Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Geffert Maroš**
Program: Informační technologie
Název: **Autorizační a autentizační řešení na platformě Java**
Authorization and Authentication Solutions on the Java Platform
Kategorie: Informační systémy
Zadání:

1. Seznamte se se současnými technologiemi pro tvorbu serverových aplikací na platformě Java. Identifikujte případy použití autorizace a autentizace v těchto aplikacích.
2. Nastudujte nejpoužívanější autorizační a autentizační knihovny dostupné na této platformě a porovnejte jejich vlastnosti s ohledem na možnosti autentizace, autorizace, správy uživatelů, odolnost proti útokům a další kritéria.
3. Po dohodě s vedoucím navrhněte ukázkovou webovou službu umožňující testování prostudovaných řešení a jejich vlastností.
4. Implementujte vytvořenou službu a proveďte nasazení na vhodný server s ohledem na testovatelnost.
5. Proveďte vyhodnocení funkčnosti všech případů použití. Zaměřte se na přihlašování uživatelů včetně OAuth a MFA, správu uživatelů, skupin a oprávnění a integraci REST API.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Moraes, E.: Jakarta EE Cookbook: Practical recipes for enterprise Java developers to deliver large scale applications with Jakarta EE, 2nd Edition, Packt Publishing Ltd, 2020
- Heffelfinger, D.R.: Java EE 8 Application Development: Develop Enterprise applications using the latest versions of CDI, JAX-RS, JSON-B, JPA, Security, and more, Packt Publishing, 2017

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 26. října 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTORIZAČNÉ A AUTENTIZAČNÉ RIEŠENIE NA PLATFORME JAVA

AUTHORIZATION AND AUTHENTICATION SOLUTIONS ON THE JAVA PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAROŠ GEFFERT**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Abstrakt

Cieľom tejto práce je zanalyzovať aktuálne autentizačne a autorizačné knižnice na platforme JAVA a vytvoriť webovú službu, ktorá podporuje autentizáciu uživateľov, vrátane viacfaktorového overenia (MFA), správu uživateľov, rolí a skupin a je odolná proti najčastejším útokom. Uskutočnil som analýzu už existujúcih riešení a na základe osvedčených postupov som vytvoril vlastné riešenie. Ako implementačný jazyk som použil Java 11 a knižnice Spring Framework, Spring Security, MyBatis a nástroj na zasielanie HTTP dotazov Postman. Implentovaná služba spĺňa stanovené kritéria, pričom som testoval funčknosť navrhnutých riešení a časovú náročnosť procesu kontroly prístupu.

Abstract

The aim of this work is to analyze the current authentication and authorization libraries on the Java platform and create a web application that supports user authentication, including multi-phase authentication (MFA), user and group authorization and is resistant to the most common attacks. I performed an analysis of existing solutions and based on best practises, I created my own solution. As an implementation language I used Java 11 and libraries Spring Framework, Spring Security, MyBatis and a tool for sending HTTP queries Postman. The implemented service satisfy the set criteria, while I tested the functionality of the proposed solutions and the time-consuming process of access control.

Kľúčové slová

autentizácia, autorizácia, MFA, OAuth, Java, porovnanie

Keywords

authentication, authorization, MFA, OAuth, Java, comparison

Citácia

GEFFERT, Maroš. *Autorizačné a autentizačné riešenie na platforme Java.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Autorizačné a autentizačné riešenie na platforme Java

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pana Ing. Radka Burgeta Ph.D. Ďalšie informácie mi poskytli Ing. Oldřich Ešner a Ing. Petr Mikušek. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Maroš Geffert
11. mája 2021

Poďakovanie

Chcel by som poďakovať vedúcemu mojej bakalárskej práce Ing. Radkovi Burgetovi za pomoc a cenné rady pri konzultáciách. Ďalej by som sa chcel poďakovať externým vedúcim z firmy Ki-Wi Digital Ing. Oldřichovi Ešnerovi a Ing. Petrovi Mikuškovi za pomocné rady, nápady a korekcie pri riešení tejto práce.

Obsah

Kapitola 1

Úvod

Jedna z najkľúčovejších oblastí v každej aplikácii je informáčná bezpečnosť. Informačná bezpečnosť predstavuje ochranu informácií vo všetkých jeho formách a počas celého cyklu ich života, teda počas ich vzniku, spracovania, ukladania, prenosu a likvidácie. Hlavne interné zabezpečenie dát je vo viacužívateľských aplikáciach nesmierne dôležité. Užívateľovi nesmie byť umožnené čítať, alebo dokonca editovať, či mazať dáta, na ktoré by nemal mať právo. Aplikácie obsahujúce takúto chybu, by mohli firmám urobiť veľmi veľké straty. Dnes existuje celá rada nástrojov a knižníc na riešenie tejto problematiky. Táto práca analyzuje a porovnáva tie aktuálne najpoužívanejšie.

V tejto práci som sa zameral na analýzu týchto nástrojov a knižníc a na základe overených postupov som vytvoril aplikáciu, ktorá spĺňa základné bezpečnostné politiky a sofistikovanejšie riadi užívateľov a skupiny.

K tejto téme ma naviedla firma Ki-Wi digital, ktorá si vyžaduje aktualizáciu modernejšieho spôsobu overovania totožnosti, vrátane viacfaktorovej autentizácie a prepracovanie verifikovania oprávnení pre užívateľov, role a skupiny. Tejto téme sa chcem venovať, pretože z môjho pohľadu, je bezpečnosť a ochrana dát vo webových aplikáciách nesmierne dôležitá. Môže to spôsobiť nemalé finančné a osobnostné škody.

Prvá kapitola je venovaná analýze technológií pre tvorbu serverových aplikácii a ich bezpečnosť. Zahŕňa to aplikačný server, technológie pre správu databázy, aplikačné rozhranie a výber najpopulárnejších autorizačných a autentizačných knižníc, kde dávam dôraz na ich možnosti zabezpečenia a odolnosť proti útokom.

Druhá kapitola porovnáva tieto najpopulárnejšie autorizačné a autentizačné knižnice. Medzi porovnávacie kritéria patrí architektúra knižnice, možnosti zabezpečenia, odolnosť voči útokom, popularita a kvalita zdrojov a dokumentácie.

V ďalšej kapitole je popis aplikácie, čo zahŕňa špecifikáciu požiadavok a prípadov užitia na základe žiadosti od firmy Ki-Wi Digital. Ďalej sa venuje výberu architektúry, návrhu doménového modelu, návrhu činnosti, prístupových bodov aplikačného rozhrania a výber technológií a knižníc na základe analýzy a porovnávacích kritérií.

Ďalšia kapitola popisuje už samotnú implementáciu webovej aplikácie a obzvlášť jej bezpečnosti. Je predstavená základná autentizácia uživateľa, viacfaktorová autentizácia ako overenie totožnosti pomocou e-mailu, alebo SMS správy, odolnosť voči útokom Cross-site scripting a Cross-site request forgery a prepracovaná správa oprávnení užívateľov, rolí a skupín.

Posledná kapitola obsahuje zrhnutie práce, testovanie implementovanej webovej služby a vyhodnotie dosiahnutých výsledkov.

Kapitola 2

Analýza technologií pre tvorbu serverových aplikácií

Môj rozsah práce zakladá na zoznámení sa so súčasnými technológiami pre tvorbu serverových aplikácií na platforme Java. Ďalej na analýze najaktuálnejších bezpečnostných knižníc na tejto platforme, ich porovnanie podľa stanovených kritérií a následne vytvorenie ukážkovej webovej aplikácie na základe nadobudnutých znalostí.

2.1 Technológie pre tvorbu serverových aplikácií

Java je robustný jazyk a v kombinácii s dobrými knižnicami môže poskytnúť najlepšie riešenie pre každú doménu, či už ide o elektronický obchod, bankovníctvo, financie a ďalšie. Existuje veľa technológií pre tvorbu serverových aplikácií postavených na Jave, kde každý z nich má svoje vlastné výhody a môže fungovať najlepšie pre rôzne prípady použitia.

2.1.1 Webový aplikačný framework

Webový aplikačný framework je nástroj, ktorý poskytuje šablóny a štruktúru pre vývoj aplikácií. S pomocou komponentných a softwarových knižníc má za cieľ vybudovať architektúru aplikácie. Umožňuje rýchlejší a ľahší vývoj webu tým, že poskytuje základné vzory pre vytváranie spoľahlivých, škálovateľných a udržiavateľných webových aplikácií. Existujú rôzne aplikačné frameworky ako napríklad Spring, Apache Struts, Grails.[4]

2.1.2 Aplikačný server

Aplikačný server je systémový softvér, na ktorom bežia webové aplikácie. Aplikačné servery pozostávajú z konektorov webového servera, programovacích jazykov, runtime knižníc, databázových konektorov a administratívneho kódu potrebného na nasadenie, konfiguráciu, správu a pripojenie týchto komponentov na webovom hostiteľovi. Aplikačný server beží za webovým serverom (napr. Apache alebo Microsoft Internet Information Services) a takmer vždy pred databázou SQL. Webové aplikácie bežia na vrchole aplikačných serverov a sú napísané v jazykoch, ktoré aplikačný server podporuje, a volajú runtime knižnice a komponenty, ktoré aplikačný server ponúka. Medzi príklady týchto aplikačných serverov, zahrniem Oracle WebLogic Server, Tomcat, alebo IBM WebSphere Application server.[7].

2.1.3 Databáza a knižnica perzistencie

Prepojenie databázy s java aplikáciou má na starosti framework perzistencie. Presúva dáta programu v najprirodzenejšej podobe (v pamäťových objektoch) do pernamentného uložiska dát. Framework perzistencie spravuje databázu a mapovanie medzi databázou a objektami a zjednodušuje tak proces vývoja.[18]

Pokiaľ ide v jave o prístup k relačným databázam, tak sa zvyčajne myslí na dve možnosti:

- SQL (Structured Query Language)
- ORM (Object Relational Mapping)

Pretože použitie SQL s Java API JDBC (Java Database Connectivity) je náchylné na chyby, prvou voľbou je zvyčajne ORM.

ORM

Cieľom rámca ORM je skryť prístup k databáze pred používateľom. Problém s ORM spočíva v tom, že uživateľ nemá úplnú kontrolu nad prístupom do databáze, čo môže spôsobiť niekoľko problémov. Najbežnejším problémom je zlý výkon spôsobený tým, že sa vývojári zvyčajne hlboko neponárajú do podrobností knižnice. Tento prístup zvyčajne vedie k príliš veľkému počtu príkazov SQL, ktoré vykonáva knižnica ORM. Ďalším problémom je náchylnosť k útoku SQL injection, kedže vývojár nevie ovplyvniť to, akým spôsobom sa dotazy aplikujú na databázu.[13]

Na trhu existuje veľa takýchto frameworkov (či už otvorených, alebo komerčných), ako je napríklad MyBatis, Spring Data, alebo Hibernate.

2.1.4 Aplikačné rozhranie REST

Aplikačné rozhranie REST (tiež známe ako RESTful API) je aplikačné programové rozhranie, ktoré vyhovuje obmedzeniam architektonického štýlu REST a umožňuje interakciu s webovými službami RESTful. REST je skratka pre prenos reprezentačného stavu a vytvoril ju vedec Roy Fielding. REST si získava popularitu, ale existuje veľa debát a rastúcich obáv o modelovanie webových služieb REST.

Zjednodušuje a oddeľuje architektúru, čo umožňuje nezávislosť každej časti. Jednotlivé zdroje založené na zdrojoch sú identifikované v požiadavkách pomocou identifikátorov URI ako identifikátorov prostriedkov. Jednotné rozhranie, ktoré musia poskytovať všetky služby REST, je základom jeho návrhu.

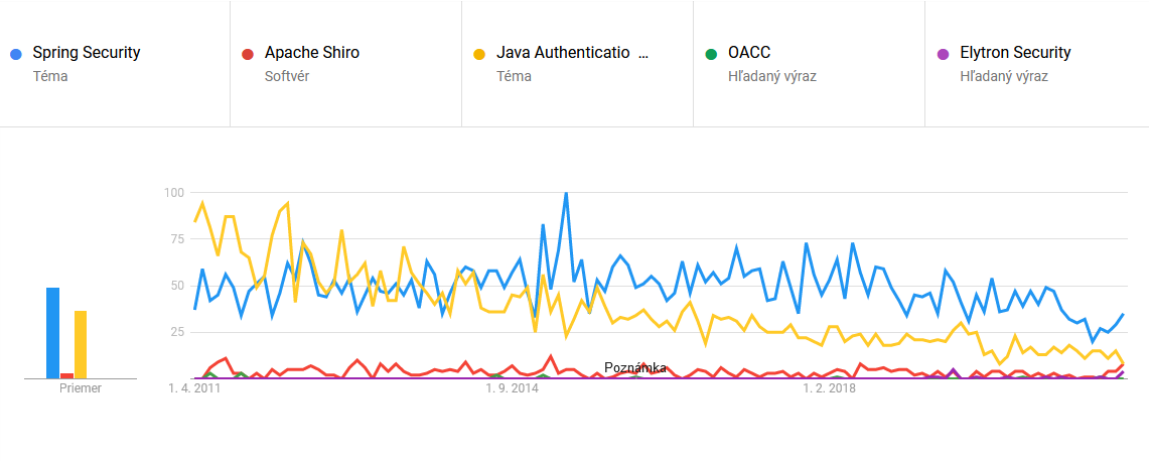
Keď sa požiadavka klienta uskutoční pomocou rozhrania RESTful API, prenesie sa reprezentácia stavu prostriedku na žiadateľa alebo do koncového bodu. Tieto informácie alebo reprezentácia sa doručujú v jednom z niekoľkých formátov prostredníctvom protokolu HTTP, JSON, HTML, obyčajný text a pod. JSON je najpopulárnejší programovací jazyk, ktorý sa používa, pretože napriek svojmu názvu je jazykovo agnostický a je dobre čitateľný ľuďmi aj strojmi.[22]

2.2 Autentizačné a autorizačné knižnice

Vo svete java existuje niekoľko bezpečnostných knižníc, no nie všetky sú pravidelne aktualizované a udržiavané. Preto si v tejto sekcii predstavíme tie najpopulárnejšie, ktoré budeme v ďalších kapitolách porovnávať.

