper*.

```
public class RoleHelper {

    public static boolean hasAccess(HttpSession session,
String... allowedRoles) {
        if (session == null || session.getAttribute("role") ==
null) {
            return false;
        }

        String userRole = (String)
session.getAttribute("role");

        for (String role : allowedRoles) {
            if (role.equalsIgnoreCase(userRole)) {
                return true;
            }
        }
        return false;
    }

    public static String getUserRole(HttpSession session) {
        if (session == null) return null;
        return (String) session.getAttribute("role");
    }
}
```

Obrázok 10 Trieda *RoleHelper*

Táto trieda obsahuje metódu *hasAccess(...)*, ktorá overuje, či má používateľ, identifikovaný v rámci *session*, požadovanú rolu pre vykonanie konkrétnej akcie alebo zobrazenie časti stránky. Metóda umožňuje zadanie viacerých rolí a vracia hodnotu *true* vtedy, ak má prihlásený používateľ uloženú túto rolu v *session*. Metóda *createHeader(...)* generuje horný navigačný panel stránky, ktorý sa prispôsobuje podľa toho, či je používateľ prihlásený a akú má rolu. Dynamicky zobrazuje položky menu ako napríklad:

- Moje eventy a Pridať event – pre staff a admin
- Zmeniť role – exkluzívne pre admin
- Upraviť profil, Moje rezervácie a Odhlásiť sa – pre všetkých prihlásených



Ak používateľ nie je prihlásený, zobrazí sa mu tlačidlo Prihlásiť sa. Príklad časti tejto dynamickej hlavičky: Príklad použitia triedy *RoleHelper* v servletoch:

```
if (RoleHelper.hasAccess(ses, "admin", "staff")) {
    out.println("<a href='myEventServlet'>Moje eventy</a>");
    out.println("<a href='pridatEventServlet'>Pridať Event</a>");
}

if (RoleHelper.hasAccess(ses, "admin")) {
    out.println("<a href='zmenitRoleServlet'>Zmeniť role</a>");
}
```

Obrázok 11 použitie *RoleHelper*

V časti kódu na obrázku (Obrázok 11) sme použili triedu *RoleHelper* a jej metódu *hasAccess(...)* pre výpis jednotlivých odkazov, ktoré dokážu používateľa presmerovať na iné časti aplikácie. Podobne sme vytvorili aj metódu *createBody(...)*. Táto metóda je v jednotlivých servletoch implementovaná samostatne a obsahuje hlavnú logiku zobrazovania – napríklad výpis eventov, detail udalosti, registračný formulár a podobne. Vďaka oddeleniu od hlavičky a päty sa zvyšuje prehľadnosť kódu a znižuje sa duplikácia HTML výstupu.

Ako poslednú časť sme vytvorili metódu *createFooter(...)*. Táto metóda zabezpečuje zobrazenie spodnej časti stránky, ktorá obsahuje:

- názov a copyright projektu,
- fixné umiestnenie na spodok obrazovky,
- rovnaký vzhľad na všetkých podstránkach.

### 3.2.4 Výpis HTML obsahu na stránku

Pri vývoji aplikácie v JavaServer Pages a servletoch vypisujeme HTML priamo cez *PrintWriter* do HTTP odpovede. Servlet ako Java trieda komunikuje s webovým prehliadačom prostredníctvom objektu *HttpServletResponse*. Pomocou jeho metódy *getWriter()* sme získali *PrintWriter*, cez ktorý môžeme vypisovať textový výstup, teda aj HTML. Tento postup môžeme vidieť na obrázku (Obrázok 12) nižšie:

```

response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();

private void createFooter(PrintWriter out, HttpServletRequest
request) {
    out.println("<footer>");
    out.println("<p>© 2024 Event Management</p>");
    out.println("</footer>");
}

```

Obrázok 12 metóda *createFooter()*

V tomto príklade servlet programovo vytvára HTML dokument. Kód vo formáte *out.println(...)* zapisuje každý riadok HTML ako text. Tento prístup sme používali v rámci celej aplikácie.

### 3.2.5 Výpis eventov

Na výpis eventov v aplikácii sme vytvorili servlet *mainServlet.java*, ktorý funguje ako úvodná stránka celej aplikácie. Táto stránka umožňuje výpis všetkých eventov, ktoré sa nachádzajú v databáze *hotel\_events* v tabuľke *event*. Pre prehľadnú vizualizáciu jednotlivých eventov sme použili elementy Bootstrap kartičiek ktoré umožňujú používateľovi jednoducho a rýchlo získať hlavné informácie o evente a kliknutím prejsť na stránku s podrobnejšími informáciami.

Samotný výpis sme implementovali v metóde *createBody(...)*, ktorá sa volá v metódach *doGet()* a *doPost()*. Táto metóda najprv nadviaže spojenie s databázou a vykoná SQL dotaz, ktorého výsledkom je súbor záznamov z tabuľky *event* (Obrázok 13):

```

private void createBody(PrintWriter out, HttpServletRequest
request) {
    try {
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM event");
    }
}

```

Obrázok 13 metóda *createBody()*

Následne sa každý event vypíše ako vlastná karta (Bootstrap card) s údajmi ako názov, dátum, organizátor a informácia o dostupnosti miest vďaka získania údajov z redefinovaného *ResultSet(rs)*.

```

while (rs.next()) {
    int eventId = rs.getInt("id");
    out.println("<a href='eventDetailServlet?eventId=" + eventId
+ "'>");
    out.println(rs.getString("event_name"));
    out.println("<p>" + rs.getString("organizer_company") + "
</p>");
    out.println("<p>Dátum: " + rs.getString("event_date") + "
</p>");
    int volneMiesta = rs.getInt("available");
    if (volneMiesta > 0) {
        out.println("<p>Voľné miesta</p>");
    } else {
        out.println("<p>Miesta sú plné</p>");
    }
}

```

Obrázok 14 Výpis kartičiek

Prístup na obrázku (Obrázok 14) vyššie zabezpečuje, že každý event sa zobrazí ako samostatná kartička so základnými údajmi a odkazom na podrobný detail eventu (*eventDetailServlet.java*). V prípade že event nemá dostupné miesta (*available = 0*), upozorní to používateľa červeným textom.

### 3.2.6 Implementácia rezervácie eventov

V rámci servletu *eventDetailServlet.java* sme kládli dôraz na implementáciu funkcií, ktoré umožňujú prihláseným používateľom interagovať s konkrétnym eventom po kliknutí na jeho kartu. Servlet sa skladá z viacerých častí, pričom sme medzi najdôležitejšie funkcionality z pohľadu funkčnosti aplikácie považovali možnosť rezervácie eventu a správu eventu.

