

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

VIAC-FAKTOROVÁ AUTENTIFIKÁCIA

**Samuel Adler, Juraj Rak, Jakub Rosina,
Martin Pač, Dávid Zabák, Filip Kadúch**

2022

Názov práce: Viacfaktorové overenie pomocou autentifikačného a autorizačného servera

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je implementovať autentifikačný a autorizačný server. Systém bude implementovaný tak, aby ho bolo možné nasadiť ako modul do webovej aplikácie – napríklad e-shopu, ktorý potrebuje autentifikovať a autorizovať používateľov.

Úlohy:

1. Pripravte architektúru pre autentifikačný a autorizačný server s transaction risk management systémom.
2. Architektúru pripravte podľa dostupnej literatúry. Vypracujte schémy v archimate jazyku použitím Archi opensource nástroja.
3. Implementujte dvojfaktorovú autentifikáciu tak, aby ju bolo možné vykonať cez počítač a mobilné zariadenie.
4. Navrhnite a vypracujte vhodné prípady použitia pre enrolment, login, access authorisation, change authorisation, transaction authorisation a logout
5. Oboznámte sa s použiteľným opensource serverom na autentifikáciu alebo vhodnou aplikáciou či knižnicou, podľa ktorej bude možná implementácia prípadov použitia
6. Analyzujte a implementujte openCV knižnicu pre rozpoznávanie tvári. Je použiteľná aj iná knižnica? Ak áno, porovnajte.
7. Implementujte risk evaluation server. Existuje nejaký opensource nástroj na vyhodnotenie rizika? Pri vyhodnotení pracujte s informáciami ako IP adresa, OS, krajina, verzia systému, prehliadač a jeho verzia.
8. Navrhnite a implementujte vhodný spôsob dvojfaktorovej autentifikácie s využitím mobilnej aplikácie Google Authenticator alebo Microsoft Authenticator. Analyzujte a porovnajte použitie OTP alebo push notifikácie.
9. Oboznámte sa s certifikačnými autoritami. Analyzujte použitie MS Active Directory alebo EJBCA. Vyberte vhodnejší spôsob.
10. Vyhodnoťte výsledky implementácie z funkčnej a používateľskej stránky.

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autori:	Samuel Adler, Jakub Rosina, Martin Pač, Dávid Zabák, Juraj Rak, Filip Kadúch
Téma:	Viac-faktorová autentifikácia
Vedúci záverečnej práce:	Ing. Marek Repka PhD.
Miesto a rok predloženia práce:	Bratislava 2022

Téma práce je založená na systéme viac-faktorovej autentifikácie. Systém je vyvíjaný ako webová aplikácia, pri ktorej je potrebné dbať na viacero faktorov. Hlavným cieľom je vyvinúť komplexný systém, ktorý bude slúžiť ako autentifikačný a autorizačný server, kde risk evaluation server bude vyhodnocovať riziko pri prihlásení potenciálneho používateľa na základe viacerých informácií, ako sú napríklad

- IP adresa
- prehliadač a jeho verzia
- operačný systém a jeho verzia
- krajina

Jedným z ďalších cieľov práce je implementovať funkcionality pri registrácii a prihlásení tak, aby sa daný používateľ vedel autentifikovať cez konkrétnu aplikáciu, ako napríklad Google Authenticator alebo Microsoft Authenticator, čiže buď pomocou jednorazového kódu, alebo pomocou push notifikácie, ktorú dostane pri snahe prihlásiť sa. Implementáciu vykonáme ďalej tak, že jednotlivým používateľom budeme priradzovať role, na začiatku každý dostane obvyčajnú user rolu. Po ukončení implementácie bude musieť celá funkcionality prejsť testovaním či spĺňa všetky podmienky na úspešné zahájenie produkcie.

Kľúčové slová: spring boot, autentifikácia, autorizácia, risk server, rozpoznávanie tváre

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Authors:	Samuel Adler, Jakub Rosina, Martin Pač, Dávid Zabák, Juraj Rak, Filip Kadúch
Topic:	Multi-factor authentication
Supervisor:	Ing. Marek Repka PhD.
Place and year of submission:	Bratislava 2022

The topic of the work is based on a multi-factor authentication system. The system is developed as a web application, in which it is necessary to pay attention to several factors. The main goal is to develop a comprehensive system that will serve as an authentication and authorization server, where the risk evaluation server will evaluate the risk when a potential user logs in based on more information such as

- IP address
- browser and its version
- operating system and its version
- country

One of the other goals of the work is to implement the registration and login functionality so that the user can authenticate through a specific application such as Google Authenticator or Microsoft Authenticator, ie using either a one-time code or push notification that he receives when trying to log in. We will further implement the implementation by assigning roles to individual users, at the beginning each will receive an ordinary user role. After the implementation is completed, the entire functionality will have to be tested to see if it meets all the conditions for a successful start of production.