2.2.1 Výber najpopulárnejších knižníc

S použitím nástroja Google Trends som zostavil graf, ktorý zobrazuje počet vyhľadávaných výrazov v čase od roku 2011 až po súčastnosť. Vyhľadávané boli názvy rôznych bezpečnostných frameworkov. Tento graf môže byť mierne skreslený, keďže sa do úvahy neberú verzie daných frameworkov.



Obr. 2.1: Graf najpopulárnejších bezpečnostných knižníc podľa nástroja Google Trends.

Na začiatku roku 2011 bol na vrchole JAAS (Java Authentication and Authorization Service), kedy sa začal dostávať vysoko do povedomia už Spring Security. V nasledujúcich rokoch Spring prevýšil JAAS. V súčastnosti JAAS stále stráca na svojej popularite a z úzadia ho začína predbiehať Apache Shiro a Elytron Security. Z grafu je vidno, že aktuálne tri najpopulárnejšie bezpečnostné knižnice sú Spring Security, JAAS a Apache Shiro a práve týmto by som sa chcel v tejto práci venovať.

2.2.2 Spring Security

Spring Security je jedným z bezpečnostných modulov Spring frameworku. Projekt bol zahájený v roku 2003 pod názvom „Acegi Security“, tým že bol verejne vydaný pod licenciou Apache v roku 2004. Následne bola Acegi začlenená do portfólia Spring ako Spring Security. Spring security je výkonný a vysoko prispôsobiteľný framework pre overovanie a riadenie prístupu. Jedná sa o štandard pre zabezpečenie aplikácií založených na frameworku Spring. Skutočná sila tohto frameworku sa nachádza v tom, ako sa dá jednoducho rozšíriť tak, aby spĺňal vlastné požiadavky.[19]

[0]Spring Security - https://docs.spring.io/spring-security/site/docs/current/reference/html5/
[0]Apache Shiro - https://shiro.apache.org/documentation.html
[0]JAAS - https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/tutorials/
[0]OACC - http://oaccframework.org/documentation.html
[0]Elytron Security - https://docs.wildfly.org/17/WildFly_Elytron_Security.html

Tento framework poskytuje[8]:

- LDAP (Lightweight Directory Access Protocol)
- Signle Sign-on
- Autentizáciu
- Remember me
- Autorizáciu
- podporu zabezpečenia proti CSRF a XSS útokom
- a mnoho ďalšieho...

2.2.3 Apache Shiro

Pred vstupom do Apache Software Foundation v roku 2008 mal Shiro už 5 rokov a bol známy ako JSecurity. V tej dobe existoval len JAAS. Jeho autentizačné schopnosti boli do istej miery prijatelné, no aspekty autorizácie boli bezmyšlienkové a frustrujúce. JAAS bol silno zviazaný s obavami o zabezpečenie na úrovni virtuálneho stroja, napríklad s určovaním, či by malo byť povolené načítanie triedy v JVM (Java Virtual Machine). Shiro tím prišiel stým, že vývojárom aplikácie záleží viac na tom, čo môže koncový uživateľ aplikácie robiť, než na tom, čo by kód mohol robiť vo vnútri JVM.

Apache Shiro je výkonný a ľahko použiteľný bezpečnostný framework, ktorý dokáže overovať uživateľov, autorizáciu, kryptografiu a správu relácií. Je ho možné použiť k zabezpečeniu akejkoľvek aplikácie.

Poskytuje aplikačné bezpečnostné api pre vykonávanie nasledujúcich aspektov:

- Preukázanie totožnosti uživateľa
- Kontrola prístupu
- Kryptografia
- Správa relácie
- Remember me
- LDAP

Shiro tiež podporuje niektoré pomocné funkcie, ako je zabezpečenie webových aplikácií, testovanie jednotiek a podpora multithreadingu, ale tieto existujú kvoli posílneniu výšie uvedených hlavných záujmov.[6]

2.2.4 Java Authentication and Authorization Service

Služba Java Authentication and Authorization Service (JAAS) bola predstavená ako voliteľný balík pre sadu Java 2 SDK 1.3. a neskôr bol integrovaný do J2SDK 1.4.

JAAS možno použiť na dva účely:

1. Na autentifikáciu používateľov, aby spoľahlivo a bezpečne určili, kto v súčasnosti vykonáva kód.

2. Na autorizáciu používateľov, aby sa zabezpečilo, že majú oprávnenia potrebné na vykonávanie vykonaných akcií.

JAAS implementuje verziu Java štandardného frameworku PAM (Pluggable Authentication Module). PAM je všeobecný framework pre autentizáciu, ktorý jednotlivým programom ponúka univerzálne API nezávislé na konkrétnom spôsobe autentizácie. PAM má modulárnu architektúru, ktorá poskytuje správcovi systému vysokú flexibilitu v úpravách pravidiel pre prihlasovanie.

Java tradične poskytuje kontroly prístupu založené na zdrojových kódoch. Chýbala mu však schopnosť dodatočne vynucovať riadenie prístupu na základe toho, kto kód spúšťa. JAAS poskytuje framework, ktorý o takúto podporu rozširuje bezpečnostnú architektúru Java.[17]

Kapitola 3

Porovnanie autorizačných a autentizačných knižníc

V následujúcej kapitole sa určia kritéria na základe ktorých budem porovnávať tri zvolené bezpečnostné knižnice. Tieto kritéria analyzujú, do akej miery knižnica podporuje určitú vlastnosť, architektúra, aká je popularita knižnice, poskytnutá kvalita dokumentácie a dostupnosť zdrojov. Kritéria budú ohodnocované slovne napr. (podporuje, nepodporuje, čiastočne podporuje, vysoká alebo nízka náročnosť a pod.).

Všetky kritéria sú členené do kategórií. V mojej práci som si zadefinoval celkom štyri kategórie, podľa ktorých budem porovnávať.

- Architektúra knižnice
- Možnosti zabezpečenia
- Popularita
- Poskytnutá dokumentácia
- Dostupnosť zdrojov

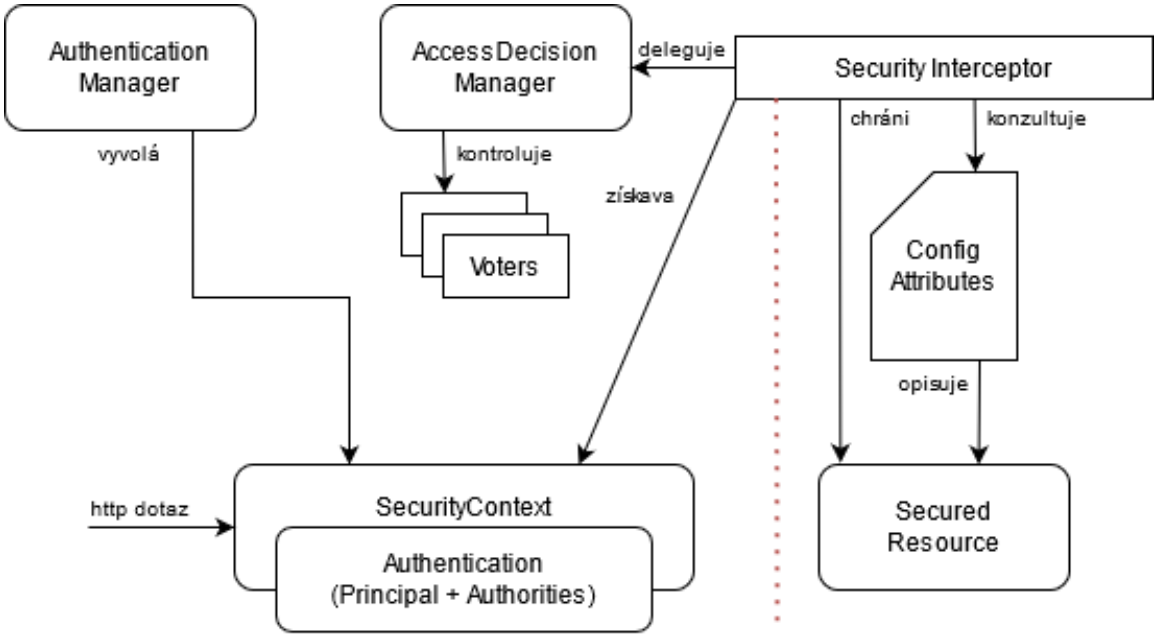
Výsledné porovnanie teda pozostáva z viackriteriálneho hodnotenia týchto variant.

3.1 Architektúra knižnice

Spring Security

V Spring security existuje 5 základných konceptov:

- Autentifikácia (kto ste)
- Autorizácia (môžete získať prístup)
- Principal (aktuálne prihlásený užívateľ)
- Granted authority (oprávnenie)
- Roly (skupina povolení)



Obr. 3.1: Architektúra knižnice a jej moduly v Spring Security [24]

Pridáva doň filtre, aby zachytil požiadavky predtým, ako idú na servlet. V obrázku 3.1 je uvedený proces autentifikácie a autorizácie tejto knižnice.

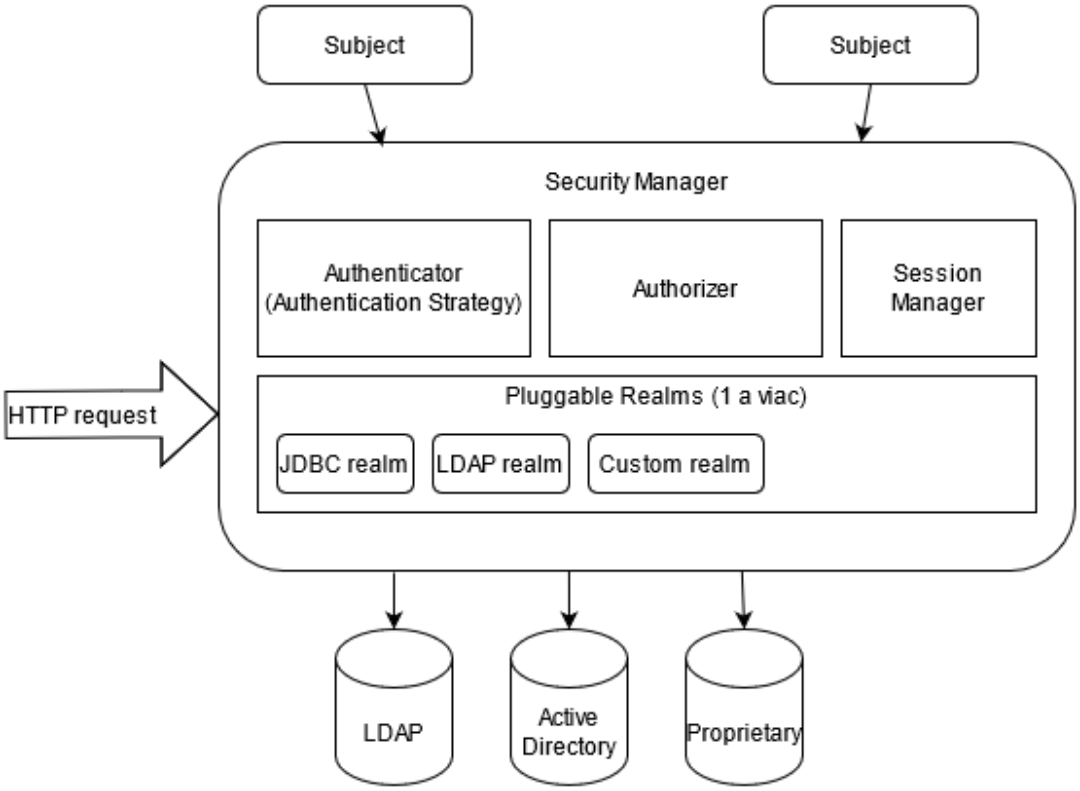
`AuthenticationManager` spracováva požiadavky na autentifikáciu. Bezpečnostný kontext je „miesto“, kde sú uložené informácie týkajúce sa užívateľa a zoznam autorizácií tohto užívateľa.

Volajúca metóda musi prejsť cez `SecurityInterceptor` predtým, ako je metóda skutočne vykonaná. Server získava autorizačné informácie z bezpečnostného kontextu, ktorý obsahuje informácie o užívateľovi a jeho oprávneniach, aby mohol tieto operácie skontrolovať.

`AccessDecisionManger` neimplementuje svoje vlastné rozhodnutie o prístupe, ale namiesto toho používa voličov, ktorí hlasujú za udelenie, alebo odmietnutie prístupu. V prípade, že autentifikácia zlyhá, knižnica vyvolá výnimku `AuthenticationException`, ktorá sa vráti naspäť k filtru, takže ju možno zachytiť alebo ukázať používateľovi[10].

Apache Shiro

Cieľom aplikácie Apache Shiro je zjednodušiť zabezpečenia aplikácií tým, že sú intuitívne a ľahko použiteľné. Na najvyššej koncepčnej úrovni má architektúra Shiro 3 základné koncepty: `Subject`, `SecurityManager` a `Realms`. Nasledujúci diagram predstavuje prehľad na koncepčnej úrovni o tom, ako tieto komponenty interagujú.



Obr. 3.2: Architektúra a jej moduly v knižnici Apache Shiro [5]

Subjekt

`Subjekt` je v podstate bezpečnostný „pohľad“ aktuálne pôsobiaceho používateľa. Všetky inštancie subjektu sú viazané s komponentom `SecurityManager`. Pri vzájomnom pôsobení so Subjektom sa tieto interakcie premenia na interakcie špecifické pre daný Subjekt pomocou nástroja SecurityManager.

SecurityManager

`SecurityManager` je srdcom architektúry frameworku Shiro a funguje ako „zastrešujúci“ objekt, ktorý koordinuje jeho komponenty, ktoré spolu tvoria graf objektu. Spravuje tiež pohľad na každého používateľa aplikácie, takže vie, ako vykonávať bezpečnostné operácie na používateľa.

Realms

`Realms` fungujú ako „konektory“ medzi knižnicou Shiro a bezpečnostnými údajmi aplikácie. Keď nastane čas skutočne interagovať s údajmi súvisiacimi so zabezpečením, ako sú napríklad používateľské účty, s cieľom vykonať autentifikáciu a autorizáciu, vyhľadá Shiro veľa z týchto vecí z jedného alebo viacerých `Realmov` nakonfigurovaných pre aplikáciu.

V tomto zmysle je `Realm` v podstate bezpečnostný DAO. Zapuzdruje detaily spojenia a podľa potreby sprístupňuje súvisiace údaje knižnici Shiro. Pri konfigurácii sa musí špecifiko-

vať aspoň jeden `Realm`, ktorý sa má použiť na autentifikáciu a autorizáciu. `SecurityManager` môže byť nakonfigurovaný s viacerými `Realms`, ale je vyžadovaný minimálne jeden.