Prvou významnou časťou servletu je rezervácia. Po načítaní daného eventu sa na základe session identifikuje, či je daný používateľ prihlásený. Dôležitým faktorom pre zobrazovanie tlačidiel je to, či bol daný event týmto používateľom v minulosti rezervovaný. Ak nie, zobrazí sa mu tlačidlo „Rezervovať“, ktoré po odoslaní *POST* požiadavky na servlet *rezervujEventServlet.java* zabezpečí vytvorenie rezervácie. Ak si používateľ event v minulosti rezervoval, zobrazí sa mu tlačidlo „Zrušiť rezerváciu“. V tomto prípade sa odošle požiadavka na servlet *zrusRezervacijuServlet.java*.

```

if (!maRezervaciu) {
    if (dostupnost == 1) {
        out.println("<form method='post'
action='rezervujEventServlet'>");
        out.println("<input type='hidden' name='event_id'
value='" + eventId + "'>");
        out.println("<button type='submit'>Rezervovať</button>");
        out.println("</form>");
    } else {
        out.println("<p>Event je plne obsadený.</p>");
    }
} else {
    out.println("<form method='post'
action='zrusRezervaciuServlet'>");
    out.println("<input type='hidden' name='event_id' value='" +
eventId + "'>");
    out.println("<button type='submit'>Zrušiť
rezerváciu</button>");
}
}

```

Obrázok 15 Overenie rezervácie eventu

Na obrázku (Obrázok 15) je viditeľné, že po kliknutí na tlačidlo rezervovať sa na *rezervujEventServlet.java* odošle v skrytom poli kľúč *event\_id*, aby servlet vedel, ktorý event sa má rezervovať. Ak však používateľ už rezerváciu vytvoril, môže samotnú rezerváciu aj zrušiť. Po kliknutí na tlačidlo „Zrušiť rezerváciu“ sa v skrytom poli pošle kľúč *event\_id* do servletu *zrusRezervaciuServlet.java*. Týmto spôsobom sme zabezpečili, aby sa odstránil záznam z tabuľky *event\_reservation* pre príslušného používateľa a daným *event\_id*. Po vykonaní operácie sa znovu načíta *eventDetailServlet.java*.

Po spustení servletu *rezervujEventServlet.java* sa overí, či je používateľ vôbec prihlásený. Ak nie je, presmeruje ho to späť na prihlasovaciu stránku. Ak je prihlásený, servlet si zo session zistí jeho ID a zároveň z parametrov získa *event\_id*, na ktorý chce používateľ spraviť rezerváciu.

Ďalej sa z databázy vyberie dátum daného eventu, aby sa neskôr mohol uložiť k rezervácii. Servlet potom skontroluje, či si daný event v minulosti nerezervoval. Ak áno, preskočí sa ukladanie a používateľa presmeruje späť na *eventDetailServlet.java*. Ak rezervácia ešte nie je zaznamenaná, vytvorí sa záznam v databáze. V rámci tejto operácie sa taktiež uloží čas začiatku a konca eventu. Po úspešnom zapísaní do databázy sa používateľ automaticky presmeruje na *eventDetailServlet.java*, kde sa mu zobrazí hláška, že rezervácia prebehla úspešne.

V servlete *zrusRezervaciuServlet.java* sme zabezpečili, aby si prihlásený používateľ mohol zrušiť rezerváciu na daný event. Najskôr servlet skontroluje, či je

používateľ prihlásený. Ak je prihlásený, servlet si zo session zoberie jeho používateľské ID a z požiadavky (requestu) vezme *event\_id*, ku ktorému sa má rezervácia zrušiť. Následne sa z databázy odstráni záznam z tabuľky *event\_reservation*, kde sa zhoduje *user\_id* a *event\_id*. Týmto spôsobom sme zabezpečili vymazanie prepojenia medzi daným používateľom a eventom, ktorý si predtým rezervoval. Po úspešnom odstránení rezervácie servlet presmeruje používateľa na stránku s detailom eventu. Zároveň sme v URL adrese pridali informáciu *zrusenie=ok*, vďaka čomu sa tam potom zobrazí hláška, že rezervácia bola úspešne zrušená.

Druhou kľúčovou funkcionalitou je správa samotného eventu. Prístup k tejto akcii, má len používateľ s rolou *admin* alebo samotný tvorca eventu. V prípade že daný používateľ spĺňa tieto požiadavky, zobrazia sa mu tlačidlá pre úpravu eventu (*upravEventServlet.java*) alebo vymazanie eventu z databázy (*vymazEventServlet.java*).

```
if (RoleHelper.hasAccess(ses, "admin") || currentUserId == creatorId) {  
    // Vymazať udalosť  
    out.println("<form method='post' action='vymazEventServlet'  
onsubmit='return confirm(\"Naozaj chcete vymazať tento event?  
\")';>");  
    out.println("<button type='submit'>Vymazať</button>");  
    // Upraviť udalosť  
    out.println("<form method='get'  
action='upravEventServlet'>");  
    out.println("<button type='submit'>Upraviť</button>");  
}
```

Obrázok 16 Tlačidlá pre admina a tvorca

### 3.2.7 Operácie nad Eventami

Ďalším dôležitým prvkom našej aplikácie je možnosť vymazať existujúci event. Táto funkcionalita je dostupná len pre administrátora alebo pre používateľa, ktorý daný event vytvoril. O túto operáciu sa stará servlet *vymazEventServlet.java*, ktorý je implementovaný ako *POST* požiadavka a zabezpečuje bezpečné odstránenie údajov z databázy. Po stlačení tlačidla „Vymazať“ sa servlet spustí a najprv skontroluje, či je používateľ prihlásený. Následne si servlet načíta ID prihláseného používateľa a jeho rolu zo session. Zároveň z požiadavky získa ID eventu, ktorý chce používateľ odstrániť.

Prvým krokom je zistenie, kto vytvoril daný event, teda aké *user\_id* má uložené v databáze pre tento konkrétny event. Ak sa zistí, že používateľ je buď administrátor alebo ten, kto event vytvoril, pokračuje sa v mazaní. Servlet najprv odstráni všetky rezervácie, ktoré sú na daný event vytvorené v tabuľke *event\_reservation*. Až potom sa

vykoná odstránenie samotného eventu z tabuľky event. Týmto spôsobom sme zabezpečili, že nevzniknú v databáze neplatné záznamy. Po úspešnom vymazaní servlet presmeruje používateľa späť na hlavnú stránku s parametrom *deleted=ok*, čo slúži na zobrazenie potvrdenia o úspešnom vymazaní daného eventu. Naopak, ak používateľ nemá právo daný event vymazať, alebo ak event neexistuje, servlet zabezpečí presmerovanie a informovanie používateľa.

Ďalšou nevyhnutnou operáciou v našej aplikácii bolo upravovanie údajov o existujúcich eventoch zaznamenaných v databáze. Túto funkcionality môžu využívať len samotní tvorcovia eventu alebo používatelia s rolou *admin*. Pre implementáciu tejto funkcionality sme vytvorili servlet *upravEventServlet.java*. Tento servlet pracuje s dvoma metódami – *doGet()* a *doPost()*. Metóda *doGet()* je zodpovedná za zobrazenie formulára, ktorý je rovnaký ako formulár pre pridávanie eventov. V tomto formulári sa používateľovi predvyplnia údaje o evente, ktoré má zapísané v tabuľke *event*. Pri načítaní stránky si servlet prečíta údaje z tabuľky podľa *event\_id*, ktoré boli odoslané cez URL parametre, a vypíše ich do HTML formulára. Používateľ tak môže jednoducho zmeniť údaje ako:

- názov,
- dátum,
- spoločnosť, ktorá event organizuje,
- hotel,
- popis,
- dostupnosť miest.