Keywords: spring boot, authentication, authorization, risk server, face recognition

Podakovanie

Chceme sa poďakovať vedúcemu tímového projektu, ktorým bol Ing. Marek Repka PhD., za odborné vedenie, rady a pripomienky, ktoré nám pomohli pri vypracovaní tejto práce.

Obsah

1	OAuth	7
1.1	História	7
1.2	Bezpečnostné problémy	7
1.2.1	OAuth 1.0	7
1.2.2	OAuth 2.0	7
1.3	OAuth 2.0	7
1.4	Použitie	8
2	Oath, Oath ocra	9
2.1	Oath	9
2.2	Oath Ocra	9
3	SAML	11
3.1	Použitie	11
3.2	SAML provider	11
3.3	Funkcionalita	11
3.4	Výhody	12
3.5	Nevýhody	12
4	MQTT a bezpečnosť	13
4.1	Základné vlastnosti MQTT	13
4.2	MQTT základné typy správ komunikácie s brokerom	13
4.3	Sémantika tém	13
4.4	Štruktúra správ	14
4.5	Bezpečnosť MQTT	14
4.6	Implementácia	14
4.6.1	Connect	15
4.6.2	Login/Password connection	16
4.6.3	Publish	17
4.6.4	Subscribe	17
4.6.5	Disconnect	18
5	JMS, AMQP a MQTT	19
5.1	Posielanie správ pre účely autorizácie	19
5.2	Java Messaging Service	20
5.3	Advanced Message Queueing Protocol	21
5.4	Využitie ActiveMQ pre účely autorizácie	22
5.5	Návrh riešenie prostredníctvom ActiveMQ brokera	22

5.6	Architektúra ActiveMQ Brokera	23
5.6.1	Queue vs topic	23
6	API security	24
6.1	Slabá alebo žiadna autentifikácia	24
6.2	Šifrovanie citlivých údajov	24
6.3	Posielanie viac dát ako je potrebné	24
6.4	Zabudnuté endpointy alebo shadow API	25
6.5	Podozrivé používanie API	25
7	OpenCV - face recognition	26
8	Metódy pre rozpoznávanie tváre	27
8.1	F-Score	28
8.2	Použité techniky	28
8.3	FaceNet	30
9	Návrh riešenia	31
9.1	Architektúra	31
9.1.1	Autentifikačná platforma	31
9.1.2	Klientske rozhranie	31
9.1.3	Risk manažment	33
9.1.4	Dvojfaktorová autentifikácia	33
9.1.5	Prihlásenie	34
9.1.6	Registrácia	35
9.1.7	Autorizácia	36
9.1.8	Odhlásenie	37
9.1.9	Synchrónne servisy	38
9.1.10	Asynchrónne servisy	39
9.2	Server	40
9.2.1	CAS	40
9.2.2	Keycloak	40
9.2.3	Java Spring Boot	41
9.2.4	Výber serveru	42
9.3	Certifikačné authority	42
9.3.1	Microsoft Active Directory	42
9.3.2	EJBCA	44
9.3.3	Výber najlepšej authority	45
10	Implementácia	46

10.1	Výsledná architektúra	46
10.1.1	Diagram pre <i>High-level</i> architektúru	46
10.1.2	Diagram klientského rozhrania	47
10.1.3	Diagram pre manažment rizík	48
10.1.4	<i>Diagram pre device enrolment</i>	49
10.1.5	Diagram autorizácie	50
10.1.6	Diagram prihlásenia	51
10.1.7	Diagram registrácie	52
10.2	Viac-faktorová autentifikácia	53
10.2.1	Authy aplikácia	53
10.2.2	Authy One Time Password	53
10.2.3	Authy push notifikácie	56
10.3	Registrácia	59
10.3.1	Implementácia	61
10.3.2	Vytvorenie tokenu	61
10.3.3	Overenie tokenu	61
10.3.4	Jednorazová URL	61
10.3.5	Verifikácia vstupov	62
10.4	Prihlásenie	63
10.4.1	Postup implementácie	64
10.4.2	Druhy prihlásenia	64
10.5	Face recognition	64
10.5.1	Frontend stack	64
10.5.2	Frontend metódy pre registráciu a autentifikáciu	64
10.5.3	Backend stack	65
10.5.4	Backend metódy pre registráciu a autentifikáciu	65
10.6	Systémové roly	66
10.6.1	Vytvorenie používateľských rolí	66
10.7	Bezpečnostná konfigurácia	66
10.7.1	Risk server	67
11	Prínos	68
	Záver	69
	Zoznam použitej literatúry	70