Shiro poskytuje hotové `Realms` na pripojenie k množstvu zdrojov bezpečnostných adresárov, ako je LDAP, relačné databázy (JDBC), zdroje textovej konfigurácie ako INI, súbory vlastností a ďalšie. Tak isto je možnosť si doplniť svoju vlastnú implementáciu `Realmu` tak, aby reprezentovali vlastné zdroje dát.

Authenticator

Authenticator je komponenta, ktorá je zodpovedná za vykonávanie a reakciu pokusu používateľa o autentifikáciu. Keď sa používateľ pokúsi prihlásiť, túto logiku vykoná Authenticator. Autentikátor vie, ako sa koordinovať s jednou alebo viacerými `Realmami`, ktoré ukladajú príslušné informácie o používateľovi. Údaje získané z týchto oblastí sa používajú na overenie totožnosti používateľa, aby sa zaručilo, že používateľ je skutočne ten, za koho sa vydáva.

Authorizer

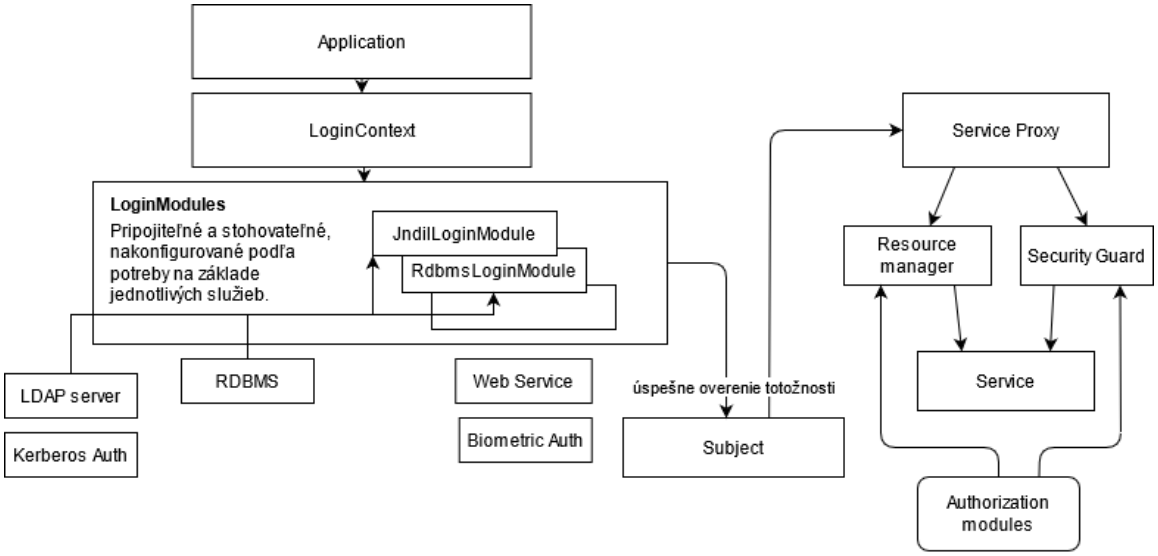
Autorizátor je komponenta zodpovedná za kontrolu prístupu používateľov v aplikácii. Je to mechanizmus, ktorý nakoniec hovorí, či má používateľ niečo povolené alebo nie. Rovnako ako Authenticator, aj Authorizer vie, ako sa koordinovať s viacerými zdrojmi údajov, aby získal prístup k informáciám o rolách a povoleniach. Autorizátor používa tieto informácie na presné určenie, či je používateľovi dovolené vykonať danú akciu.

Session Manager

SessionManager vie ako vytvárať a spravovať životné cykly relácií používateľov a poskytovať tak užívateľom robustné prostredie relácií vo všetkých prostrediach. Knižnica Shiro má schopnosť natívne spravovať relácie používateľov v akomkoľvek prostredí, aj keď nie je k dispozícii žiadny kontajner Web / Servlet alebo EJB. V predvolenom nastavení použije Shiro existujúci mechanizmus relácie, pokiaľ je k dispozícii (napr. Servlet Container), ale pokiaľ taký neexistuje, napríklad v samostatnej aplikácii, použije integrovanú správu podnikových relácií, ktoré ponúkajú rovnakú skúsenosť s programovaním.[1]

JAAS

JAAS definitívne zjednodušil vývoj bezpečnosti v Jave zavedením abstraktnej vrstvy medzi aplikáciou a používaným mechanizmom autentifikácie a autorizácie. Táto abstrakcia pomáha pri používaní rôznych bezpečnostných mechanizmov podľa výberu bez zmeny kódu v aplikácii.



Obr. 3.3: Proces autentizácie a autorizácie uživateľov v knižnici Java Authentication and Authrozation Service [11]

Ako vidíme na obrázku 3.3 aplikácia priamo interaguje s LoginContext. LoginContext funguje ako rozhranie medzi aplikáciou a sadou jedného alebo viacerých modulov LoginModules, ktoré sú priamo nakonfigurované. Tieto moduly LoginModules sú zodpovedné za spracovanie autentifikácie pomocou bezpečnostnej infraštruktúry. JAAS poskytuje aj referenčné implementácie LoginModule a môžeme vyvíjať vlastné prepínacie moduly. Jednoduchý konfiguračný súbor stačí na to, aby pomohol pri nastavovaní aplikácie s výberom implementácií.[21]

Autorizácia JAAS rozširuje existujúcu bezpečnostnú architektúru Java, ktorá pomocou bezpečnostnej politiky určuje, aké prístupové práva sú udelené vykonávaciemu kódu. Táto architektúra je zameraná na kód. To znamená, že povolenia sa udeľujú na základe charakteristík kódu: odkiaľ kód pochádza a či je podpísaný, a pokiaľ áno, tak kým.[9]

3.2 Možnosti zabezpečenia

Možnosti zabezpečenia je možné zhodnotiť podľa spôsobu autentizácie, autorzácie, odolnosti voči rôznym typom útokov ako je napríklad CSRF, alebo XSS a možnosti ponúkaných rozšírení zabezpečenia ako je napríklad oAuth.

3.2.1 Z pohľadu autentizácie

Autentizácia je proces rozpoznávania identity používateľa. Je to mechanizmus priradenia prichádzajúceho dotazu k množine identifikačných údajov. Poskytnuté identifikačné údaje „credentials" sa porovnajú s údajmi v databáze s informáciami oprávneného používateľa, alebo v rámci autentifikačného servera. Najbežnejší spôsob overenia totožnosti používateľa je overenie používateľského mena a hesla. Autentizáciu uživateľov rozdeľujeme do troch kategórií.

- Basic Authentication
- Form Login

- Digest Authentication

Basic Authentication

Základné overenie je formálne definované v štandarde Hypertext Transfer Protocol, RFC 1945. Keď sa klient pripojí k webovému serveru, odošle správu „WWW-Authenticate: Basic“ v hlavičke HTTP. Krátko na to odošle prihlasovacie údaje na server pomocou kódovania base64. Pri použití protokolu HTTPS sú tieto „credentials“ chránené, takže sa to nepovažuje za nezabezpečené, a preto si základné oprávnenie v priebehu rokov získalo rozsiahle využitie. Najväčší problém základného overenia súvisí s odhlásením zo servera, pretože väčšina prehľadávačov má tendenciu ukladať relácie do medzipamäte a nekonzistentne rieši potrebu správneho ukončenia a vymazania stavov pripojenia, aby sa iný používateľ nemohol prihlásiť s obnovením prehliadača.[14]

Form Login

Overovanie na základe formulára nie je formalizované žiadnym RFC. V podstate ide o programovú metódu autentifikácie, ktorú vývojári vytvárajú, aby zmiernili nevýhodu základnej autentifikácie. Väčšina implementácií overovania na základe formulára má nasledujúce vlastnosti:

1. Nepoužívajú formálne techniky autentifikácie HTTP.
2. Na prenos hodnôt používateľského mena a hesla na server používajú štandardné polia formulára HTML.
3. Server overí „credentials“ a potom vytvorí „reláciu“, ktorá je viazaná na jedinečný kľúč, ktorý sa odovzdáva medzi klientom a serverom pri každej požiadavke http.
4. Keď používateľ klikne na „odhlásiť sa“ alebo server ho odhlási (napríklad po určitej dobe nečinnosti), server zneplatní kľúč relácie, čo spôsobí, že každá následná komunikácia medzi klientom a serverom bude vyžadovať opätovné overenie prihlasovacích údajov prostredníctvom formulára, aby bolo možné vytvoriť nový kľúč relácie.[12]

Digest Authentication

Overenie prístupu Digest je jednou z metód, ktoré môže webový server použiť na vyjednanie prihlasovacích údajov, pomocou webového prehliadača používateľa. To sa dá použiť na potvrdenie totožnosti používateľa pred odoslaním citlivých informácií. Toto overenie aplikuje hashovaciu funkciu na používateľské meno a heslo pred ich odoslaním cez sieť. Naopak, základná autentifikácia prístupu používa namiesto hašovania ľahko reverzibilné kódovanie Base64, takže je nezabezpečená, pokiaľ sa nepoužíva v spojení s TLS[23].

Porovnanie

Framework	Basic Authentication	Form Login	Digest Authentication
Apache Shiro	podporuje	podporuje	podporuje
Spring Security	podporuje	podporuje	podporuje
JAAS	podporuje	podporuje	podporuje

Tabuľka 3.1: Porovnanie frameworkov s ohľadom na možnosti základnej autentizácie, digest autentizácie, alebo overením pomocou formulára.

3.2.2 Z pohľadu autorizácie

Autorizácia alebo kontrola prístupu je funkcia špecifikovania prístupových práv k zdrojom. Inými slovami, kto má k čomu prístup. Autorizácia má tri základné prvky - oprávnenia, roly a používatelia.

Povolenia

Povolenia sú najatomickejšou úrovňou bezpečnostnej politiky a sú vyhláseniami o funkčnosti. Povolenia predstavujú to, čo je možné urobiť v aplikácii. Dobre sformulované povolenie popisuje typy zdrojov a aké akcie sú možné pri interakcii s týmito prostriedkami.

- Je možné prečítať súbor?
- Môže sa odstrániť záznam zákazníka?

Povolenia predovšetkým určujú akcie (otváranie, čítanie, mazanie atď.) na prostriedku (súbor, záznam zákazníka atď.). V rámci kontroly prístupu existuje niekoľko úrovní povolení.[16]

- **Úroveň zdrojov** - toto je najširšia a najjednoduchšia tvorba. Zdroj je zadaný, ale nejde o konkrétnu inštanciu tohto zdroja. Napríklad uživateľ môže upravovať záznamy zákazníkov.
- **Úroveň inštancie** - Povolenie určuje inštanciu prostriedku. Napríklad uživateľ môže upraviť záznam zákazníka.
- **Úroveň atribútu** - Povolenie teraz určuje atribút inštancie alebo zdroja. Uživateľ môže upraviť adresu v zázname zákazníka.

Roly

V súvislosti s autorizáciou sú roly efektívnym súborom povolení, ktoré sa používajú na zjednodušenie správy povolení. Používateľom teda možno priradiť roly namiesto toho, aby im boli priamo pridelené oprávnenia. Napríklad banková aplikácia môže mať napríklad rolu správcu alebo rolu bankového pokladníka.

Existujú dva typy rolí.

- Implicitné roly
- Explicitné roly

Implicitná rola implikuje množinu správania založených iba na názve roly. Explicitná rola má vyslovene pridelené povolenia, a preto predstavuje explicitnú kolekciu povolení. Veľkými výhodami explicitných rolí sú ľahšia ovládateľnosť a nižšia údržba aplikácie.[1]

Užívateľ

Užívatelia môžu v aplikácii vykonávať určité akcie prostredníctvom ich priradenia k rolám alebo priamym povoleniam. JAAS výslovne nedefinuje roly ani skupiny. Namiesto toho sú implementované ako konkrétne triedy, ktoré používajú rozhranie `java.security.Principal`. Tak isto nedefinuje ako podporovať hierarchiu riadenia prístupu na základe rolí (RBAC), v ktorej môže role udeliť. V ďalších prípadoch sa z pohľadu autorizácie knižnice líšia len minimálne. V konečnom výsledku prinášajú podobnú kontrolu prístupu na úrovni užívateľov, rolí (okrem JAAS) a povolení, čo môžeme vidieť v tabuľke nižšie.[17]

Framework	Role Check	Permission Check
Apache Shiro	hasRole()	isPermitted()
Spring Security	hasRole()	hasAuthority()
JAAS	X	checkPermission()

Tabuľka 3.2: Porovnanie frameworkov s ohľadom na možnosti kontroly prístupu rolí a oprávnení.

3.2.3 OAuth

OAuth je autorizačný protokol, ktorý popisuje, ako nesúvisiace servery a služby môžu bezpečne povoliť autentizovaný prístup k svojim aktívam bez toho, aby skutočne zdieľali počiatočné súvisiace prihlasovacie údaje. V reči autentizácie sa to nazýva zabezpečená, tretia strana, užívateľský agent, či delegovaná autorizácia.

OAuth, ktorý od začiatku vytvoril a silne podporuje Twitter, Google a ďalšie spoločnosti, bol vydaný ako otvorený štandard v roku 2010 ako RFC 5849 a rýchlo sa stal všeobecne prijatým. V priebehu nasledujúcich dvoch rokov prešiel podstatnou revíziou a bola vydaná verzia 2.0.[15]

Najjednoduchším príkladom protokolu OAuth je prihlásenie na web, ktorý ponúka jednu alebo viac príležitostí na prihlásenie pomocou prihlásenia na inom webe / službe. Po vybraní poskytovateľa prepojeného s iným webom, užívateľa overí a web, ku ktorému sa pôvodne prihlasovalo, potom sám povolí prístup pomocou povolenia získaného od druhého webu.

Spring Security má rozšírenia poskytujúce podporu pre oba Oauth, kde na druhej strane to Apache Shiro a JAAS nepodporuje.

3.2.4 Odolnosť voči útokom

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) je útok, ktorý núti koncového používateľa vykonávať nežiaduce akcie vo webovej aplikácii, v ktorej je momentálne autentizovaný. S malou pomocou sociálneho inžinierstva (napríklad odoslaním odkazu prostredníctvom e-mailu alebo rozhovoru) môže útočník oklamať používateľov webovej aplikácie, aby vykonali akcie podľa jeho výberu. Ak je obeťou normálny používateľ, úspešný útok CSRF môže používateľa prinútiť k vykonaniu požiadaviek na zmenu stavu, ako je prevod finančných prostriedkov,

zmena ich e-mailovej adresy atď. Ak je obeťou administratívny účet, CSRF môže ohroziť celú webovú aplikáciu.