Na druhej strane sme implementovali metódu *doPost()*, ktorá sa spúšťa po odoslaní formulára. V tomto kroku servlet načíta všetky údaje z formulára, skontroluje, či je používateľ autorizovaný vykonať túto úpravu, a nakoniec pomocou *PreparedStatement* aktualizuje existujúci záznam v databáze. Po úspešnom upravení záznamu sme automaticky nastavili hodnotu *updated\_at* na aktuálny čas, aby sme mali prehľad o tom, kedy naposledy prišlo k zmene záznamov v tabuľke. Po uložení zmien je používateľ presmerovaný späť na stránku s detailom eventu, pričom sa do URL doplní parameter *updated=ok*, ktorý umožní zobrazit' hlásenie o úspešnej úprave.

### 3.2.8 Hotely a hodnotenia

Súčasťou aplikácie je aj stránka pre správu hotelov, ktoré slúžia ako miesta konania jednotlivých eventov. Každý event je viazaný na konkrétny hotel a používateľ



má možnosť po jeho absolvovaní hotel ohodnotiť. Hodnotenie pozostáva z textovej poznámky a číselného skóre od 0 do 5. Tieto hodnotenia sme zaznamenávali v samostatnej tabuľke a pri zobrazení detailu hotela sa vypisujú spolu s menom autora a dátumom pridania. Celý tento systém prispieva k prehľadnosti a zlepšuje používateľskú skúsenosť pri výbere eventov a ich lokalít.

Eventy


Hľadať eventy

Zobraziť hotely

### Hotel Iota

Adresa: 606 Meadow Dr

Hodnotenie: 4 ★



#### Hodnotenia hotela

Michael Taylor – 4/5 ★

Nice place.

Hodnotené: 2024-09-30

© 2024 Event Management

Obrázok 17 Detaily a hodnotenie hotela

### 3.3 VÝVOJ APLIKÁCIE V JAVA SPRING BOOT

V tejto podkapitole popisujeme jednotlivé kroky vývoja aplikácie pomocou technológie Java Spring Boot.

#### 3.3.1 Vývojové prostredie

Pri vývoji aplikácie pomocou Java Spring Boot sme zvolili IntelliJ IDEA, ktoré poskytuje rozsiahlu podporu pre prácu so Spring Boot projektmi vrátane automatického dopĺňania, správy závislostí a vstavaného spúšťania aplikácie cez vstavaný Tomcat server. Závislosti a buildovanie projektu sme riadili pomocou nástroja *Maven*, pričom všetky knižnice sú definované v súbore pom.xml. V projekte sme využili najmä závislosti

*spring-boot-starter-web*, *spring-boot-starter-thymeleaf*, *spring-boot-starter-data-jpa*, *mysql-connector-java* a *spring-boot-starter-security*. Aplikácie sme nasadzovali lokálne na serveri Tomcat, ktorý je predinštalovaný v Spring Boot a nevyžaduje samostatnú konfiguráciu ako pri tradičných servletoch.

Dáta sme opäť uchovávali v databáze *MySQL*, pričom konfiguráciu pripojenia sme zabezpečili v súbore *application.properties*. Tam sme definovali prihlasovacie údaje k databáze, URL pripojenia, a taktiež parametre týkajúce sa JPA.

```
spring.application.name=Projekt
spring.datasource.url=jdbc:mysql://localhost:3306/hotel_events
spring.datasource.username=root
spring.datasource.password=
, []
}
```

Obrázok 18 *Application.properties*

Štruktúra projektu je rozdelená do balíkov, ktoré reprezentujú hlavné vrstvy aplikácie:

- config – zahŕňa nastavenie prístupu a autentifikácie,
- controller – spracúva a obsluhuje HTTP požiadavky,
- repository (dao) – slúži na komunikáciu s databázou,
- entity – definuje triedy tabuliek v databáze,
- service – obsahuje aplikačnú logiku.

### 3.3.2 Implementácia prihlasovania v Java Spring Boot

Prihlasovanie používateľa sme v aplikácii realizovali cez kombináciu vlastného kontroléra, service a repository vrstvy, pričom dôraz sme kládli na bezpečnosť spracovania hesla a používateľských údajov. Na backendovej strane sme vytvorili kontrolér *UserController*, ktorý spracováva požiadavky z prihlasovacieho formulára. Po zadaní prihlasovacích údajov sa POST požiadavka odošle na endpoint */login*, ktorý je spracovaný metódou *loginUser(...)*.



```

@PostMapping("/login")
public String loginUser(@RequestParam String login, @RequestParam
String pwd, HttpSession session, Model model) {
    User user = userService.findByEmail(login);
    if (user != null && BCrypt.checkpw(pwd, user.getPassword()))
    {
        session.setAttribute("user", user);
        return "redirect:/";
    }
    model.addAttribute("error", "Nesprávne prihlasovacie
údaje.");
    return "login";
}
}

```

Obrázok 19 Metóda *loginUser()*

Metóda *findByEmail* je implementovaná v *UserService*, kde sa používateľ načítava pomocou *repository* triedy, ktorá využíva JPA. Ak je heslo správne, používateľa uložíme do session a presmerujeme ho na hlavnú stránku. V opačnom prípade sa zobrazí chybová hláška. Overenie údajov prebieha výhradne na strane servera. Týmto spôsobom sme zabezpečili vyššiu bezpečnosť aplikácie (Obrázok 19).

### 3.3.3 Implementácia prístupu k funkcionalitám aplikácie

Zabezpečený prístup k funkcionalitám aplikácie v Spring Boot sme riešili pomocou *Spring Security*, kde sme definovali pravidlá pre prihlásenie, autorizáciu a odhlásenie používateľa. Toto nastavenie zabezpečenia sme vykonali v triede *SecurityConfig.java*, kde sme pomocou metódy *securityFilterChain* zadefinovali, ktoré URL adresy sú verejné a ktoré sú prístupné len autorizovaným používateľom.

```

.authorizeHttpRequests(authorizeRequests -> authorizeRequests
    .requestMatchers("/", "/login", "/register").permitAll()
    .requestMatchers("/admin/**").hasRole("admin")
    .requestMatchers("/staff/**").hasRole("staff")
    .requestMatchers("/user/edit-profile").authenticated()
)

```

Obrázok 20 Metóda *securityFilterChain*

Taktiež sme definovali vlastnú prihlasovaciu stránku */login*, ako aj správanie po úspešnom prihlásení a odhlásení. Heslá sú zabezpečené hashovacou funkciou *BCrypt*, ktorú sme nastavili cez komponent *PasswordEncoder*. Týmto nastavením sme zabezpečili, že sa používateľ nedostane k časti aplikácie, ku ktorej nemá prístup, čím sme zabezpečili ochranu dát a správne prístupy podľa rolí.