Zoznam obrázkov a tabuliek

Obrázok 1	Pridanie dependency	15
Obrázok 2	Deklarovanie služby	15
Obrázok 3	Povolenia	15
Obrázok 4	Connection setup	16
Obrázok 5	Login/Password	16
Obrázok 6	Publish	17
Obrázok 7	Subscribe	17
Obrázok 8	Disconnect	18
Obrázok 9	Java Messaging Service	20
Obrázok 10	AMQP protokol	21
Obrázok 11	zdroj: vlastné spracovanie	22
Obrázok 12	Architektúra ActiveMQ Brokera	23
Obrázok 13	Porovnanie konvolučných techník rozpoznania tváre	27
Obrázok 14	Porovnanie konvolučných techník rozpoznania tváre cez F-Score	28
Obrázok 15	Čas trénovania	29
Obrázok 16	Čas predikovania	29
Obrázok 17	High level architektúra	31
Obrázok 18	Klientske rozhranie	32
Obrázok 19	Risk manažment	33
Obrázok 20	Proces prihlásenia	34
Obrázok 21	Proces registrácie	35
Obrázok 22	Proces autorizácie	36
Obrázok 23	Proces odhlásenia	37
Obrázok 24	Synchrónny proces	38
Obrázok 25	Asynchrónny proces	39
Obrázok 26	Finálny vzhľad <i>high-level</i> architektúry	46
Obrázok 27	Finálny vzhľad klientského rozhrania	47
Obrázok 28	Finálny vzhľad fungovania manažmentu rizík	48
Obrázok 29	<i>Device enrolment</i> počas registrácie	49
Obrázok 30	Finálny vzhľad <i>high-level</i> procesu autorizácie	50
Obrázok 31	Finálny vzhľad <i>high-level</i> procesu prihlásenia	51
Obrázok 32	Finálny vzhľad <i>high-level</i> procesu registrácie	52
Obrázok 33	Parametre URL	54
Obrázok 34	Možné odozvy po poslaní požiadavky	55

Obrázok 35	Dodatočné parametre pre URL	57
Obrázok 36	Dodatočné parametre pre URL	58
Obrázok 37	Prvý krok pri registrácii nového užívateľa	59
Obrázok 38	Jednorazová URL odoslaná na email registrujúceho sa užívateľa	59
Obrázok 39	Registračný formulár	60
Obrázok 40	Prihlasovanie užívateľa	63
Obrázok 41	Úspech prihlásenia	63

Zápisnica stretnutí

Dátum stretnutia - Čo sa konzultovalo a dohodlo

15.10.2021

- Porovnať implementáciu a náročnosť CAS, Keycloak a Spring serverov

22.10.2021

- Pozrieť si ako sa najlepšie dá implementovať SAML, Oauth, Oath a Oath oca, MQTT, OpenCV

12.11.2021

- Implementovať landing page na zadanie emailovej adresy. Jednorazová URL linka pri registrácii s možnosťou nastaviť platnosť URL. Spraviť zvlášť tabuľku v databáze pre registrovaných, kde budeme rozlišovať, či sa registrácia schváli alebo nie. Spojazdniť čo najskôr školský server. Prioritou je implementácia a funkčný prototyp, až potom písanie textu.

19.11.2021

- Lepšie implementovať session management z pohľadu security. Pripraviť vlastný blacklist na IP adresy, krajiny, verziu prehliadača atď. Treba to vedieť poslať serveru na risk analýzu. V Archimate treba spraviť synchrónne a asynchrónne servisy. Analyzovať MQ messaging, MQTT vs. JMS a vybrať si do čoho sa pustíme.

26.11.2021

- Pozrieť si Android Cloud Messaging a push notifikácie. Aký je thin a fat client ? MQTT postaviť neskôr medzi risk a autorizačným serverom, implementovať asynchrónne. V tabuľke pre registrácie budeme potrebovať informácie o používateľovi, jeho mobilnom zariadení. Implementovať Google Authenticator. Databáza by mala zatiaľ mať tabuľky users, devices, sessions, tokens atď.

3.12.2021

- Najprv spraviť Android appku na autentifikáciu, neskôr možno na iOS. Ak sa stihne, tak pozrieť SOAP UI a čo je to mocking. Na koniec januára mať pripravený základný scenár s dvojfaktorovou autentifikáciou.

10.12.2021

- Ak sa podarí tak do januára implementovať aj vyhodnocovanie informácií ako sú prehliadač, jeho verzia, OS, IP adresa, odtlačok zariadenia. Normalizovať strings, napríklad orezať. Dať väčšiu prioritu Firebase.