Cross-Site Scripting

Cross-Site Scripting (XSS) útoky sú typom injekcie, kedy sa škodlivé skripty injektujú na inak neškodné a dôveryhodné webové stránky. K útokom XSS dochádza, keď útočník použije webovú aplikáciu na odoslanie škodlivého kódu, zvyčajne vo forme skriptu na strane prehliadača, inému koncovému používateľovi. Chyby, ktoré umožňujú tieto útoky uspieť, sú pomerne rozšírené a vyskytujú sa kdekoľvek, kde webová aplikácia využíva vstup od používateľa v rámci výstupu, ktorý generuje, bez toho, aby ho overovala alebo kódovala.

Útočník môže pomocou XSS poslať škodlivý skript nič netušiacemu používateľovi. Prehliadač koncového používateľa nijako nevie, že skriptu by sa nemalo dôverovať a skript vykoná. Týmto spôsobom môže škodlivý skript získať prístup k ľubovoľným súborom cookie, tokenom relácie alebo iným citlivým informáciám, ktoré prehliadač uchová a použije s danou stránkou. Tieto skripty môžu dokonca prepísať obsah stránky HTML.[3]

Framework	OAuth	ochrana proti CSRF	ochrana proti XSS
Apache Shiro	nepodporuje	nepodporuje	nepodporuje
JAAS	nepodporuje	nepodporuje	nepodporuje
Spring Security	podporuje	podporuje	podporuje

Tabuľka 3.3: Porovnanie frameworkov s ohľadom na podporu OAuth a prevencie proti Cross-site request forgery a Cross-site scripting útokom.

3.3 Popularita

Na začiatku kapitoly sme si ukázali popularitu jednotlivých knižníc s použitím nástroja Google Trends viz. 2.2.1. Teraz sa zameráme na porovnanie popularity z viacerých prespektív.

Na serveri Stack Overflow

Stack Overflow je stránka s otázkami a odpoveďami pre vývojárov. Obsahuje otázky a odpovede na širokú škálu tém programovania. Prvým kritériom porovnania bude teda počet otázok, kde má framework vlastný tag a k nemu prisluchajúci počet sledovateľov.

Framework	Počet otázok	Sledovatelia
Apache Shiro	1200	262
Spring Security	24.7k	13.8k
JAAS	871	78

Tabuľka 3.4: Počet sledovateľov a položených otázok na platforme Stack Overflow.

Z tabuľky je vidieť, že Spring Security je najsledovanejší framework s najväčším počtom otázok.

Na serveri GitHub

GitHub je webová platforma na správu a spoluprácu verzií pre vývojárov softvéru. GitHub uľahčuje sociálne kódovanie tým, že poskytuje webové rozhranie pre úložisko kódov Git. Porovnávať budem počet uživateľov tejto platformy, ktorí označili repozitáre github hviezdou. Jedná sa o spôsob akým sa sledujú zmeny v danom repozitári. Ďalším kritériom bude počet git forkov. Fork je operácia klonu git vykonávaná na serverovej kópií repozitory projektov.

Následujúca tabuľka nezahrňuje JAAS, pretože sa jeho implementácia nachádza na tejto platforme.

Framework	Github hviezdy	Forks
Apache Shiro	3,420	1,990
Spring Security	5,806	4,488
JAAS	X	X

Tabuľka 3.5: Počet github hviezd a forkov na platforme GitHub.

Pravidelnosť vývoja

Dôležitým aspektom pre vývoj knižnice je jeho pravidelná aktualizácia. Jedná sa o dátum vydania poslednej verzie danej knižnice.

Framework	Dátum vydania poslednej verzie	Verzia
Apache Shiro	January 2021	v2.0
Spring Security	November 2020	v5.5.0-M1
JAAS	February 2002	X

Tabuľka 3.6: Dátumy vydania posledných verzií.

3.4 Dostupnosť zdrojov a dokumentácia

Kvalitné zdroje a dokumentácia sú pri frameworkoch nesmierne dôležité, obzvlášť bezpečnostných. Predsa len jedna z najbežnejších zraniteľností v aplikácií je používanie komponent, kde nepoznáme chyby zabezpečenia.

Toto kritérium by som si dovolil hodnotiť mierne subjektívne, kedže nie je v mojích silách spočítať všetky počty článkov a návodov, ktoré sa zaoberajú danou problematikou.

Z môjho pohľadu sa dokumentácia Spring Security nesmierne líši od dokumentácie Shira, alebo JAAS. Nakoľko frameworky Apache Shiro a JAAS sú popísané pomerne stručne a bez nejakých praktických príkladov, tak na druhej strane Spring ponúka obshialy dokument, kde praktické príklady prevyšujú množstvo textu. Tak isto existuje plno návodov a článkov tretích stran, ktorých počet podľa môjho názoru prevyšuje počet návodov a článkov zvyšných dvoch frameworkov. Predsa len je to knižnica, ktorá má nesmierne silnú komunitu.

3.5 Vyhodnotenie kritérií

V tejto kapitole sme porovnávali tri najpopulárnejšie bezpečnostné frameworky postavené na platforme java. Videli sme, že vo väčšine kritérií zvíťazila knižnica **Spring Security**. Z môjho pohľadu má Spring Secuiry zaslúžene prvé miesto ako už z pohľadu použitia, tak aj prevedenia a údržby. V nasledujúcej kapitole predstavím webovú službu, v ktorej zabezpečenie bude implementované práve týmto frameworkom.

Kapitola 4

Návrh demonstračnej aplikácie

Existuje niekoľko prístupov ako v dnešnej dobe koncipovať architektúru aplikácií. Návrh aplikácie výchadza z princípov a technológií, ktoré boli popísané v predchádzajúcich kapitolách. V tejto kapitole sú špecifikované požiadavky, popis webovej služby, entity v systéme, prípady užitia, prístupové body a výber technologií s, ktorými budem danú webovú službu implementovať.

Cieľom aplikácie je demonštrovať vybraný bezpečnostný framework, s nasadením autentizácie, autorizácie, správu rolí a uživateľov.

4.1 Špecifikácia požiadavkov

Ako som už spomínal, tak táto práca má za účel zmodernizovať a vylepšiť autentizáciu a kontrolu prístupu uživateľov v našej webovej službe KiWi-Server. Zameráme sa na overenie prístupu uživateľa, kedy sa klasické overenie totožnosti posilní verifikáciou e-mailu a sms verifikáciou pomocou OTP kódu. Ďalej sa implementuje štandard oAuth a zavedie sa sofistikovaná správa uživateľov, skupin a rolí riadené datmi na úrovni inštancií. Následne sa vytvorí ochrana proti útokom Cross-Site request forgery a Cross-site scripting. Všetky tieto riešenia ukážem na demonstračnej webovej službe, následne otestujem a zhodnotím.

4.2 Popis webovej služby

Svet sa posledné obdobie trápi s pandémiou a dôsledkom toho začínajú vznikať systémy presne s touto problematikou. Preto som dostal nápad vytvoriť sýstem "pandemické centrum", ktoré by spravovalo požiadavky pacientov ako je napríklad test na ochorenie covid-19, očkovanie, test na protilátky a podobne. Systém taktiež zhŕňa uživateľov ako zamestnacov, ktorý by mali v danej službe určité roly a oprávnenia. Téma zdratníctvo je veľmi citlivá a je určite nežiadújce, aby sa k citlivým údajom dostali ľudia, ktorý nato nemajú, alebo nemusia mať oprávnenie. Kvalitné zabezpečenie je preto nevyhnutné. V nasledujúcej kapitole si rozoberieme celú štruktúru navrhovanej webovej služby.

4.3 Entity v systéme

1. **Pacient (Patient)**
   Osoba, ktorá sa prihlásila na vyšetrenie, alebo úkon spojený s pandemickým centrom. Obsahuje data o pacientovi: rodné číslo, meno, kontaktné údaje a pod.

2. **Uživateľ (User)**
   Registrovaný uživateľ v systéme. Obsahuje data o uživateľovi, email ako prihlasovacie meno a heslo

3. **Návšteva (Visit)**
   Návšteva daného pacienta, ktorý požiadal o vyštrenie, alebo úkon. Obsahuje informácie o návšteve daného pacienta ako je čas príchodu, typ vyštrenia, symptomy a podobne.

4. **Úkon (Procedure)**
   Vyštrenie, alebo úkon pre pacienta. Obsahuje informácie o úkone pacienta ako typ úkonu (test, očkovanie), výsledok, cena atď.

5. **Rola (Role)**
   Obsahuje data o uživateľskej roli v systéme. Názov role ako identifikátor.

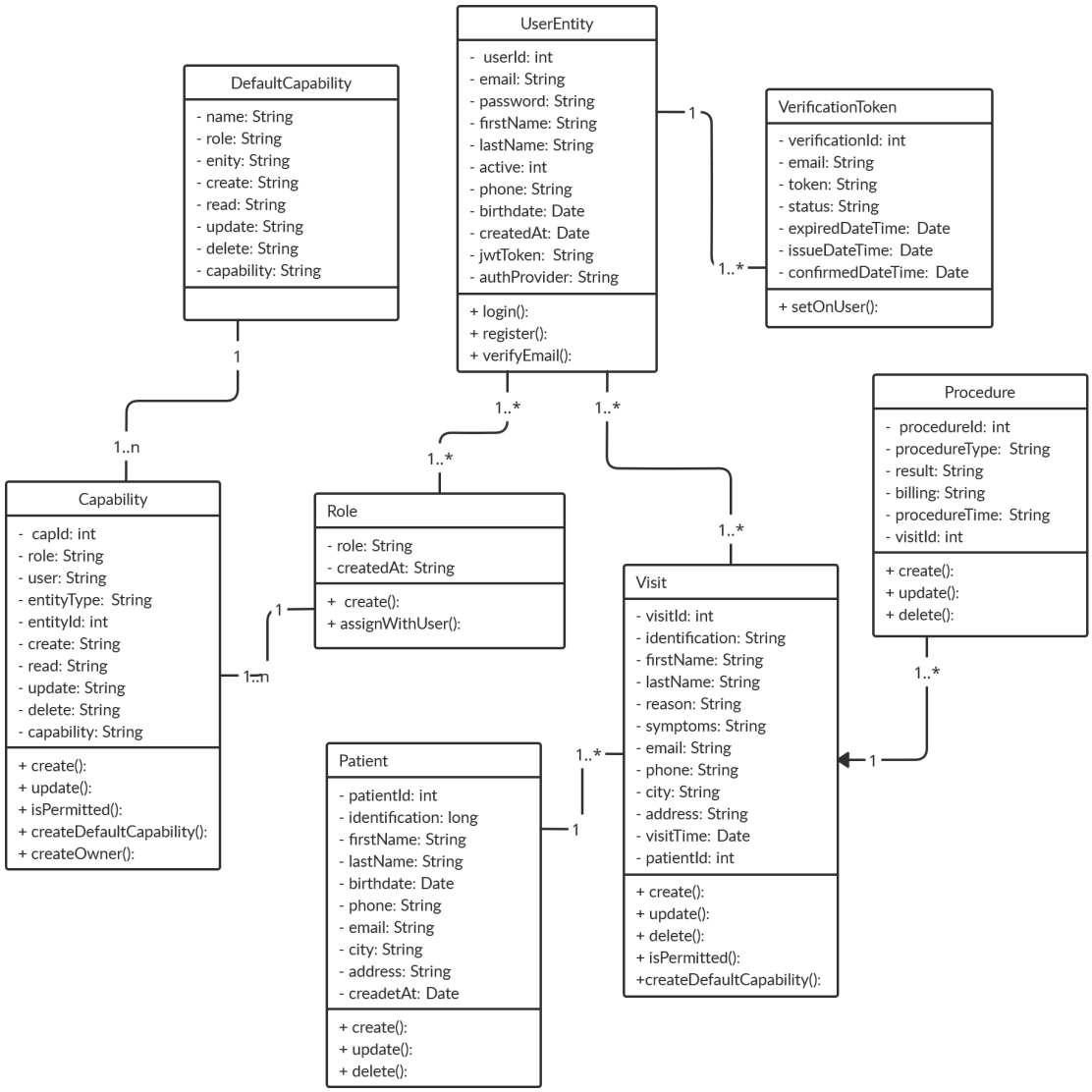
6. **Schopnosť (Capability)**
   Umožňuje autorizáciu uživateľov. Drží informácie o tom k akej entite má uživateľ alebo skupina prístup a čo s danou entitou môže robiť.

7. **Predvolená Schopnosť (DefaultCapability)**
   Umožňuje definovať predvolené oprávnenia pre jednotlivé role. Obsahuje rolu, entitu a množinu povoľovacích operácií, ktoré sa naviažú na oprávnenie.

8. **Verifikačný Token (VerificationToken)**
   Token sa využíva v spojení s verifikáciou e-mailu uživateľa. Obsahuje identifikátor uživateľa (v mojom prípade e-mail), unikátny token a čas expirácie.

4.4 Diagram tried

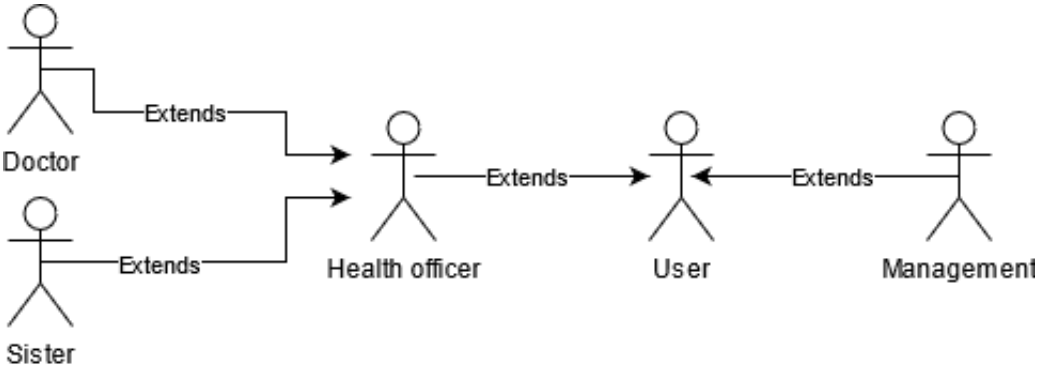
Účelom diagramu tried je modelovať statické zobrazenie aplikácie. Diagramy tried je jediný diagram, ktorý je možné priamo mapovať pomocou objektovo orientovaných jazykov a ktoré sa tak v čase vývoja široko používajú.