### 3.3.4 Thymeleaf v Spring Boot

V rámci vývoja aplikácie v Spring Boot sme na zobrazovanie používateľského rozhrania využívali šablónovací systém *Thymeleaf*, ktorý umožňuje dynamicky generovať HTML obsah na základe dát z kontroléra. Dôležitým prvkom pre vytváranie a vypisovanie obsahu boli práve fragmenty, ktoré nám slúžili ako znovu použiteľný prvok UI. Tieto jednotlivé fragmenty *footer.html* a *header.html* sme vkladali do hlavných HTML súborov pomocou *th:replace*. Tento prístup nám zabezpečil jednotný vzhľad stránky naprieč celou aplikáciou bez potreby duplikácie HTML kódu. Takýto spôsob generovania HTML nám výrazne pomohol k dosiahnutí efektivity v porovnaní so servletovým riešením. Týmto spôsobom sme eliminovali ručné vypisovanie HTML cez *PrintWriter*.

### 3.3.5 Výpis eventov pomocou Java Spring Boot

Zobrazenie eventov na hlavnej stránke sme implementovali pomocou kontroléra *EventController*, kde sme v rámci metódy s anotáciou *@GetMapping("/")* načítali všetky eventy z databázy pomocou služby *EventService*. Táto služba následne využíva *EventRepository*, ktorá dedí z rozhrania *JpaRepository* a zabezpečuje jednoduché získanie všetkých záznamov z tabuľky event. Takto získané údaje sú následne prenesené do *index.html*, kde sú pomocou *Thymeleaf* cyklu *th:each* vypisované do kariet.

```
@GetMapping("/")
public String showHomePage(Model model) {
    List<Event> events = eventService.getAllEvents();
    model.addAttribute("events", events);
    return "index";
}

//index.html
<div th:each="event : ${events}">
    <h5 th:text="${event.eventName}"></h5>
    <p th:text="${event.organizerCompany}"></p>
    <a th:href="@{/event/detail/{id}(id=${event.id})}">Zobraziť
    detail</a>
</div>
```

Obrázok 21 *@GetMapping()* a *index.html*

### 3.3.6 Zobrazenie detailov eventu

Po kliknutí používateľa na konkrétnu kartu eventu, presmeruje ho na podstránku s detailnými informáciami o evente. Prístup k týmto detailom sme zabezpečili v kontroléri *EventController*, ktorý načíta detail eventu na základe jeho ID.

```

@GetMapping("/event/detail/{id}")
public String showEventDetail(@PathVariable Long id, Model model,
    HttpSession session) {
    Event event = eventService.getEventById(id);
    model.addAttribute("event", event);
    return "event-detail";
}

```

Obrázok 22 Metóda *showEventDetail(...)*

Zobrazenie jednotlivých prvkov na stránke, ako sú napríklad tlačidlá na registráciu, zrušenie registrácie, úpravu alebo vymazanie udalosti, sme zabezpečovali kombináciou backendovej logiky v kontroléri a dynamického vykresľovania pomocou šablónovacieho systému Thymeleaf. V kontroléri sme si pripravili pomocné premenné, ktoré určovali, čo sa má na stránke používateľovi zobraziť. Medzi tieto premenné patrila napríklad informácia o tom, či má aktuálny používateľ rezerváciu na daný event, či je jeho autorom alebo či má rolu administrátora. Tieto údaje sme následne pomocou objektu *Model* odoslali do šablóny. V samotnej šablóne sme následne využili výraz *th:if*, ktorý slúžil na podmienené zobrazenie konkrétnych častí HTML kódu. Vďaka tomu sa daný prvok zobrazil len vtedy, ak boli splnené všetky požadované podmienky. Takto sme zabezpečili aby bol kód prehľadnejší a zároveň sme zabezpečili ochranu pred neautorizovanými akciami používateľa.

### 3.3.7 Implementácia rezervácie eventov v Spring Boot

Pri implementácii rezervácie eventov vo *frameworku* Spring Boot sme sa snažili vyhnúť manuálnej validácii v každom kontroléri ako pri servletoch v JSP. Preto sme využili funkcie na overovanie identity a prístupových práv pomocou *Spring Security*. Rezervácia z pohľadu používateľa funguje podobne. Po kliknutí na tlačidlo „Rezervovať“ sa spustí *POST* požiadavka na endpoint */event/reserve/{eventId}*, ktorý spracováva metóda v *EventReservationController*. Identita používateľa je zabezpečená v aktuálnej session, pričom informácie o používateľovi získame priamo z funkcie *Principal*. Kontrola, či si už používateľ daný event rezervoval, prebieha v *eventReservationRepository*, ktorý volá SQL dotaz do databázy. Samotné zapísanie a uloženie danej rezervácie do databázy sme zabezpečili v metóde *save()* nad entitou *EventReservation*.

Pri zabezpečení zrušenia rezervácie sme implementovali odoslanie požiadavky na endpoint */event/cancel-reservation/{eventId}*, kde opäť kontrolujeme, či je daný záznam v databáze priradený aktuálnemu používateľovi. Ak áno, záznam sa z databázy odstráni

a používateľ je presmerovaný späť na stránku s detailom eventu. Tu sa mu vďaka odoslanému parametru *zrusenie=ok* zobrazí hláška o úspešnom zrušení rezervácie.

```
@PostMapping("/event/reserve/{id}")
public String reserveEvent(@PathVariable("id") Long eventId,
    Principal principal, RedirectAttributes redirectAttributes) {
    User user = userRepository.findByEmail(principal.getName());
    Event event =
    eventRepository.findById(eventId).orElseThrow();

    if (eventReservationRepository.existsByUserAndEvent(user,
    event)) {
        return "redirect:/event/detail/" + eventId;
    }
    EventReservation reservation = new EventReservation();
    reservation.setUser(user);
    reservation.setEvent(event);
    eventReservationRepository.save(reservation);
    redirectAttributes.addAttribute("rezervacia", "ok");
}
```

Obrázok 23 Rezervácia eventu

### 3.3.8 Spravovanie eventov pomocou Spring Boot

V rámci vývoja aplikácie sme zabezpečili správu eventov pre používateľov, ktorí sú tvorcami eventu, alebo majú pridelenú rolu admin. Táto funkcionality zahŕňa dve hlavné operácie – úpravu údajov udalosti a jej úplné odstránenie z databázy. Rovnako ako pri implementácii rezervácie eventov sme aj pri tejto časti aplikácie využili funkcionality *Spring Security*. Na stránke s detailom o danom evente sú tlačidlá „Upraviť“ a „Vymazať“ dostupné len v prípade, že aktuálny používateľ spĺňa oprávnenie. Toto je zabezpečené pomocou podmienok priamo v šablóne HTML využívajúcej Thymeleaf.

```
div th:if="${#authentication.name == event.creator.email or
    #authorization.expression('hasRole(''ROLE_ADMIN'')')}">
```

Obrázok 24 podmienka pre autentifikáciu

Pri kliknutí na tlačidlo pre upravenie eventu je používateľ presmerovaný na stránku s formulárom, kde sú predvyplnené všetky údaje o udalosti. Tieto dáta sú získané cez kontrolér *EventController*, ktorý načíta existujúcu udalosť podľa ID a odošle ju do HTML súboru. Po upravení údajov používateľ odošle formulár na endpoint */event/update*, kde sa zmena uloží do databázy vďaka *eventService*. O správnu aktualizáciu údajov sa stará servisná vrstva, ktorá zároveň kontroluje, či má používateľ oprávnenie upravovať daný záznam.