28.12.2021

- Dokončiť implementáciu Firebase do konca týždňa. Pri registrácii vyžadovať OTP. Prepojiť Facenet s API. Napojiť risk server pre získanie client informácií - browser a verzia, OS, IP, device fingerprint.

7.1.2022

- Ešte viac informácií zo session vytiahnuť, resp. čo najviac. Pokúsiť sa integrovať Microsoft Authenticator a jeho push notifikácie. Vyladiť demo a potom podrobne ukázať databázu atď. Umiestniť na web všetko podľa podmienok TP1.

14.1.2022

- Doladiť demo a fixnúť nájdené bugy. Každý člen tímu napísať 2 krátke odstavce a rozdeliť ich do dvoch častí - čo sa naučil/dosiahol a taktiež, kde vidí priestor na zlepšenie. Dokončiť web a informácie o tíme podľa podmienok TP1 a poslať všetko mailom vedúcemu.

15.2.2022

- Naštudovať si problematiku implementácie Microsoft Authenticatoru. Otestovať Firebase a Retrofit pre účely tvorby mobilnej aplikácie. Ďalšou úlohou je zlepšiť úspešnosť pre Face Recognition

25.3.2022

- Vyriešiť problém s tokenom po vymazaní cache pamäte. Prepájanie userovho zariadenia so serverom a udržanie cache pamäte. Z začať riešiť EJBCA vs MS Active Directory certifikačné authority.

26.4.2022

- Nebudeme nakoniec implementovať vlastnú mobilnú aplikáciu pre push notifikácie. Vyriešiť posielanie notifikácií cez AuthyNotificationController. Zabezpečiť registráciu cez Authy. Tvorba jednorazovej URL adresy pre dočasného usera pri prvej registrácii. Pre risk server treba spraviť vyhodnotenie values po úspešnom prihlásení usera.

6.5.2022

- Upraviť verifikácie vstupov od užívateľa pri zadávaní informácií pri registrácii a prihlasovaní. Napojiť Face Recognition na server. Doriеšiť finalizáciu risk serveru.

Zaznamenávanie zmien

Šprint - Čo sa spravilo

24.10.2021 - 31.10.2021

- Rozbehanie CAS serveru, Poslanie žiadosti o pridelenie serveru, Archimate schémy, Teória - vhodná implementácia Oath, Oath Ocra, Rozbehanie Keycloak, Rozbehanie Spring boot, Teória - vhodná implementácia MQTT, Príkladná implementácia OpenCV

1.11.2021 - 7.11.2021

- Žiadny progress

8.11.2021 - 14.11.2021

- Login a Odhlásenie Spring Boot, Dummy Web appka, Registrácia Spring Boot, Login a registrácia v Archimate

14.11.2021 - 21.11.2021

- Mail landing page, Tabuľky cez JPA, Archimate business logika, Dočasná URL, PCA metóda, Vhodná implementácia a platforma pre OpenCV

22.11.2021 - 28.11.2021

- Archimate autorizácia a logout, Získanie miesta na serveri, Analytická časť práce, Architektúra pre server

29.11.2021 - 5.12.2021

- Archimate synchrónne a asynchrónne servisy, Session management z pohľadu security, MQTT architektúra, Vyhodnocovanie rizika pri logine

6.12.2021 - 12.12.2021

- Android základná funkcionálna+ dizajn, Retrofit kostra pre Android app, Nasadenie spring na server

13.12.2021 - 19.12.2021

- Stránka o nás, Implementácia Google Auth - login + registrácia. Ošetriť nedostupnosť Risk servera.

27.12.2021 - 16.1.2022

- Android app - push notifikácie, Spísanie dokumentu za celý semester, Google Auth - check OTP pred ukončením registrácie, Korektné zasielanie push notifikácii konkrétnym používateľom, Doladanie registrácie a Google Auth. Prepojenie Androidu so serverom cez Firebase.

16.1.2022 - 13.3.2022

- Implementácia MS Authenticatora

14.3.2022 - 27.3.2022

- Autorizácia prístupu užívateľov. Integrácia aplikácie Authy. Úprava OTP pre aplikáciu Authy. Push notifikácie cez Authy.

28.3.2022 - 15.4.2022

- Finalizácia a úprava menších bugov. Verifikácia vstupov od užívateľa.