Obr. 4.1: Diagram tried v navrhovanej demonstračnej aplikácií

4.5 Prípady užitia rolí

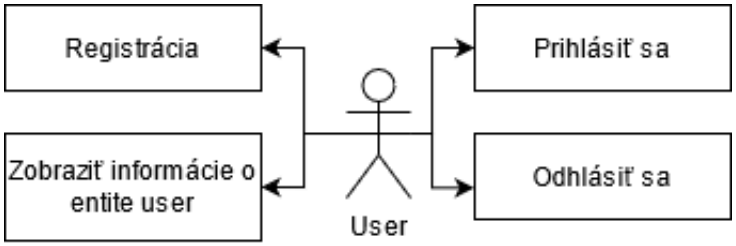
Prípady užitia som nadefinoval tak, aby som pokryl väčšinu situácií kontroly prístupu a ukázal tak na minime entít flexiblitu autorizačného návrhu. Roly a ich relácie sú zakreslené v nasledujúcom diagrame.



Obr. 4.2: Schéma rolí a ich relácií v systéme

Bežný uživateľ

Prípady užitia pre rolu `USER`. Rola `USER` je základná rola pre každého registrovaného užívateľa.



Obr. 4.3: Prípady užitia pre rolu User.

Táto rola nemá nejaké špeciálne prípady užitia, skôr slúži na ukážku overenia totožnosti pomocou oAuth a verifikáciu e-mailu.

Prihlásiť sa

Každému uživateľovi je umožnené sa prihlásiť do webovej služby pomocou e-mailu a hesla.

Odhlasiť sa

Každému uživateľovi je umožnené sa odhlásiť z tejto webovej služby.

Registrácia

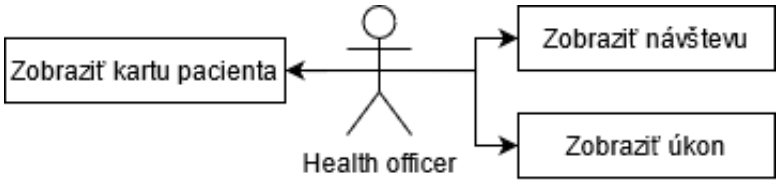
Všetci užívatelia sa môžu registrovať akýmkoľvek dostupným spôsobom (klasická registrácia, oAuth).

Zobraziť informácie o entite

Uživatelia môžu zobraziť informácie o svojom účte, ktoré zhromažďuje systém za behu tejto webovej služby.

Zdravotnícky personál

Prípady užitia pre rolu `EMPLOYER`. Rola `EMPLOYER` je rola pre každého uživateľa zdravotníckeho personálu.



Obr. 4.4: Prípady užitia pre rolu Health officer.

Zobraziť kartu pacienta

Zdravotnícky personál si môže zobraziť detaily všetkých pacientov, napríklad osobné alebo kontaktné údaje.

Zobraziť návštevu

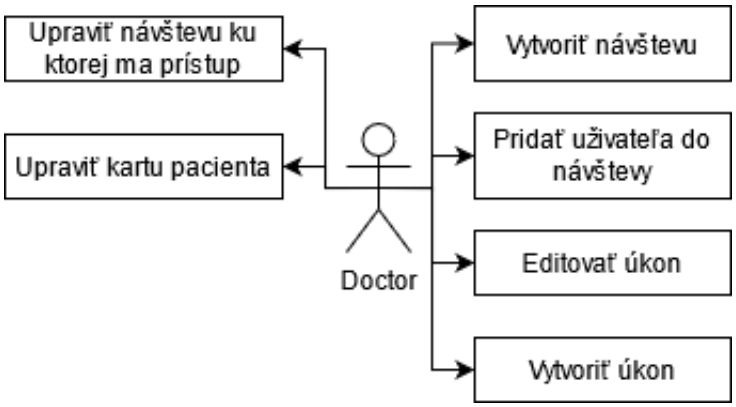
Zdravotnícky personál môže zobraziť návštevu pacienta.

Zobraziť úkon

Zdravotnícky personál môže čítať úkony pacienta v návšteve.

Ošetrovateľ

Prípady užitia pre rolu ošetrovateľ (`DOCTOR`).



Obr. 4.5: Prípady užitia pre rolu Doktor.

Vytvoriť návštevu

Ošetrovateľ môže vytvoriť návštevu pre daného pacienta. Po vytvorení návšetvy sa uživateľ stáva súčasne vlastníkom návštevy, ktorá mu povoľuje všetky operácie s touto konkrétnou inštanciou.

Editovať návštevu

Ošetrovateľ môže upraviť svoju návštevu, alebo návštevu ku ktorej má prístup.

Editovať úkon

Ošetrovateľ môže upraviť úkon pacienta v návšteve ku ktorej má prístup.

Vytvoriť úkon

Ošetrovateľ môže vytvoriť úkon pre pacienta v návšteve ku ktorej má prístup. Po vytvorení úkonu sa uživateľ stáva súčasne vlastníkom úkonu, ktorá mu povoľuje všetky operácie s touto konkrétnou inštanciou.

Pridať uživateľa do návštevy

Ošetrovateľ môže pridať uživateľa do návštevy a tým mu umožniť prístup.

Sestra

Prípady užitia pre rolu sestra (`SISTER`).

Obr. 4.6: Prípady užitia pre rolu Sestra.

Upraviť návštevu ku ktorej má prístup

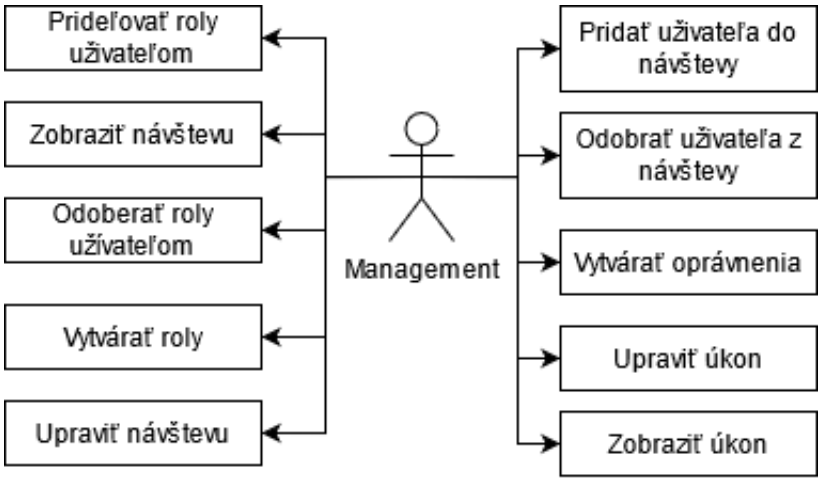
Sestra môže upraviť návštevu daného pacienta, ku ktorej má prístup, napríklad zmenou času návštevy.

Editovať / Vytvárať úkony v návšteve

Sestra môže upraviť alebo vytvoriť úkon v návšteve, ku ktorej má prístup. Po vytvorení úkonu sa uživateľ stáva súčasne vlastníkom úkonu, ktorá mu povoľuje všetky operácie s touto konkrétnou inštanciou.

Vedenie

Prípady užitia pre rolu vedenie (`MANAGEMENT`).



Obr. 4.7: Prípady užitia pre rolu Vedenie.

Pridelovať / odoberať roly

Vedeniu je umožnené prideľovať alebo odoberať roly užívateľom v systéme, vrátane tej svojej.

Vytvárať roly

Vedenie môže vytvárať roly pre užívateľov.

Vytvárať oprávnenia

Vedenie môže vytvárať oprávnenia pre rôznych užívateľov.

Zobraziť / upraviť úkon

Vedenie môže čítať alebo upraviť úkony pacienta v návšteve.

Pridať / odobrať uživateľa do / z návštevy

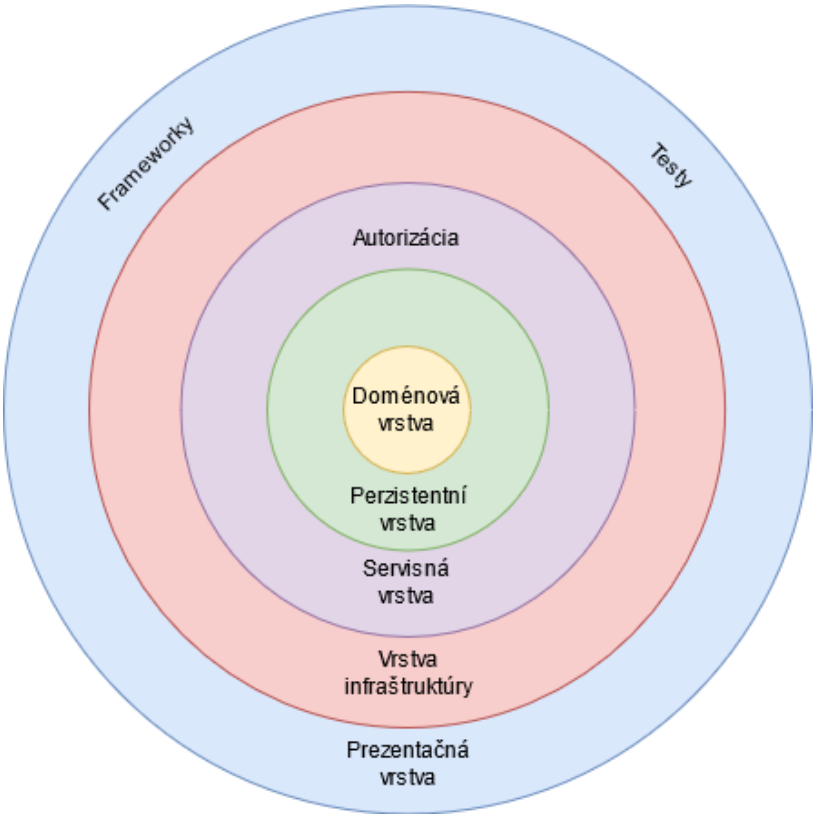
Vedenie môže pridať uživateľa do návštevy a tým mu umožniť prístup.

Zobraziť / Upraviť návštevu

Vedenie môže zobraziť, alebo upraviť návštevu akéhokoľvek pacienta.

4.6 Architektúra

Architektúra tejto webovej služby sa rozdeľuje do niekoľkých vrstiev. Návrh architektúry je koncipovaný tak, aby odpovedal čistej architektúre tzv. "clean architecture". Vrstva, ktorá



Obr. 4.8: Architektonický navrh jednotlivých vrstiev v systéme.

je zanorená, nesmie vedieť vôbec nič o vrstve, ktorá je nad ňou. Cieľom bolo dosiahnuť tieto základné vlastnosti.

- Nezávisle od frameworkov. Architektúra nezávisí od existencie nejakej knižnice softvéru nabitého funkciami. To umožňuje použiť framework len ako implementačný detail.
- Testovateľné. Bussiness pravidlá je možné testovať bez používateľského rozhrania, databázy, webového servera alebo iného externého prvku.
- Nezávislé od uživateľského rozhrania. Užívateľské rozhranie sa môže ľahko meniť bez zmeny zvyšku systému.
- Nezávislé na databáze.

Doménová vrstva

Predstavuje entity v aplikácií, ktoré reprezentujú jednotlivé triedy.

Perzistentná vrstva

Táto vrstva je zodpovedná za perzistenciu údajov a bussiness vrstva ju používa na prístup do vyrovnávacej pamäte a databázy.

Servisná vrstva

Servisná vrstva predstavuje abstrakciu nad doménovou logikou. Definuje hranicu aplikácie s vrstvou služieb, ktorá ustanovuje množinu dostupných operácií a koordinuje reakciu aplikácie v každej operácii. Vrstva má tak isto na starosti správu uživateľov a rolí a riadi kontrolu prístupu.

Vrstva infraštruktúry

Infraštruktúra implementuje rozhranie, ktoré je nevyhnutne potrebné pre správne fungovanie aplikačnej logiky systému. Zabezpečuje funkcionalitu pre odoslanie emailov a SMS správ, pridávanie užívateľov do návštev, generovanie textu pre užívateľov a pod.

Prezentačná vrstva

Táto vrstva nereprezentuje uživateľské rozhranie, ale sadu controllerou a api endpointov, ktorá vracia informácie vo formáte JSON. Webová služba získava požiadavky od uživateľov a po správnej autorizácii ich predáva vrstve infraštruktúry, ktorá ich ďalej spracováva.

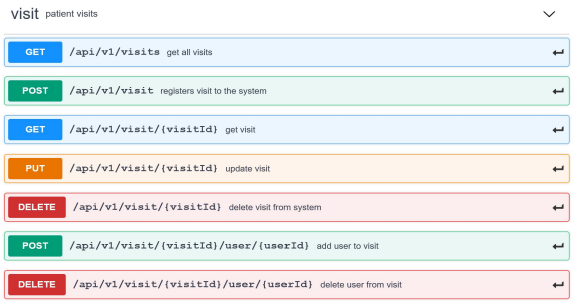
4.7 Prístupové body

Prístupové body, alebo „api endpointy" sú zjednodušene povedané konce komunikačného kanála. Keď rozhranie API interaguje s iným systémom, kontaktné body tejto komunikácie sa považujú za koncové body. V prípade rozhrania API môže koncový bod obsahovať adresu URL servera alebo služby.

Na vizualizáciu api endpointov som použil nástroj swagger. Jedná sa o open-source službu, ktorá umožňuje vývojárom designovať a dokumentovať REST API. Dôležitou súčasťou tohto nástroja je aj SwaggerUI. Toto rozšírenie tvorí grafické prostredie, ktoré poskytuje prehľadné prezeranie REST dokumentácie a zároveň ju umožňuje testovať bez nejakého externého nástroja ako je napríklad curl.