Odstránenie eventov prebieha podobne. Po stlačení tlačidla pre vymazanie eventu sa odošle *POST* požiadavka na endpoint */event/delete/{id}*. V tomto prípade metóda kontroléra odstráni daný záznam z databázy, pričom predtým zmaže aj všetky rezervácie, ktoré sú k nemu pripojené.

### 3.3.9 Implementácia Hotelov a hodnotení v Spring Boot

V tejto časti aplikácie sme informácie o hoteloch zobrazovali prostredníctvom kontroléra, ktorým sme odovzdávali zoznam hotelov do šablóny ako objekty entity *Hotel*. Samotný výpis sme implementovali v HTML šablóne pomocou funkcie *th:each*, ktorá prechádza(iteruje) zoznam hotelov a dynamicky generuje HTML kód s údajmi ako názov, adresa a priemerné hodnotenie.

Pri zobrazení detailu hotela sme k objektu hotela načítavali aj všetky príslušné hodnotenia z tabuľky *hotel\_reviews*. Tieto hodnotenia boli automaticky mapované pomocou vzťahu *@OneToMany*, čo nám umožnilo jednoduchý prístup ku všetkým hodnoteniam patriacim danému hotelu. Zobrazovanie hodnotení sme riešili priamo v samotnej šablóne, pričom každý záznam obsahoval meno používateľa, dátum, hodnotenie (0-5) a text poznámky.

Nato aby používateľ mohol pridávať hodnotenia konkrétnemu hotelu sme využili formulár, ktorý odosiela dáta na endpoint v kontroléri a tie sme následne uložili pomocou *ReviewRepository*. Taktiež sme využili JPA repozitáre, ktoré nám umožnili filtrovať recenzie podľa cudzieho kľúča *hotel\_id* bez toho, aby sme ručne písali SQL dotazy.

## 4 VÝSLEDKY

Po úspešnej implementácii oboch verzií webovej aplikácie sme sa v nasledujúcej časti venovali samotnému testovaniu ich funkčnosti. Zamerali sme sa na porovnanie praktických rozdielov medzi použitými technológiami, ktoré sa prejavili počas implementácie a testovania jednotlivých riešení.

### 4.1 PRIEBEH TESTOVANIA APLIKÁCIÍ

Po dokončení implementácie oboch verzií aplikácie sme sa zamerali na ich dôkladné testovanie. Testovanie sme vykonávali manuálne, pričom sme sa snažili nasimulovať správanie reálnych používateľov rôznych rolí. Každá z rolí mala špecifické oprávnenia, a preto sme sa zamerali na overenie prístupových práv používateľa, dostupnosti jednotlivých funkcií a správneho zobrazovania obsahu v aplikácii.

Testovanie prebiehalo v lokálnom prostredí s databázou MySQL, pričom sme si vopred naplnili testovacie dáta do tabuliek. V JSP verzii sme sa zamerali najmä na fungovanie jednotlivých servletov a správnu manipuláciu so session. V prípade Spring Boot aplikácie sme testovali funkčnosť jednotlivých controllerov a ich komunikáciu s databázou prostredníctvom servisnej a repository vrstvy. Sledovali sme, ako sa aplikácie správajú pri typických akciách, ako je prihlásenie, registrácia udalosti, vytváranie nového eventu, hodnotenie hotelov, zmena používateľského profilu alebo úprava rolí používateľov. Testovali sme taktiež reakcie aplikácie na pokus o prístup na stránku bez prihlásenia alebo s nedostatočnými oprávneniami.

Tento spôsob testovania nám umožnil overiť funkčnosť jednotlivých častí a identifikovať rozdiely v prístupe k programovaniu medzi oboma technológiami. Toto testovanie nám následne umožnilo objektívne porovnanie z hľadiska kvality kódu, bezpečnosti a výkonu.

### 4.2 POROVNANIE TECHNOLÓGIÍ

V tejto kapitole sme sa zaoberali samotnými rozdielmi pri implementácii jednotlivých aplikácií. Testovali sme aplikácie na základe kvality kódu, výkonnosti, bezpečnosti a podpory komunity.

#### 4.2.1 Kvalita kódu

Počas vývoja oboch verzií aplikácie sme sledovali najmä prehľadnosť, udržateľnosť a čitateľnosť zdrojového kódu. Už v úvode sme sa stretli s tým, že

využívanie technológie JSP a servletov viedlo k výrazne nižšej prehľadnosti kódu. Keďže výpis HTML prebiehal priamo v jave pomocou *PrintWriter*, veľké množstvo HTML kódu sa miešal s aplikačnou logikou, čo nielen sťažovalo orientáciu v kóde, ale komplikovalo aj jeho neskoršie upravovanie a testovanie. Keďže sa HTML kód generoval dynamicky cez reťazce, často dochádzalo k vizuálnemu zahlteniu servletov, najmä pri zložitejších výpisoch eventov alebo detailov eventu.

V prípade Spring Boot sa nám tento problém podarilo eliminovať vďaka použitiu *Thymleaf*. Vďaka oddeleniu prezentačnej vrstvy od logiky aplikácie sme dokázali zachovať lepšiu štruktúru projektu a súčasne zjednodušiť vývoj používateľského rozhrania. HTML šablóny sme mali samostatne uložené v priečinku *templates*, čo nám umožnilo ich úpravu bez potreby zasiahnuť do aplikačnej logiky. V kontroléroch sme sa preto sústredili výhradne na spracovanie požiadaviek a manipuláciu s dátami, čo výrazne zlepšilo čitateľnosť a ucelenosť celého projektu. Zároveň sme v Spring Boot využívali rozloženie projektu do jednotlivých častí ako controller, service, repository či config, čo prispelo k lepšej organizácii samotného vývoju webovej aplikácie. Tento prístup nám umožnil jednoduchšie testovanie jednotlivých komponentov a taktiež rýchlejšiu orientáciu v projekte počas vývoja a jeho úpravách.

Celkovo môžeme konštatovať, že kvalita kódu bola výrazne vyššia v prípade Spring Boot aplikácie, hlavne vďaka čistej architektúre a oddeleniu jednotlivých vrstiev aplikácie.