Úvod

Téma práce je založená na systéme viac-faktorovej autentifikácie. Systém je vyvíjaný ako webová aplikácia, pri ktorej je potrebné dbať na viacero faktorov. Hlavným cieľom je vyvinúť komplexný systém, ktorý bude slúžiť ako autentifikačný a autorizačný server, kde risk evaluation server bude vyhodnocovať riziko pri prihlásení potenciálneho používateľa na základe viacerých informácií, ako sú napríklad

- IP adresa
- prehliadač a jeho verzia
- operačný systém a jeho verzia
- krajina

Jedným z ďalších cieľov práce je implementovať funkcionality pri registrácii a prihlásení tak, aby sa daný používateľ vedel autentifikovať cez konkrétnu aplikáciu, ako napríklad Google Authenticator alebo Microsoft Authenticator, čiže buď pomocou jednorazového kódu, alebo pomocou push notifikácie, ktorú dostane pri snahe prihlásiť sa. Implementáciu vykonáme ďalej tak, že jednotlivým používateľom budeme priradzovať role, na začiatku každý dostane obyčajnú user rolu. Po ukončení implementácie bude musieť celá funkcionality prejsť testovaním či spĺňa všetky podmienky na úspešné zahájenie produkcie.

1 OAuth

OAuth je otvorený štandardný autorizačný protokol alebo rámec, ktorý umožňuje možnosť zabezpečeného prístupu. Pomocou tohto prístupu je užívateľovi umožnené prístup k službám pomocou prihlásenia sa cez iné služby ako je napríklad Facebook alebo Google, bez toho aby užívateľ odovzdal heslo používanej služby. V prípade, že príde k úniku na používanej službe heslo ostáva v bezpečí na väčších službách, napríklad na Facebook-u.

OAuth je špeciálne navrhnutý pre prácu s HTTP (Hypertext Transfer Protokol). Umožňuje vydávať prístupové tokeny klientom tretích strán s jasne daným súhlasom na strane používateľa pomocou autorizačného serveru [1, 2].

1.1 História

Vývoj protokolu OAuth sa začal v roku 2006, kedy Blaine Cook vyvíjal implementáciu Twitter OpenID. Počas tohto vývoja spolu s ďalšími prišli k záveru, že neexistujú žiadne otvorené protokoly pre delegovanie prístupu. Na základe toho v roku 2007 vznikla diskusná skupina, ktorá mala za úlohu vypracovať návrh na zrealizovanie implementácie otvoreného protokolu. 4. decembra 2007 bol skupinou vydaný konečný návrh na vypracovanie protokolu OAuth Code 1.0. Neskôr na základe požiadaviek na vyššiu bezpečnosť a hlavne rýchlosť bol implementovaný protokol OAuth 2.0, ktorý je momentálne používaný [3].

1.2 Bezpečnostné problémy

1.2.1 OAuth 1.0

v prípade prvej verzie bola bezpečnostná chyba zabezpečenia fixácie relácie oznámená v roku 2009. Na základe toho bola neskôr vydaná bezpečnostná oprava a verzia dostala názov 1.0a.

1.2.2 OAuth 2.0

V roku 2013 sa pracovná skupina pre internetové inžinierstvo rozhodla uverejniť model hrozby pre OAuth 2.0. Jednou z popísaných hrozieb je hrozba s názvom "Covert Redirect". Ďalšou hrozbou je súvisí s viacerými autorizačnými servermi. V takomto prípade užívateľ môže byť zmätený a svoje citlivé údaje môže posielat na server, ktorý sa správa škodlivo.

1.3 OAuth 2.0

Verzia 2.0 je kompletný redizajn verzie OAuth 1.0. Tieto dve verzie nie sú vôbec kompatibilné a v prípade novšie vytvorených aplikácií sa už používa iba verzia 2.0. OAuth 2.0 je oproti prvej verzii výrazne rýchlejší keďže podporuje až šesť tokov pre rôzne typy aplikácií a požiadaviek. Tiež podporuje aj prenos pomocou zašifrovaného a novšieho pro-

tokolu HTTPS. To znamená že toky nemusia byť šifrované na koncových bodoch, keďže sú zašifrované už počas prenosu [4].

1.4 Použitie

Dnes je protokol OAuth využívaný všetkými veľkými službami. Facebook pre svoju GraphAPI umožňuje prístup iba pomocou protokolu OAuth. Microsoft a Google využívajú OAuth 2.0 pre prístup ku všetkým svojim rozhraniam API.

2 Oath, Oath ocra

2.1 Oath

OATH skratka znamená Initiative for Open Authentication - jedná sa o spoluprácu zameranú na vývoj otvorenej referenčnej architektúry využívajúcej dostupných štandardov na vytvorenie silnej autentifikácie. Má veľa koordinujúcich a prispievajúcich členov, ktorý navrhujú štandardy pre rôzne autentifikačné technológie so zámerom znížiť náklady a zjednodušiť ich funkcie. Prinášajú riešenia, ktoré umožňujú silnú autentifikáciu všetkých používateľov na všetkých zariadeniach a vo všetkých sieťach. Ich hlavnou víziou je aby v budúcnosti existovala sieť, v ktorej sa spotrebitelia budú cítiť bezpečne pri zadávaní osobných údajov online, kde by mohli obchodní partneri bezpečne spolupracovať a zdieľať údaje medzi doménami [5].