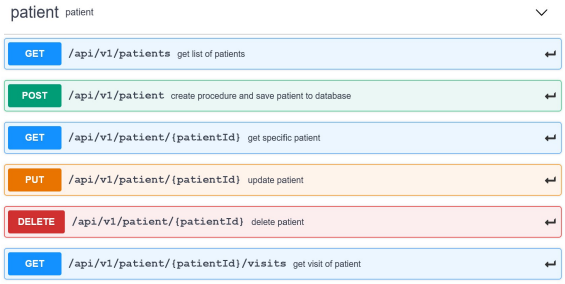
V následujúcej časti sú poskytutné api endpointy vizualizované práve nástrojom swagger.

Návšteva

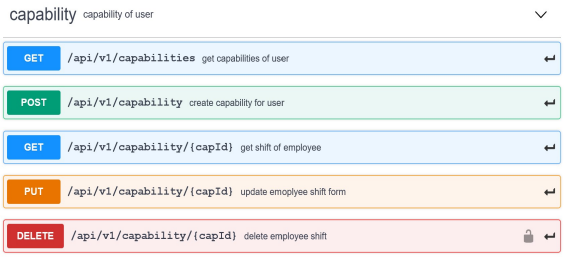


Úkon

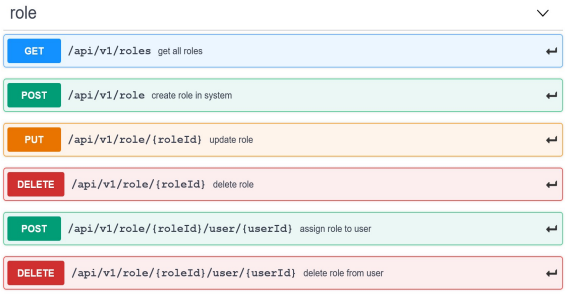
Pacient



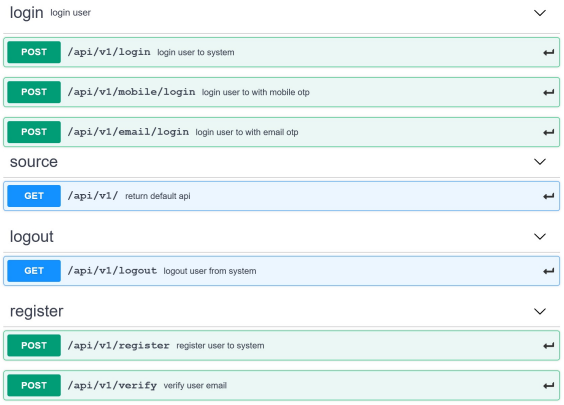
Schopnosti



Prihlásenie, registrácia



Role



4.8 Autentizácia a Autorizácia

K implementácií autentizácie využijem techniku overenia pomocou JWT (Jason Web Token). JSON Web Token je otvorený štandard (RFC 7519), ktorý definuje kompaktný a samostatný spôsob bezpečného prenosu informácií medzi stranami. Tieto informácie možno overiť a dôverovať im, pretože sú digitálne podpísané. Dotazy na neautentifikovaných užívateľov sa nedostanú ani ku kontrolérom, takže nebudú vôbec spracované. Klientovi sa vráti http error kód 401 (unauthorized). Ak je klient autentifikovaný tak dotaz sa príjme a prejde k ďalšiemu spracovaniu. Kedže ani jedna porovnávaná knižnica neposkytuje riešenie autentizácie pomocou jwt tokenu, musel som použiť na implementáciu tohto mechanizmu knižnicu „jjwt", ktorá overí každý prichádzajúci dotaz na server.

Schéma autentizácie

Webová služba využíva tokeny typu Bearer. Tento Bearer token sa predáva v hlavičke dotazu „Authorization".

Príklad tokenu:

Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJkb2N0b3JAZ21haWwuY29tIiwiZXhwIjoxNjE3NDY4OTQ2LCJpYXQiOjE2MTc0NTA5NDZ9.DosjbEtrZwHzjBe59JjpCQphlBYVYoUMuuJapf1S-tXWqeXlcEDSCXaJstI2kVtZwyzhnd5rSeJ4H1I5pNSb_Q

Za každým dotazom, kde si to koncový bod vyžaduje sa overí validita tohto tokenu. Ak

je uživateľ plne autentizovaný a token je validný (napr. nevypršala mu expirácia) tak je proces úspešný a pokračuje sa v spracovávaní dotazu.

Schéma autorizácie

Pri kontrole prístupu som zvolil autorizačný model RBAC (Role based access controll).

Kontrola prístupu na základe rolí

Kontrola prístupu na základe rolí (RBAC) obmedzuje prístup k sieti na základe rolí. Stala sa jednou z hlavných metód rozšírenej kontroly prístupu. Povolenia na role môžu byť zdedené prostredníctvom hierarchie rolí a zvyčajne odrážajú povolenia potrebné na vykonávanie definovaných funkcií. Daná rola sa môže vzťahovať na jedného jednotlivca alebo na niekoľko jednotlivcov.

4.9 Výber technológií

Výber technológií a knižníc pre moju demonstračnú aplikáciu vychádza z predchádzajúcich kapitol. Cieľom nebolo vybrať len tie najmodernejšie a najpopulárnejšie, ale zamerať sa aj na komunikáciu jednotlivých technológií medzi sebou.

4.9.1 Aplikačný framework

Pre moju aplikáciu som si vybral Spring framework. Spring je silný framework, ktorý sa zaoberá mnohými bežnými problémami v prostredí Java EE. Zahŕňa podporu pre správu bussiness objektov. Uľahčuje proces programovania, ako je napríklad programovanie pomocou rozhraní namiesto tried. Spring umožňuje vývojárom vyvíjať podnikové aplikácie pomocou programovania modelov POJO a POJI. Je modulárny a umožňuje používať iba tie časti, ktoré práve potrebujem.[20]

Najdôležitejšími dôvodomi, prečo som si ho vybral sú:

1. Oficiálna dokumentácia pokrýva prakticky všetko.
2. Poskytuje RESTful služby
3. Oficiálna webová stránka obsahuje aj sériu skvelých návodov vo video a textových formátoch.
4. Existujú odkazy na úložiská Github pre vzorové aplikácie Spring a existuje aj veľa tutoriálov tretích strán a to svedčí o tom, že Spring je tak široko využívaný mnohými skúsenými vývojármi.
5. Umožňuje nám vybrať si ľubovoľnú jeho časť izolovane.

4.9.2 Aplikačný server

Pre túto prácu som si vybral Tomcat server pretože podporuje všetky potrebné vlastnosti, je pomerne jednoduchý a vysoko flexibilný. Ponúka mimoriadnu úroveň bezpečnosti. Tomcat má k dispozícii množstvo kvalitnej dokumentácie vrátane širokej škály online návodov. Vďaka tomu je populárnou voľbou na vyplnenie úlohy aplikačného servera takmer vo všetkých webových aplikáciách Java.

4.9.3 Aplikačné rozhranie REST

Ako som už spomínal v predchádzajúcej sekcii 4.9.1 vybral som Spring framework, ktorý poskytuje RESTful služby a ponúka jednoduchú konfiguráciu a manipuláciu s touto architektúrou (REST anotácie, RequestBody a ResponseBody anotácie, podpora formátovania JSON).

4.9.4 Framework perzistencie

Na implementáciu perzistencie medzi databázou a webovou službou som zvolil framework MyBatis. Umožňuje písanie ručných dotazov, čo je z môjho pohľadu veľká výhoda, pretože vývojár presne vie aké dotazy sa aplikujú na databázu. Nie je potrebný žiaden predkompilátor a je možné mať plný prístup ku všetkým funkciám SQL. Neimplementuje žiadne externé rozhrania, takže nevzniká závislosť daného frameworku na webovej službe a je ho možné kedykoľvek vymeniť za iný. Taktiež podporuje vytváranie dynamických SQL dotazov, čo aj v tejto práci využívam.

4.9.5 Bezpečnostný framework

Výber bezpečnostného frameworku sa konkretizoval v predchádzajucej kapitole viz. 3.5

Kapitola 5

Implementácia

Nasledujúca kapitola sa zaoberá implementáciou autentizácie, autorizácie a odolnosti voči útokom vo webovej službe, ktorú som navrhol a spracoval. Každá požiadavka je popísaná v samostatnej sekcii, ktorej názov odpovedá danej požiadavke. Implementáciu rozdeľujem na tri podstatné časti.

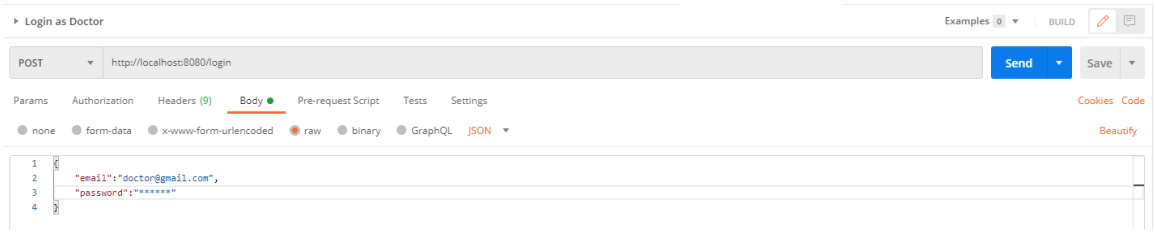
- Autentizácia uživateľov vrátane MFA.
- Kontrola prístupu uživateľov a rolí.
- Odolnosť voči útokom.

Webová služba je nasadená v prostredí Azure na adrese:

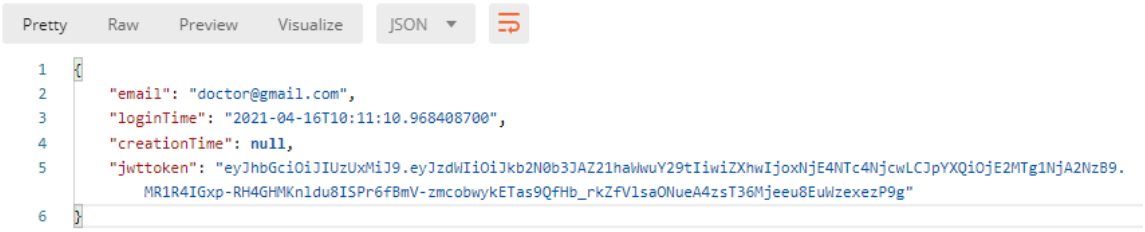
- `https://pandemic-center-bp.azurewebsites.net/`

5.1 Autentizácia

Autentizácia je implementovaná mechanizmom JWT, ktorý sme si popísali v predchadzajúcej kapitole 4.8. Klient sa môže autentizovať zaslaním POST dotazu na webovú adresu `https://pandemic-center-bp.azurewebsites.net/api/v1/login` vo formáte 5.1, kde e-mail a heslo predstavuje prihlasovacie údaje. Ak je overenie úspešné, tak server vygeneruje token, ktorý sa skladá z uživateľských údajov a oprávnení v zahešovanej forme. Následne klient obdrží tento token 5.2, ktorý sa vloží do hlavičky dotazu, čo mu povolí pohyb po webovej službe. Životnosť tokenu sú 2 hodiny. To znamená, že to je čas potrebný do expirácie tohto tokenu, čo spôsobí, že uživateľ už nebude schopný sa overiť pomocou tohto tokenu a bude sa musieť znovu prihlásiť. Problém nastáva vtedy ak sa chce uživateľ odhlásiť skôr ako sa token expiruje. Po odhlásení klientská strana vymaže token z miestneho úložiska alebo súborov cookie, ale token je na serveri stále platný. Ak by útočník získal tento token, tak stále môže byť schopný vykonávať žiadosti v mene používateľa, kým nevyprší jeho platnosť. Jeden zo spôsobov ako tomu zabrániť je vytvoriť blacklist tokenov. Do blacklistu sa ukládajú tokeny, ktoré už nemajú byť platné no ich časová expirácia ešte nevypršala. Na základe toho systém vie, že token je už neplatný a tým pádom má zamietnuť overenie totožnosti a vrátiť error kód 401 (unauthorized).



Obr. 5.1: Dotaz na overenie totožnosti uživateľa



Obr. 5.2: Obdržaný token po úspešnom overení totožnosti užívateľa

5.2 Viacfaktorová autentizácia

Viacfaktorová autentizácia (MFA) je metóda autentifikácie, ktorá vyžaduje, aby používateľ poskytol dva alebo viac overovacích faktorov na získanie prístupu k zdroju, ako je napríklad webová služba. MFA je kľúčovou súčasťou politiky silnej identity a prístupu. Namiesto požadovania používateľského mena a hesla MFA vyžaduje jeden alebo viac ďalších overovacích faktorov, čo znižuje pravdepodobnosť úspešného kybernetického útoku.[2]

5.2.1 Overenie totožnosti pomocou emailu

V dnešnej dobe sa na emaily uchová takmer všetko na jednom mieste: fotografie, zmluvy, faktúry, niekedy dokonca heslá. E-maily sú prepojené so všetkými ostatnými digitálnymi účtami, od bankových účtov po sociálne siete, Cloudové služby atď. Poskytovatelia domény preto musia dbať na ochranu osobných údajov svojich zákazníkov, aby si zachovali svoju popularitu. Emailová adresa je úzko spojená s užívateľom a je veľmi malá pravdepodobnosť, že by došlo k jej odcudzeniu. Preto som použil ako overovací faktor verifikáciu emailu. Vo svojej aplikácií používam dva typy overovania pomocou emailu.

1. Overenie emailovej adresy na existenciu a či je aktívna alebo platná. Platný e-mail je ten, ktorý môže prijímať správy od iných odosielateľov.

2. Zaslanie verifikačného kódu na email pre úspešnú autentizáciu.

Prvý spôsob sa aplikuje po registrácií nového uživateľa do systému. Druhá varianta sa vykoná pri procese overovania totožnosti. Na zasielanie týchto správ som použil rozhranie JavaMail API.

JavaMailApi

JavaMail API je rozhranie, ktoré poskytuje protokolovo nezávislý framework na zasielanie správ. Rozhranie poskytuje sadu abstraktných tried definujúcich objekty, ktoré tvoria poštový systém.

5.2.2 Overenie totožnosti pomocou SMS

Overenie pomocou SMS je bežný spôsob, ako do aplikácií pridať druhú formu overenia. SMS správa sa obvykle skladá z jedinečného kódu. Uživateľ sa musí autentifikovať do systému a následne mu webová služba zašle otp kód na mobilný telefón používateľa. Keď používateľ príjme správu na svoj mobilný telefón, môže zadať kód do prisluchajúcej webovej služby. Uživateľ v tomto procese používa počítač aj mobilný telefón. Ide teda o dvojfaktorový overovací systém.