#### 4.2.2 Testovanie výkonu

Pre otestovanie výkonnosti oboch aplikácií sme rovnako ako autori (Yu et al., 2004) využili open-source testovací nástroj *Apache JMeter*<sup>3</sup>. Tento nástroj nám umožnil simulovať záťaž a otestovať odozvu aplikácii pri rôznych typoch požiadaviek. Pomocou tohto nástroja sme vykonali testy pre operáciu na pridanie eventu. Testy sme vykonali samostatne pre každú technológiu oddelene. Pre každý test sme vygenerovali 100 požiadaviek a sledovali sme základné metriky ako:

- čas spracovania požiadavky (average, min, max),
- priepustnosť (throughput),
- štandardnú odchýlku,
- objem prenosových dát.

---

<sup>3</sup> Zdroj technológie: [https://jmeter.apache.org/download\\_jmeter.cgi](https://jmeter.apache.org/download_jmeter.cgi)



*Tabuľka 2 Testovanie aplikácie v JSP*

Samples	Average	Min	Max	Std.Dev.	Throughput	Recieved KB/s	Sent KB/s
100	23 ms	1 ms	48 ms	22,37	97,2/sec	739,11	45,08

V prípade aplikácie v JSP sme zaznamenali priemerný čas spracovania jednej požiadavky 23 milisekúnd. Najrýchlejšia požiadavka trvala 1 milisekundu a zároveň najpomalšia 48 milisekúnd. Štandardná odchýlka bola 22,37, čo poukazuje na pomerne stabilný výkon. Priepustnosť systému bola 97,2 požiadaviek za sekundu, pričom systém zvládol bezchybne všetky požiadavky s nulovým percentom chybovosti. Objem prenesených dát dosahoval 739,11 KB/s pri prijímaní a 45,08 KB/s pri odosielaní.

*Tabuľka 3 Testovanie aplikácie v Spring Boot*

Samples	Average	Min	Max	Std.Dev.	Throughput	Recieved KB/s	Sent KB/s
100	16 ms	2 ms	218 ms	40	97,2/sec	511,2	49,73

Pri testovaní Spring Boot aplikácie sme zaznamenali priemerný čas spracovania požiadavky 16 milisekúnd, čo je o 7 milisekúnd menej ako pri JSP. Minimálna odozva bola opäť 2 milisekundy, maximálna však dosiahla až na hodnotu 218 ms, čo sa prejavilo aj na vyššej štandardnej odchýlke 40,19. Priepustnosť systému bola rovnaká ako pri JSP. Taktiež ako v predchádzajúcom prípade nedošlo k žiadnym chybám počas spracovania požiadaviek. Zaznamenali sme tiež vyšší objem prenesených dát pri odosielaní kde bola hodnota až 49,73 KB/s, a taktiež pri prijímaní, kde hodnota vystúpila až na 511,20 KB/s.

Vďaka týmto meraniam vieme zhodnotiť, že aplikácia vyvinutá pomocou Spring Boot dosahovala lepšiu priemernú rýchlosť spracovania požiadaviek. Napriek tomu sme zaznamenali väčšie výkyvy v pri štandardnej odchýlke odozvy. Naopak, JSP aplikácia síce pracovala o niečo pomalšie, no jej výkon bol stabilnejší a predvídateľnejší. Obe aplikácie zvládli spracovať rovnaké množstvo požiadaviek bez výskytu chýb, pričom si udržali podobnú priepustnosť.

#### **4.2.3 Testovanie bezpečnosti**

Pri vývoji oboch aplikácií sme sa zamerali na implementáciu základných bezpečnostných opatrení, ktoré zabezpečovali ochranu údajov a správu prístupu používateľov. V rámci testovania bezpečnosti sme hodnotili hlavne tieto aspekty:



- spôsob ukladania hesiel,
- overovanie prihlásených používateľov,
- autorizáciu prístupu k funkcionalitám,
- prácu so session.

V aplikácii vytvorenej pomocou JSP a servletov sme implementovali hashovanie hesiel pomocou knižnice BCrypt, pričom heslá sa pri registrácii ukladali do databázy už zašifrované. Pri prihlasovaní sa heslo porovnávalo s hashom v databáze. Prístup k jednotlivým funkcionalitám bol ošetrený pomocou kontrolnej triedy *RoleHelper*, ktorá na základe používateľskej role rozhodovala, či má používateľ prístup k danej časti aplikácie. Takto sme zamedzili možnosť používateľa zadávať cesty k jednotlivým servletom priamo cez URL adresu. Session sme riešili manuálne pri každom servlete, v ktorom sme overovali, či je používateľ prihlásený a akú má rolu. V prípade neautorizovaného prístupu sa používateľovi zobrazovala chybová hláška alebo bol presmerovaný na základnú stránku webovej aplikácie.

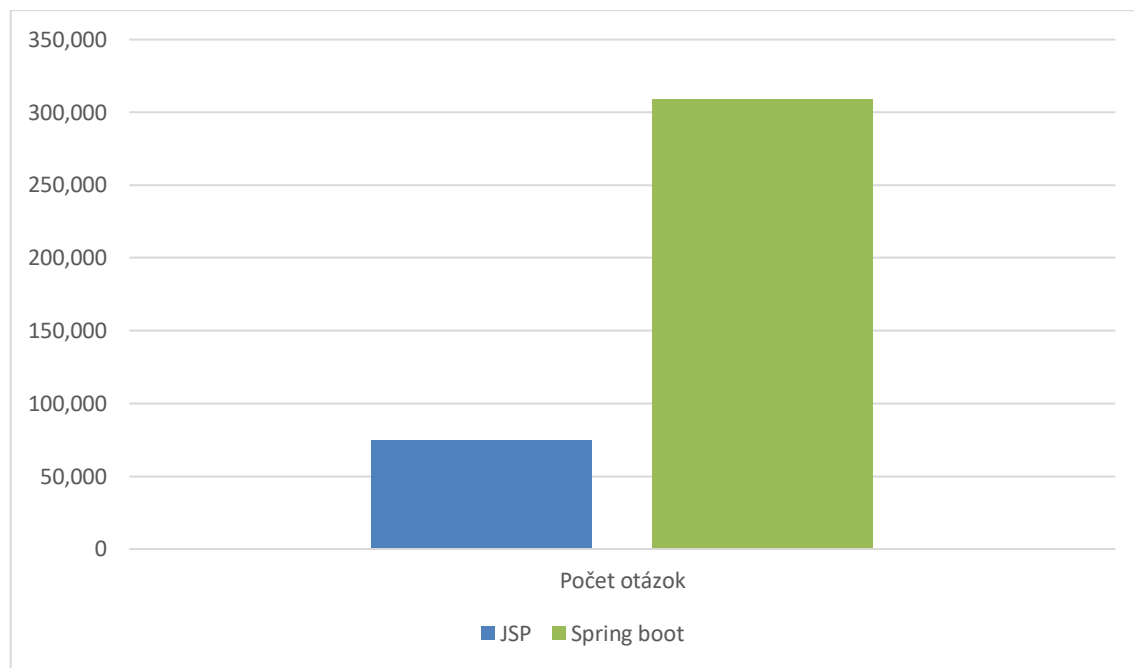
Na druhej strane, pri aplikácii v Spring Boot sme využívali vstavané mechanizmy zabezpečenia zo *Spring Security*. Autentifikácia prebiehala pomocou vlastnej implementácie *UserDetailsService*, pričom heslá boli opäť šifrované pomocou *BCryptPasswordEncoder*. Autorizácia bola riešená pomocou anotácií a bezpečnostnej konfigurácie v triede *SecurityConfig*, kde sme nastavili prístupové pravidlá pre jednotlivé URL adresy. Výhodou tohto riešenia bolo, že sme nemuseli manuálne overovať prístup pri každej požiadavke. O toto všetko sa postarala konfigurácia bezpečnostného reťazca. Celkovo sme si pri Spring Boot aplikácii všimli vyššiu úroveň bezpečnosti a to hlavne vďaka centralizovanej bezpečnostnej konfigurácii (anglicky declarative security configuration). Táto konfigurácia nám pomohla eliminovať chyby spôsobené manuálnym overovaním a taktiež jednoduchšie rozšírenie funkcionality.