2.2 Oath Ocra

OCRA je skratka pre OATH challenge-response algorithm, je jeden z algoritmov, ktorý slúži na viac-faktorovú autentifikáciu. Jedná sa o jeden z algoritmov, ktorý bol vytvorený práve vďaka OATH. Podstatou takej autentifikácie je, že od používateľa vyžiada pri prihlásení na účet zadať nejaký vstup pred tým, než vytvorí OTP (jednorazové heslo pre prihlásenie). Ten vstup môže byť hocijaký variabilný údaj dohodnutý medzi oboma stranami (tokenom a serverom). Môže sa jednať o nejaké počítadlo, timestamp, PIN, session identifikátor alebo dokonca aj kontrolnú otázku. Vstup musí tiež obsahovať, ktoré údaje a ktoré hashovacie funkcie sa používajú takže ten údaj musí mať aj hlavičku. OCRA algoritmus na rozdiel od typickej autentifikácie umožňuje verifikáciu oboma spôsobmi [6]. Umožňuje koncovému používateľovi zadať výzvu pre server, takže pri vzájomnom režime umožňuje 2 oddelené kalkulácie: 1 client->server a 1 server->client. Vzorec na algoritmus OCRA je nasledovný:

$$\text{OCRA} = \text{CryptoFunction}(\text{K}, \text{DataInput})$$

kde:

- K je zdieľaný súkromný kľúč známy oboma stranami,
- DataInput je štruktúra ktorá obsahuje rôzne hodnoty vstupných údajov,
- CryptoFunction je funkcia, ktorá vykonáva OCRA výpočet zo súkromného kľúča K a z hodnôt DataInput.

Základná CryptoFunction pre ocra je HOTP-SHA1-6, teda jedná sa o HOTP (hash-based message autentifikácia, ktorá vytvorí jednorazové heslo na základe súkromného kľúča a

počítadla) s dynamickým krátením na 6-miestnu hodnotu a kombináciou hodnôt z DataInput pomocou SHA-1 hashovacej funkcie (secure hash algorithm 1 - ktorý spracuje vstup a vyprodukuje 20-bajtovú hashovaciu hodnotu zvyčajne v 40 miestnom hexadecimálnom formáte).

3 SAML

Security Assertion Markup Language je štandard, ktorý umožňuje poskytovateľom odovzdávať autorizačné povolenia. Bližšie to znamená, že môžeme použiť jednu sadu poverení na prihlásenie sa do viacerých webových aplikácií. Vieme oveľa jednoduchšie spravovať prihlásenie používateľa ako keď by sme mali spravovať samostatné údaje do e-mailu, softvéru alebo Active Directory [7]. Transakcie SAML používajú XML jazyk na štandardnú komunikáciu medzi poskytovateľom identity a poskytovateľmi služieb. SAML slúži na prepojenie medzi autentifikáciou používateľa a autorizáciou na používanie služby.

3.1 Použitie

- zjednodušuje korporátne autentifikačné a autorizačné procesy pre používateľov,
- poskytuje riešenie, ktoré umožňuje poskytovateľovi identity a poskytovateľom služieb existovať oddelene od seba,
- implementuje metódu prenosu všetkých autentifikácií a oprávnení používateľov medzi poskytovateľmi služieb a identít

Celá autentifikácia SAML je proces overenia identity a práv používateľa. Autorizácia SAML hovorí poskytovateľovi služby, aký prístup udeliť autentifikovanému používateľovi.

3.2 SAML provider

Poskytovateľ SAML je systém, ktorý pomáha nadobudnúť používateľovi prístup k službe, ktorú potrebujú. Existujú dva základné typy poskytovateľov SAML, poskytovateľa služieb a poskytovateľa identity. Poskytovateľ služieb potrebuje autentifikáciu od poskytovateľa identity, aby udelil autorizáciu používateľovi. Poskytovateľ identity vykonáva autentifikáciu, že koncový užívateľ je, kto hovorí, že je a vysiela tieto údaje poskytovateľovi služieb spolu s prístupovými právami používateľa služby.

Microsoft Active Directory alebo Azure sú bežnými poskytovateľmi identít. Salesforce a iné riešenia CRM sú zvyčajne poskytovateľmi služieb, pretože závisia od poskytovateľa identity pre autentifikáciu používateľa.