5.3 OAuth2.0

OAuth je autorizačný protokol, ktorý popisuje, ako nesúvisiace služby môžu bezpečne povoliť autentizovaný prístup bez zdieľania prihlasovacích údajov. Vďaka Spring Security a Spring Boot je implementácia webovej aplikácie pomocou protokolu OAuth 2.0 príjemná a jednoduchá.

Jadrom tejto implementácie je metóda `oauth2Login()`, ktorá sa používa na povolenie podpory prihlásenia Spring Security OAuth 2.0. Následne je potrebné nakonfigurovať súbor `application.properties`.

```
spring:
  security:
    oauth2:
      client:
        registration:
          custom:
            client-id: ssoClient-1
            client-secret: ssoClientSecret-1
            scope: read,write
            authorization-grant-type: authorization\_code
            redirect-uri: http://localhost:8082/ui-one/login/oauth2/code/custom
```

`Spring.security.oauth2.client.registration` je koreňovým menným priestorom na registráciu klienta. Definoval som klienta s registračným ID. Následne som definoval jeho `client-id`, `client-secret`, `scope`, `authorized-grant-type` a `redirect-uri`, ktoré by samozrejme mali byť rovnaké ako tie, ktoré sú definované pre autorizačný server.

Vďaka tejto zmene sa po dotázaní na uri */oauth2/authorization/<provider>*, možeme prihlásiť pomocou zadaného providera. Po overení totožnosti a prijatí všetkych autorizácií, ktoré požiadajú, sa web presmeruje späť na domovskú stránku. Ak prihlásenie do providera ostane aktívne, nemusí sa znova overiť pomocou oauth. To znamená jednotné prihlásenie.

5.4 Autorizácia

Správu uživateľov a rolí som sa rozhodol implementovať vlastným spôsobom. To znamená, že som nepoužil žiaden framework. Urobil som tak z toho dôvodu, pretože pri použití frameworku pri kontrole prístupu vzniká obrovská závislosť na objekte typu „Subject“ (Vo svete spring security to je objekt „Authentication“). Tento objekt drží aktuálny kontext (Session) užívateľa. V tejto práci som sa tomuto objektu snažil obkľukou vyhnúť. Jeho

použitie by znamenalo vytvorenie si nemalej závislosti na danom frameworku a tým pádom aj náročnejšiu potencionálnu výmenu.

Návrh spočíva vo výbere autorizačného modelu a databázového návrhu. Cieľom bolo, aby riadenie prístupu bolo riadené výhradne dátami, nie kódom a to z dôvodu jednoduchosti úpravy rolí, alebo oprávnení.

5.4.1 Autorizačná schéma

Ako autorizačnú schému som použil model RBAC viz. 4.8, ktorú som mierne rozšíril. Kontrolu prístupu rozdeľujem do troch kategórií.

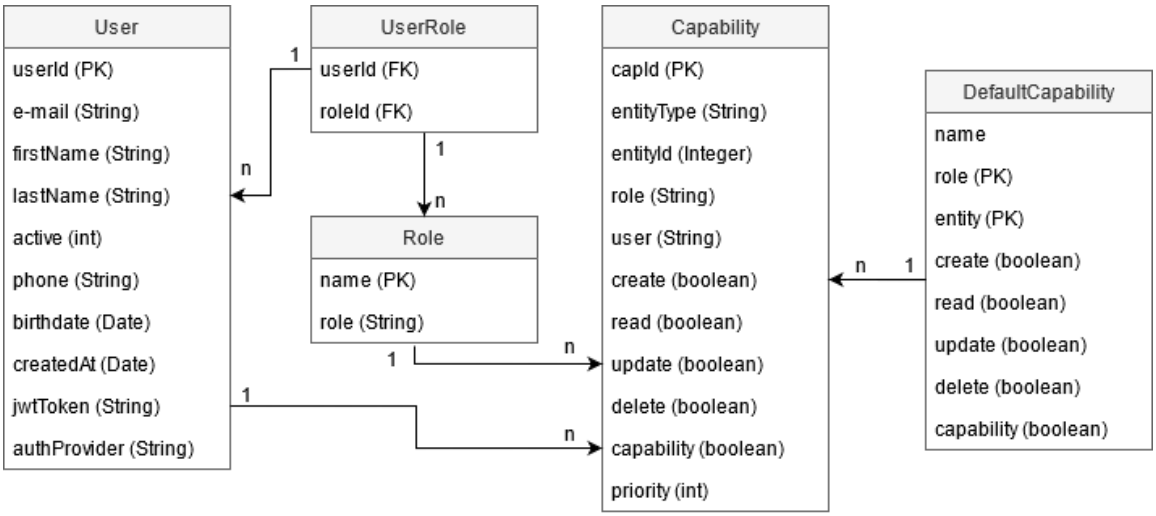
1. Užívateľské oprávnenia
2. Oprávnenia dané skupinou
3. Oprávnenia dané rolou

Oprávnenia majú definovanú hierarchiu:

`Užívateľské oprávnenie > Oprávnenie dané skupinou > Oprávnenie dané rolou`.

To znamená, že ak niektoré z oprávnení povoľuje prístup k danej inštancii a ďalšie ho zakazuje, tak výsledok bude hodnota oprávnenia s najväčšou prioritou.

Databázový model



Obr. 5.3: Dotabázovi model riadenia prístupu uživateľov a rolí

DefaultCapability

Táto entita umožňuje definovať predvolené chovanie jednotlivých rolí v určitej inštancii objektu. Niekedy je potrebné vytvárať oprávnenia automaticky po vykonaní nejakej operácie. Napríklad: *Mám uživateľa U1 s rolou doktor, ktorý nie je účastnikom návštevy N. Podľa*

definovaných prípadov užitia užívateľ U1 nemá žiadne práva k návšteve N, okrem oprávnenia čítať. Vlastník U2 návšetvy N potrebuje pomôcť pri lekárskom zákroku a tým pádom si prizve do návštevy užívateľa U1. Po pridaní užívateľa U1 sa vytvorí oprávnenie (capability) pre užívateľa U1 na základe predvoleného oprávnenia (operácia editovať návštevu) vzťahujúca sa k role doktor. Ak predvolené oprávnenie neexistuje, tak sa vytvorí oprávnenie len s povolením zobraziť, čiže "read".

Príklad predvolenej schopnosti:

Name	role	entity	read	update	create	delete	capability
owner	null	null	true	true	true	true	true
doctor_in_visit	doctor	visit	true	true	false	false	false
sister_in_visit	sister	visit	true	true	false	false	false
user_in_visit	user	visit	true	false	false	false	false

Tabuľka 5.1: Predvolené oprávenia pre role

V skratke, predvolené oprávnenia slúžia na definovanie práv (capability) užívateľa na základe ich rolí v určitej inštancií objektu.

Capability

Záznam v tejto entite definuje oprávnenia užívateľa, role, alebo skupiny v objekte, prípadne inštancií daného objektu.

Príklad schopnosti:

ID	role	user	eType	eID	read	update	create	delete	cap	P
1	doctor	null	visit	null	true	false	true	false	false	1
2	sister_v_5	null	visit	5	true	false	false	false	false	2
3	user	null	visit	null	false	false	false	false	false	1
4	personal	email	visit	1	true	true	true	false	false	3

Role

Atribut role definuje rolu užívateľa v systéme. Ak má tento atribut hodnotu personal, tak sa jedná o oprávnenie pre konkrétneho užívateľa.

User

Atribut user definuje konkrétneho užívateľa v systéme pomocou unikátneho identifikátora. V tomto prípade používam email. Ak je hodnota **null** nejedná sa o konkrétneho užívateľa, ale o rolu alebo skupinu.

eType a eID

Atributy eType a eID udávajú o aký objekt a jeho inštanciu sa jedná. Ak je hodnota stĺpca **null**, tak sa jedná o všetky inštancie danej entity.

Priority

Určuje prioritu danej schopnosti. V systéme je možné, že nastanú situácie, kedy sa určitá operácia vzťahuje ako na užívateľa, tak aj na rolu. V takomto prípade je potrebné rozlišovať oprávnenia na dôležité a menej dôležité. To určuje priorita jednotlivých oprávnení. Užívateľské oprávnenia sa označujú najväčšou prioritou, oprávnenia dané rolou v určitej inštancii objektu strednou a oprávnenia dané rolou vzťahujúce sa na všetky objekty najmenšou prioritou.

Oprávnenia

Všetky ostatné atributy určujú oprávnenie uživateľa, alebo skupiny k prisluchajúcej inštancii objektu

Kontrola prístupu

Kontrola prístupu prebieha na servisnej vrstve, kedy sa pred vykonaním databázovej operácie skontroluje, či má daný uživateľ oprávnenie na vykonanie danej operácie.

Príklad:

```
@Service
public class VisitService {
    @Override
    public Visit findById(final int id, AuthorizeModel authorizeModel) {
      capabilityInteractor.isPermitted(authorizeModel);
      return visitRepository.findById(id);
    }
}
```

Pri dotazovaní sa na určitú entitu otestujem, či užívateľ, ktorý sa dotazuje má oprávnenie spraviť túto operáciu pre danú entitu. Ak mechanizmus vyhodnotí, že dané oprávnenie má, tak je proces autorizácie úspešný. V opačnom prípade server vráti error kód 403 Forbidden.

5.5 Odolnosť voči útokom

V tejto časti predstavím implementáciu odolnosti voči útokom CSRF a XSS. ochranu voči týmto útokom som implementoval za pomoci knižnice Spring Security, ktorá poskytuje túto možnosť zabezpečenia.

5.5.1 Cross-site request forgery

Falšovanie požiadaviek medzi webmi (známe tiež ako CSRF) je chyba zabezpečenia webu, ktorá umožňuje útočníkovi navádzať používateľov na vykonávanie akcií, ktoré nemajú v úmysle urobiť. Umožňuje útočníkovi čiastočne obísť rovnakú politiku pôvodu, ktorá má zabrániť rôznym webovým stránkam vo vzájomnej interferencii.

Spring poskytuje CSRF ochranu predvolene, hneď po integrácií tohto frameworku. Aby sa mohla použiť CSRF ochrana, najskôr je potrebné sa uistiť, že používam správne metódy HTTP pre čokoľvek, čo modifikuje stav (PATCH, POST, PUT a DELETE - **nie GET**). S povolenou ochranou CSRF na strane servera, je nutné zahrnúť token CSRF aj do požiadaviek na strane klienta.

Token

```
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
```

5.5.2 Cross-site scripting

Cross-site scripting (XSS) je jedným z najdôležitejších útokov na bezpečnosť webu. Táto knižnica poskytuje určitú pomoc, ale pre úplnú ochranu som musel implementovať ďalší kód. V tejto práci som použil dostupné metódy Spring Security a pridal som vlastný filter XSS.

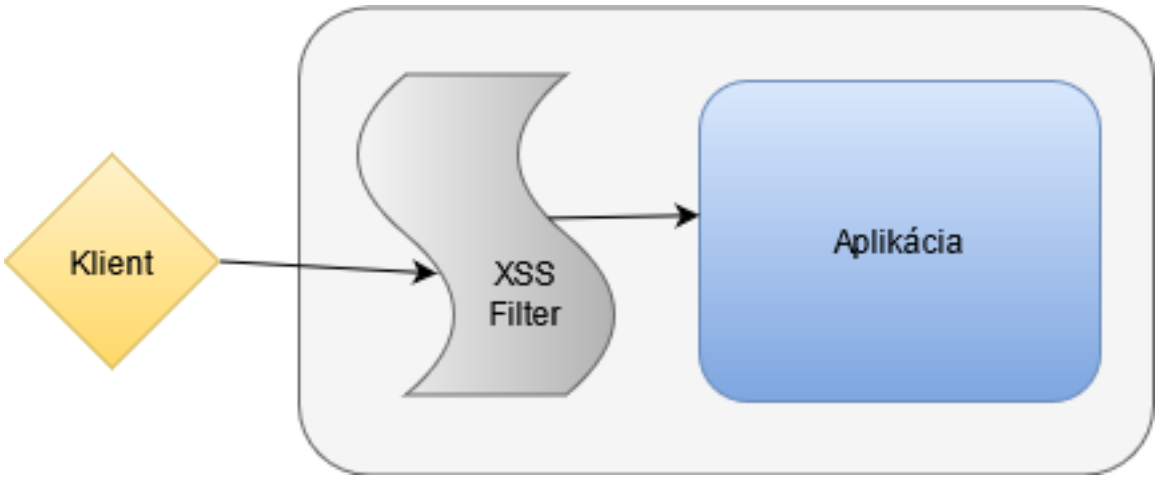
Hlavnou stratégiou na predchádzanie útokom XSS je čistenie vstupov používateľov. Vo webovej aplikácii Spring je vstupom používateľa požiadavka HTTP. Aby sme zabránili útoku, mali by sme skontrolovať obsah požiadavky HTTP a odstrániť všetko, čo by mohlo byť spustiteľné serverom alebo v prehľadávači.

Pre bežnú webovú aplikáciu, ku ktorej je prístup cez webový prehliadač, je možné použiť vstavané funkcie Spring Security (Reflected XSS). Pre webovú aplikáciu, ktorá sprístupňuje API, Spring Security neposkytuje žiadne funkcie a tým pádom som musel implementovať vlastný filter XSS, aby som zabránil útoku XSS.

Spring Security štandardne poskytuje niekoľko bezpečnostných hlavičiek. Obsahuje hlavičku X-XSS-Protection. X-XSS-Protection informuje prehliadač aby blokoval to, čo vyzerá ako XSS.

Spustenie xss ochrany prostredníctvom knižnice Spring Security.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
      .headers()
      .xssProtection()
      .and()
      .contentSecurityPolicy("script-src 'self'");
}
```



Obr. 5.4: Schéma prístupu k aplikácií prostredníctvom bezpečnostného XSS filtra.

Filter musí z tela aj hlavičky dotazu odstrániť nebezpečný obsah.

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
  throws IOException, ServletException {
    XSSRequestWrapper wrappedRequest = new XSSRequestWrapper((HttpServletRequest)
        request);
    String body = IOUtils.toString(wrappedRequest.getReader());
    if (!StringUtils.isBlank(body)) {
        body = XSSUtils.stripXSS(body);
        wrappedRequest.resetInputStream(body.getBytes());
    }
    chain.doFilter(wrappedRequest, response);
}
```

Z dotazu sa načíta kontent, ktorý následne prejde XSS filtrom. Filter detekuje škodlivý kód a odstráni ho zo vstupu. Nakoniec sa vstup vloží do dotazu a pokračuje sa v ďalšom spracovávaní tohto dotazu.