#### 4.2.4 Podpora komunity

Počas vývoja oboch verzií aplikácie sme si vyskúšali, aký význam má dostupnosť dokumentácie, online príkladov a odpovedí od komunity. Zistili sme, že výber technológie výrazne ovplyvňuje aj to, ako rýchlo dokážeme vyriešiť vyskytujúce sa problémy či nájsť odpoveď na konkrétny technický problém.

Pri sledovaní podpory zo strany komunity sme zaznamenali aj počet otázok a diskusií, ktoré sú dostupné pre obe technológie na platforme *Stack Overflow*. Zistili sme, že technológia Spring Boot bola výrazne aktívnejšia, keďže počet otázok

označených tagom [spring] alebo [spring-boot] presiahol 308 800. Napriek tomu technológia JSP mala na tej istej platforme približne 74 600 otázok s tagom [jsp] alebo [servlets]. Tento rozdiel naznačuje, že Spring Boot má nielen širšiu používateľskú základňu, ale aj viac dostupných riešení pre problémy, ktoré nám môžu výrazne pomôcť počas vývoja.



*Graf 1 Počet dotazov na Stack Overflow*

Na druhej strane sa Spring Boot preukázal ako framework s veľmi silnou komunitou a rozsiahlou modernejšou dokumentáciou. Takmer každý problém, s ktorým sme sa stretli, mal už svoje riešenie na fórach, *GitHub* repozitároch, blogoch alebo v oficiálnych tutoriáloch od vývojárov *Springu*. Okrem toho sme pri vývoji využili aj širokú podporu moderných vývojových nástrojov, ako napríklad *IntelliJ IDEA*. Využitie tohto vývojového prostredia nám výrazne zjednodušil vývoj v Spring Boot frameworku.

## ZÁVER

Naším cieľom v tejto bakalárskej práci bolo vytvoriť webovú aplikáciu slúžiacu ako plánovač udalostí a rezervácií, ktorú by sme implementovali pomocou technológií JavaServer Pages (JSP) so servletmi a moderného frameworku Spring Boot. Zámerom bolo nielen realizovať funkčné riešenie s rovnakou funkcionalitou, ale predovšetkým vytvoriť priestor na objektívne porovnanie týchto technológií z viacerých hľadísk. V rámci práce sme sa zamerali na štyri hlavné aspekty, ktorými sú kvalita kódu, výkonnosť aplikácie, úroveň bezpečnosti a podpora komunity pre vývojárov.

Počas vývoja sme si overili praktické rozdiely medzi oboma prístupmi a analyzovali sme ich. Zistili sme, že JSP technológia spolu so servletmi je priamy a relatívne jednoduchý prístup vhodný najmä na menšie projekty. Vďaka tomu, že sme všetok HTML výstup generovali priamo v Jave pomocou objektu `PrintWriter`, sme mali úplnú kontrolu nad výstupom. Na druhej strane sme však čelili problémom so zníženou čitateľnosťou a komplikovanejšou údržbou kódu. Tento fakt sa prejavil najmä pri výpisoch zložitejších stránok, ako bol napríklad detail udalosti alebo profil používateľa.

Naopak, Spring Boot nám umožnil využiť modernejšie prístupy v architektúre softvéru, a to najmä vďaka jednoznačnému oddeleniu vrstiev (controller, service, repository, entity). Využitie šablónovacieho systému Thymeleaf zároveň prinieslo možnosť čistej a oddelenej tvorby používateľského rozhrania. Prístup k dátam bol zabezpečený pomocou Spring Data JPA a všetka logika súvisiaca s autentifikáciou a autorizáciou bola centralizovane spravovaná v konfiguračnej triede `SecurityConfig`. Vďaka tomu sme dosiahli nielen vyššiu mieru bezpečnosti, ale aj jednoduchosť v správe prístupových práv.

Dôležitou súčasťou práce bolo aj testovanie výkonu. To sme realizovali prostredníctvom open-source nástroja Apache JMeter, ktorý nám umožnil nasimulovať záťaž aplikácie a sledovať jej odozvu pri opakovaných požiadavkách. Výsledky testovania ukázali, že aplikácia vyvinutá v Spring Boot dosahovala o niečo lepšiu priemernú odozvu, no zároveň vykazovala vyššiu variabilitu v meraniach. Aplikácia v JSP bola mierne pomalšia, no jej výkonnostná stabilita bola vyššia. Obidve aplikácie však zvládali rovnaký počet požiadaviek bez výskytu chýb a s veľmi podobnou priepustnosťou.

V oblasti bezpečnosti sme si overili výhody deklaratívneho prístupu v Spring Boot, ktorý nám umožnil nastavovať prístupové pravidlá pomocou anotácií a konfigurácií. Naopak, v JSP sme museli každú situáciu riešiť ručne cez session a pomocnú triedu RoleHelper, čo výrazne zvyšovalo riziko vzniku chýb a zároveň znižovalo kvalitu vývoja. Aj keď sa nám podarilo zabezpečiť funkčnosť a ochranu aj v JSP aplikácii, riešenie v Spring Boot bolo efektívnejšie a menej náchylné na chyby.

Z pohľadu dostupnosti dokumentácie a technickej podpory sa ukázalo, že Spring Boot disponuje omnoho silnejšou a aktívnejšou komunitou. Počas vývoja sme zaznamenali niekoľkonásobne vyšší počet diskusií, tutoriálov a riešení dostupných online, čo významne urýchlilo a zjednodušilo riešenie vzniknutých problémov. Aj podľa údajov zo Stack Overflow, Spring Boot výrazne prevažuje nad technológiou JSP v počte otázok a odpovedí od komunity.