3.3 Funkcionalita

SAML funguje na základe prenosu informácie o používateľoch, prihláseniach a atribútoch medzi poskytovateľmi identity a služieb. Každý používateľ sa prihlási raz do služby Single Sign On s poskytovateľom identifikácie a potom môže tento poskytovateľ odoslať ďalším poskytovateľom SAML atribúty. Poskytovateľ služby požaduje autorizáciu a autentifikáciu od poskytovateľa identifikácie. Pretože oba tieto systémy hovoria rovnakým jazykom - SAML - používateľ sa musí prihlásiť iba raz. Každý poskytovateľ identity a

poskytovateľ služieb musí súhlasiť s konfiguráciou pre SAML. Na to, aby autentifikácia SAML fungovala, musia mať obidva endpointy presnú konfiguráciu.

3.4 Výhody

- Neutralita platformy (integrovateľná na takmer každý server)
- Vylepšené online prostredie
- Prenos rizika

3.5 Nevýhody

- Web-based protokol
 - celý tok je založený na presmerovaní a to nie je vhodné pre mobilnú aplikáciu
- Komplexné požiadavky na kryptografiu
 - pre komplexné požiadavky na kryptografiu nemusia byť k dispozícii niektoré mobilné knižnice
- Potreba inštalovať ďalší server na spojazdnenie cez Android - IdentityServer od Thinktecture

4 MQTT a bezpečnosť

MQTT alebo Message Queue Telemetry Transport je protokol výmeny dát určený pre prenos dát na vzdialené miesta, kde je vyžadovaná malá veľkosť kódu a existujú obmedzené šírky pásma. Slúži k vzájomnej interakcii a komunikácií zariadení v rámci IoT.

4.1 Základné vlastnosti MQTT

- Asynchrónny protokol
- Kompaktné správy
- Podpora úrovni kvality služieb (QoS)
- Jednoduchá integrácia nových zariadení

V protokole MQTT prebieha výmena správ medzi klientom (client), ktorý môže byť publisher (poskytovateľ správ) alebo subscriber (príjemca správ), a brokerom správ. Publisher odosiela dáta na centrálny bod – brokerovi (MQTT Broker), sa zobrazuje v správe určitá téma (topic). Subscriberi môžu prijímať rôzne dáta od viac publisherov v závislosti na predplatnom pre dané témy [8].

4.2 MQTT základné typy správ komunikácie s brokerom

- Connect – naviazanie spojenia s brokerom
- Disconnect – prerušenie spojenia s brokerom
- Publish – publikovať správu do témy
- Subscribe – prihlásenie k odberu témy
- Unsubscribe – odhlásenie odberu témy

4.3 Sémantika tém

Témy sú znaky s kódovaním UTF-8. Hierarchia tém má stromovú podobu, čo zjednodušuje ich organizáciu a prístup k dátam. Témy sa skladajú z jednej alebo viac úrovní, ktoré sú oddelené lomítkom "/". Napríklad téma, kde fotopasca monitoruje pohyb zvierat v lesoch A,B,C,D v Tatrách

- /Slovakia/HighTatras/A/1
- /Slovakia/HighTatras/B/0
- /Slovakia/HighTatras/C/0

- /Slovakia/HighTatras/D/1

kde 1 znamená, že pohyb bol zaznamenaný a 0, že nebol.

4.4 Štruktúra správ

Správa MQTT sa skladá z častí:

- Fixná hlavička (prítomná vo všetkých správach)
- Variabilná hlavička (prítomná len v určitých správach)
- Dáta, "náklad" (prítomné len v určitých správach)

4.5 Bezpečnosť MQTT

Bez riadnej autorizácie môže každý overený klient publikovať a prihlásiť sa na odber všetkých dostupných tém. Z hľadiska bezpečnosti by mal byť broker schopný riadiť prístup k témam.

Ak chcete obmedziť klienta na publikovanie alebo prihlásenie sa na odber iba k autorizovaným témam, je potrebné na strane brokera implementovať povolenia k témam. Tieto povolenia musia byť konfigurovateľné a nastaviteľné [9].

Napríklad typy povolenia:

- Allowed topic (povolená téma)
- Allowed operation (publikovať, subscribe, alebo obe)
- Allowed quality of service level (0, 1, 2, všetky)

4.6 Implementácia

Služba **Paho Android** je rozhraním pre klientskú knižnicu Paho Java MQTT pre platformu Android. Pripojenie MQTT je zapuzdrené do služby Android, ktorá beží na pozadí aplikácie pre Android, a udržiava ju pri živote keď sa aplikácia pre Android prepína medzi rôznymi aktivitami.