Kapitola 6

Testovanie

Táto kapitola sa venuje testovaniu a zhodnoteniu dostiahnutých výsledkov. V tejto práci som spravil viacero testov, kde overujem rôzne časti aplikácie, hlavne zabezpečenia. Predstavím základnú funkčnosť integračných testov, sadu dotazov s príkladmi v nástroji Postman. Následne zhodnotím testy na ochranu proti CSRF a XSS útokom.

6.1 Integračné testy

Testy som implementoval pomocou knižnice JUnit. Vykonanie integračného testovania ponúka veľa výhod. Zaisťuje správne fungovanie integrovaných modulov. Zisťuje chyby súvisiace s rozhraním medzi modulmi. Pomáha modulom komunikovať s API a ďalšími nástrojmi tretích strán. Spravidla pokrýva veľký objem systému, takže je efektívnejší. Zvyšuje pokrytie testu a zvyšuje spoľahlivosť testov. Pomocou integračných testov overujem funkčnosť repozitára, servisnú vrstvu a funkčnosť jednotlivých oprávnení nad konkrétnymi entitami.

6.2 Testovanie pomocou postman (sada dotazov)

Nástrojom postman som vytvoril sadu dotazov. Súčasťou tejto sady je súprava dotazov rôzne kombinovaných prípadov užitia a súprava modulárnych častí api, kedy si testujúci môže nadefinovať vlastné parametre dotazu.

6.3 Autorizačný model

V rámci autorizácie a autorizačného modelu som testoval funkčnosti dvoch kategórií.

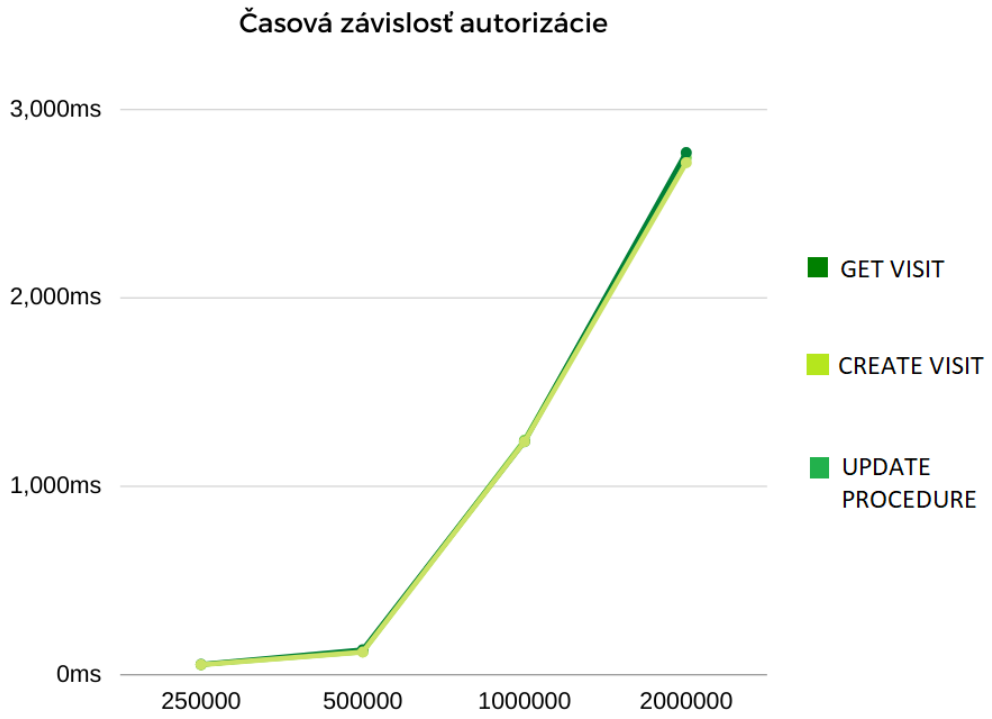
- Deterministickosť (software stále vie ako sa má v prípade autorizácie zachovať).
- Časová škálovateľnosť.

Testovanie funkčnosti prebiehalo, zasielaním rôznych kombinácií dotazov (reprezentovaním chovania) pomocou nástroja Postman. Cieľom bolo aby navrhnutá schéma mala deterministický pohľad na kontrolu prístupu.

Ďalším faktorom bola časová škálovateľnosť. Urobil som niekoľko krokové testovanie, kde som sa zameral na manipuláciu s objektami s autorizáciou a následne bez autorizácie. Testovanie prebiehalo naplnením skušobných dát do databázy v rôznych počtoch. Pokračovalo

to dotazovaním sa na rôzne objekty s rôznymi typmi operácií. Vyzbieraním dostatočného počtu informácií som výsledky spriemeroval a vložil do grafu.

Z grafu 6.1 je vidno, že čas potrebný na autorizáciu pri počte 1 000 000 oprávnení, je približne 1,3s a pri počte 2 000 000 oprávnení je to približne 2,7s. Pre porovnanie som vytvoril ďalší graf, ktorý určuje čas potrebný na vykonanie danej operácie bez použitia autorizácie. V grafe 6.2 vidíme, že pri 1 000 000 inštanciách objektu, transakcia priemerne trvá 1,8ms no pri 2 000 000 to už je takmer 3ms. Z toho vyplýva, že proces autorizácie je v porovnaní s manipuláciou inštanciami objektu pomerne časovo náročný, no škálovateľný a jednoducho rozšíriteľný.



Obr. 6.1: Graf časovej zložitosti vo vzťahu s počtom oprávnení v databáze s použitím implementovanej autorizácie

Obr. 6.2: Graf časovej zložitosti vo vzťahu s počtom oprávnení v databáze bez použitia implementovanej autorizácie

6.4 Ochrana proti CSRF

CSRF ochranu som testoval napodobením útoku. Vytvoril totožnú stránku, ktorá obsahovala útočníkové data. Následne som sa snažil prepojiť touto stránkou medzi webovými službami a spôsobiť tak nejakú škodu. Výsledkom bolo, že aplikácia detekovala útočnikovu stránku tým, že dostala token, ktorému nedôverovala a tak prístup odoprela.

6.5 Ochrana proti XSS

Ochranu proti XSS útokom som testoval dotazovaním sa na rôzne prístupové body api prostredníctvom skriptov vložených v tele dotazu.

Dotaz

```
{
    "email": "test2@gmail.com",
    "identification": "970418/6361",
    "city": "Bardejov",
    "address": "Hurbanova 50<script>alert('xss')</script>",
    "firstName": "Maros <b onmouseover=alert('XSS')></b>",
    "lastName": "Geffert",
    "phone": "+420111222333"
}
```

Obr. 6.3: Príklad XSS útoku, volženie skriptu do tela dotazu.

Odpoveď

```
{
  "visitId" : 100035,
  "identification" : 9704186361,
  "firstName" : "Maros ",
  "lastName" : "Geffert",
  "reason" : null,
  "symptoms" : null,
  "email" : "test2@gmail.com",
  "phone" : "+420111222333",
  "city" : "Bardejov",
  "address" : "Hurbanova 50",
  "birthdate" : null,
  "visitTime" : null,
  "patientId" : 99970,
  "userId" : 0
```

Obr. 6.4: Príklad XSS útoku, čistá odpoveď po detekovaní a odstránení skriptu z tela dotazu.

Vidíme, že po vložení skriptu do tela dotazu systém detekuje potencionálne nebezpečenstvo. Odstráni skript z daného poľa a pošle čistú odpoveď ďalej na spracovanie.

Kapitola 7

Záver

V tejto práci bola spracovaná problematika autentizácie, autorizácie, MFA, správa rolí, užívateľov a skupín. Porovnal som možnosti danej problematiky od klasických prístupov až po moderné architektonické návrhy.

Komunikácia medzi časťami aplikácie prebieha cez protokol http pomocou štandardu REST. Overenie totožnosti a kontrola prístupu je implementovaná tokenom JWT, ktorý umožní odľahčiť z nutnosti udržiavať stav užívateľa na serveri a celý proces overenia totožnosti tak zrýchliť a optimalizovať. K implementácií OAuth som použil knižncu Spring Security, ktorá jednoducho a príjemne umožňuje nakonfigurovať službu oAuth. Ďalej som implementoval verifikáciu užívateľa pomocou otp kódu, ktorý si uživateľ môže dať voliteľne zaslať na email, alebo mobil.

Aplikácia je odolná voči CSRF a XSS útokom. Ochranu voči týmto útokom som implementoval taktiež pomocou vybranej bezpečnostnej knižnice Spring Security.

Kontrolu prístupu som vytvoril na základe rolí (RBAC), ktorú som následne modifikoval podľa potreby. Vytvoril som riešenie, ktoré dokáže pohodlne riadiť prístup na úrovni inštancií objektov s akýmkoľvek počtom rolí alebo užívateľských oprávnení. Riešenie som navrhol tak, aby bolo jednoducho rozšíriteľné. Viacnásobným testovaním som zisťoval časovú zložitosť algoritmu za účelom zistenia škálovateľnosti kontroly prístupu.

7.1 Možné vylepšenia

Aplikáciu som sa snažil navrhnúť a modelovať tak aby bola jednoducho rozšíriteľná. Potencionál pre vylepšenia je veľký. Jedná sa napríklad o:

- Zmeniť CRUD atributy entity „capability“ na jednoduchý integer, kde oprávenia daných uživateľov by určovala hodnota bitu na určitom mieste. Toto riešenie, optimalizuje prístup k databáze. Zmenšil by sa objem tabuľky o niekoľko stĺpcov a tým, by sa zrýchlil celkový proces autorizácie.
- Vytvorenie špeciálnych databázových dotazov, ktoré by presne rozhodovali aké oprávnenie sa má z databázy vybrať. Aktuálne je problém riešený programovo, kedy z viacerých možných oprávnení algoritmus vyberie ten správny.

Literatúra

[1] Apache team. *Apache Shiro* [online]. 2011 [cit. 2021-11-04]. Dostupné z: https://shiro.apache.org.

[2] Dasgupta, D., Roya, A. a Nag, A. *Advances in User Authentication*. 1. vyd. Springer, Cham, 2017. 197-204 s. ISBN 978-3-319-58808-7.

[3] Farah, T., Shojol, M., Hassan, M. M. a Alam, D. *International Conference on Digital Information and Communication Technology and it's Applications*. IEEE, 2016, s. 74–78. ISBN 978-1-4673-9609-7.

[4] GoodFirms. *What is a Web Framework?* [online]. 2021 [cit. 2021-04-05]. Dostupné z: https://www.goodfirms.co/glossary/web-framework/.

[5] Hazlewood, L. *Apache Shiro Architecture* [online]. 2011 [cit. 2021-09-04]. Dostupné z: https://shiro.apache.org/architecture.html.

[6] Hazlewood, L. *Application Security With Apache Shiro* [online]. 2011 [cit. 2021-09-04]. Dostupné z: https://www.infoq.com/articles/apache-shiro/.

[7] IT Pro team. [online]. 2018 [cit. 2021-08-04]. Dostupné z: https://www.itpro.co.uk/strategy/29643/what-is-an-application-server.

[8] java T point. *Spring Security features* [online]. [cit. 2021-09-04]. Dostupné z: https://www.javatpoint.com/spring-security-features.

[9] Lai, C., Gong, L., Koved, L., Nadalin, A. a Schemers, R. *User Authentication and Authorization in the JavaTMPlatform*. 1. vyd. IEEE, 1999, s. 285–290. ISBN 0-7695-0346-2.

[10] Lilly021. *Spring security architecture* [online]. 2021 [cit. 2021-11-04]. Dostupné z: https://lilly021.com/spring-security-architecture/.

[11] Long, M. Creating a distributed field robot architecture for multiple robots. [online]. Máj 2021. Dostupné z: https://www.researchgate.net/figure/Overview-of-Java-Authentication-and-Authorization-Service_fig4_249982505.

[12] Marsh, K. [online]. 2019 [cit. 2021-12-04]. Dostupné z: https://kb.globalscape.com/Knowledgebase/10691/What-is-the-difference-between-basic-auth-and-formbased-auth.

[13] Martinelli, S. *Java Persistence Done Right* [online]. 2019 [cit. 2021-04-05]. Dostupné z: https://dzone.com/articles/java-persistence-done-right.

[14] MDN Web Docs. *HTTP authentication* [online]. [cit. 2021-20-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication.

[15] Nascimento, A. E. *OAuth 2.0 Cookbook*. 1. vyd. Pack Publishing Ltd., 2017. ISBN 978-1-78829-596-3.

[16] Oaks, S. *Java Security*. 2. vyd. O'Reilly, 2001. 86-90 s. ISBN ISNB: 0-596-00157-6.

[17] Oracle team. *Java Authentication and Authorization Service (JAAS) Reference Guide* [online]. [cit. 2021-13-04]. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[18] Roseindia. *What is Persistence Framework?* [online]. 2018 [cit. 2021-04-05]. Dostupné z: https://www.roseindia.net/enterprise/persistenceframework.shtml.

[19] Spring team. *Spring Security* [online]. [cit. 2021-09-04]. Dostupné z: https://spring.io/projects/spring-security#learn.

[20] Spring team. *Spring Framework Reference Documentation*. Addison-Wesley UK, 2004. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.19.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf.

[21] Srivastava, P. *Java Authentication and Authorization Service (JAAS)* [online]. [cit. 2021-13-04]. Dostupné z: https://www.javastudypoint.com/2018/12/java-authentication-and-authorization-service-jaas.html.

[22] Surwase, V. REST API modeling languages-a developer's perspective. *Int. J. Sci. Technol. Eng.* 2016, zv. 2, č. 10, s. 634–637.

[23] TechTarget. *Digest authentication* [online]. 2007 [cit. 2021-12-04]. Dostupné z: https://searchsecurity.techtarget.com/definition/digest-authentication.

[24] W3spoint team. *Spring security architecture diagram* [online]. [cit. 2021-09-04]. Dostupné z: https://www.w3spoint.com/spring-security-architecture-diagram.

Príloha A

Súčasťou technickej správy je aj CD ktoré obsahuje:

- Sada dotazov pre nástroj Postman
- Zdrojové kódy
- Elektronická verzia technickej správy
- Súbor `README.md`, kde je návod na spustenie aplikácie a všeobecné informácie