Na základe týchto výsledkov môžeme konštatovať, že cieľ práce bol úspešne splnený. Vytvorili sme dve plne funkčné verzie aplikácie, porovnali sme ich vývoj, výkon, bezpečnosť a podporu komunity vývojárov. Týmto sme preukázali schopnosť analyzovať rôzne prístupy k vývoju webových aplikácií, zhodnotiť ich výhody a nevýhody a navrhnúť vhodné technológie pre konkrétne potreby projektu.

Aplikácie, ktoré sme v rámci práce vytvorili, sú plne funkčné a pripravené na prípadné ďalšie rozšírenia, ako napríklad automatizované testovanie, nasadenie na server či doplnenie ďalších analytických funkcionalít. Vytvorené riešenia môžu byť ďalej rozširované a upravované, pričom poskytujú pevný základ pre prípadné nasadenie vo firemnom alebo školskom prostredí. Obzvlášť Spring Boot verzia aplikácie sa ukázala ako vhodná pre dlhodobé použitie vďaka svojej rozširiteľnosti, čistote kódu a integrácii s modernými vývojovými nástrojmi.

## ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

- ACETOZI, Jorge, 2017. The Spring Framework. V: *Pro Java Clustering and Scalability* [online]. B.m.: Apress, s. 47–53. Dostupné na: doi:10.1007/978-1-4842-2985-9\_8
- ARDIS, Mark, David BUDGEN, Gregory W. HISLOP, Jeff OFFUTT, Mark SEBERN a Willem VISSER, 2015. SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *Computer* [online]. 2015, roč. 48, č. 11, s. 106–109. ISSN 0018-9162. Dostupné na: doi:10.1109/MC.2015.345
- BEN EVANS, 2015. Java: The Legend [online]. 2015 [cit. 15. november 2024]. Dostupné na: <https://www.oreilly.com/library/view/java-the-legend/9781492048299/>
- COSMINA, Iuliana, 2022. An Introduction to Java and Its History. *Java 17 for Absolute Beginners* [online]. 2022, s. 1–31 [cit. 9. november 2024]. Dostupné na: doi:10.1007/978-1-4842-7080-6\_1
- GAJEWSKI, Michal a Wojciech ZABIEROWSKI, 2019. *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) : proceedings : Polyana, May 22-26, 2019* [online]. B.m.: [IEEE]. ISBN 9781728140292. Dostupné na: doi:10.1109/MEMSTECH.2019.8817390
- GONCALVES, Antonio, 2013. Java EE 7 at a Glance. *Beginning Java EE 7* [online]. 2013, s. 1–22 [cit. 9. november 2024]. Dostupné na: doi:10.1007/978-1-4302-4627-5\_1
- GÖSSNER, Jens, Philip MAYER a Friedrich STEIMANN, 2004. Interface utilization in the Java Development Kit. V: *Proceedings of the 2004 ACM symposium on Applied computing* [online]. New York, NY, USA: ACM, s. 1310–1315. ISBN 1581138121. Dostupné na: doi:10.1145/967900.968165
- GUTIERREZ, Felipe, 2018. Security with Spring Boot. V: *Pro Spring Boot 2* [online]. B.m.: Apress, s. 219–268. Dostupné na: doi:10.1007/978-1-4842-3676-5\_8
- GUTIERREZ, Felipe, 2019. Introduction to Spring Boot. V: *Pro Spring Boot 2* [online]. B.m.: Apress, s. 31–44. Dostupné na: doi:10.1007/978-1-4842-3676-5\_2
- JENDROCK, Eric., 2013. The Java EE 6 tutorial : advanced topics [online]. 2013, s. 526 [cit. 11. november 2025]. Dostupné na: <https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf>
- JOHNSON, Rod., 2002. *Expert one-on-one J2EE design and development*. B.m.: Wrox. ISBN 1861007841.

- JOHNSON, Rod, 2006. Spring Framework: The Origins of a Project and a Name. *Spring* [online] [cit. 11. november 2024]. Dostupné na: <https://spring.io/blog/2006/11/09/spring-framework-the-origins-of-a-project-and-a-name>
- JOSEPH B., Ottinger, Linwood JEFF a Minter DAVE, 2022. *An Introduction to Hibernate 6* [online]. Dostupné na: doi:10.1007/978-1-4842-7337-1
- JUNEAU, Josh, 2014. JavaServer Pages. V: *JavaServer Faces* [online]. Berkeley, CA: Apress, s. 55–97. Dostupné na: doi:10.1007/978-1-4842-0838-0\_2
- LI, Sihui a Liansham SUN, 2011. *2011 International Conference on Computer Science and Service System*. B.m.: IEEE. ISBN 9781424497638.
- NAKAIKE, Takuya, Goh KONDOH, Hiroaki NAKAMURA, Fumihiko KITAYAMA a Shinichi HIROSE, 2004. *JSP Splitting for Improving Execution Performance* [online]. Dostupné na: doi:10.1109/SAINT.2004.1266106
- ORACLE CORPORATION, 1977. Oracle [online]. 1977 [cit. 10. november 2024]. Dostupné na: <https://www.oracle.com/java/>
- ORACLE, Corporation, 2013. *The Java EE 6 Tutorial* [online]. [cit. 10. november 2024]. Dostupné na: <https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf>
- RAMIREZ-NORIEGA, Alan, Yobani MARTINEZ-RAMIREZ, Jesus CHAVEZ LIZARRAGA, Kevin VAZQUEZ NIEBLA a Jesus SOTO, 2020. A software tool to generate a model-view-controller architecture based on the entity-relationship model. V: *Proceedings - 2020 8th Edition of the International Conference in Software Engineering Research and Innovation, CONISOFT 2020* [online]. B.m.: Institute of Electrical and Electronics Engineers Inc., s. 57–63. ISBN 9781728184500. Dostupné na: doi:10.1109/CONISOFT50191.2020.00018
- SCARIONI, Carlo a Massimo NARDONE, 2019. *Pro Spring Security* [online]. B.m.: Apress. Dostupné na: doi:10.1007/978-1-4842-5052-5
- SUN MICROSYSTEMS, Inc., 1982. *Sun Microsystems, Inc.* [online]. 24. február 1982. [cit. 15. november 2025]. Dostupné na: <https://www.britannica.com/money/Sun-Microsystems-Inc>
- YU, J X, X LIN, H LU, Y ZHANG, Budi KURNIAWAN a Jingling XUE, 2004. *LNCS 3007 - A Comparative Study of Web Application Design Models Using the Java Technologies* [online]. Dostupné na: doi:<https://doi.org/10.1007/s11761-023-00364-w>

# ZOZNAM PRÍLOH

Prílohy sú dostupné vo verejnom GitHub repozitári:

- <https://github.com/JakubAntala/BakalarskaPraca>

Priložené prílohy obsahujú“

- Bakalársku prácu vo formáte PDF,
- návrh a export databázy,
- zdrojový kód aplikácie v JavaServer Pages,
- zdrojový kód aplikácie v Spring Boot,
- textový dokument s prihlasovacími údajmi.