Android používa Gradle ako vbudovaný systém pre správu dependencies, nasledujúca časť popisuje, ako je možné službu Paho Android pridať do aplikácie prostredníctvom systému Gradle.

```
1 repositories {  
2     maven {  
3         url "https://repo.eclipse.org/content/repositories/paho-releases/"  
4     }  
5 }  
6  
7  
8 dependencies {  
9     compile('org.eclipse.paho:org.eclipse.paho.android.service:1.0.2') {  
10         exclude module: 'support-v4'  
11     }  
12 }
```

Obr. 1: Pridanie dependency

4.6.1 Connect

Aby bolo možné vytvoriť connection k službe Paho Android, musí byť služba deklarovaná v súbore AndroidManifest.xml. Do tagu <application> pridáme nasledujúci úsek kódu:

```
1 <service android:name="org.eclipse.paho.android.service.MqttService" >  
2 </service>
```

Obr. 2: Deklarovanie služby

Na to, aby služba Paho Android fungovala, potrebuje nasledujúce povolenia:

```
1 <uses-permission android:name="android.permission.WAKE_LOCK" />  
2 <uses-permission android:name="android.permission.INTERNET" />  
3 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />  
4 <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Obr. 3: Povolenia

Na vytvorenie a nadviazanie pripojenia MQTT použijeme nasledujúci útržok kódu:

```
1 String clientId = MqttClient.generateClientId();
2 MqttAndroidClient client =
3     new MqttAndroidClient(this.getApplicationContext(), "tcp://broker.hivemq.com:1883",
4                           clientId);
5
6 try {
7     IMqttToken token = client.connect();
8     token.setActionCallback(new IMqttActionListener() {
9         @Override
10         public void onSuccess(IMqttToken asyncActionToken) {
11             // We are connected
12             Log.d(TAG, "onSuccess");
13         }
14
15         @Override
16         public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
17             // Something went wrong e.g. connection timeout or firewall problems
18             Log.d(TAG, "onFailure");
19         }
20     });
21 } catch (MqttException e) {
22     e.printStackTrace();
23 }
24 }
```

Obr. 4: Connection setup

4.6.2 Login/Password connection

Meno a heslo je možné zadať v rámci objektu ako v prípade nižšie:

```
1 options.setUserName("USERNAME");
2 options.setPassword("PASSWORD".toCharArray());
3
4 IMqttToken token = client.connect(options);
```

Obr. 5: Login/Password

4.6.3 Publish

MqttAndroidClient umožňuje publikovanie správ prostredníctvom `publish(topic, MqttMessage)`. Klient `MqttAndroidClient` nevytvára žiadne ukladanie správ ak nie je pripojený. Ak je vypnutý a dostane správu, vyhodí error `client.isConnected() == false`.

```
1 String topic = "foo/bar";
2 String payload = "the payload";
3 byte[] encodedPayload = new byte[0];
4 try {
5     encodedPayload = payload.getBytes("UTF-8");
6     MqttMessage message = new MqttMessage(encodedPayload);
7     client.publish(topic, message);
8 } catch (UnsupportedEncodingException | MqttException e) {
9     e.printStackTrace();
10 }
```

Obr. 6: Publish

4.6.4 Subscribe

Subscribe je možné vytvoriť pomocou metódy `MqttAndroidClient.subscribe`, ktorá ako parameter vezme topic a QOS a vráti `IMqttToken`. Token je možné použiť na sledovanie, či je možné subscription úspešne vytvoriť alebo zlyhalo:

```
1 String topic = "foo/bar";
2 int qos = 1;
3 try {
4     IMqttToken subToken = client.subscribe(topic, qos);
5     subToken.setActionCallback(new IMqttActionListener() {
6         @Override
7         public void onSuccess(IMqttToken asyncActionToken) {
8             // The message was published
9         }
10
11         @Override
12         public void onFailure(IMqttToken asyncActionToken,
13                               Throwable exception) {
14             // The subscription could not be performed, maybe the user was not
15             // authorized to subscribe on the specified topic e.g. using wildcards
16         }
17     });
18 } catch (MqttException e) {
19     e.printStackTrace();
20 }
21 }
```

Obr. 7: Subscribe

4.6.5 Disconnect

Ak chceme klienta odpojiť, zavoláme metódu `disconnect` objektu `MqttAndroidClient`. Nasledujúci príklad uloží token vrátený metódou `disconnect` a informuje `IMqttActionListener` aby dostal upozornenie na úspešné odpojenie klienta.

```
1  try {
2      IMqttToken disconToken = client.disconnect();
3      disconToken.setActionCallback(new IMqttActionListener() {
4          @Override
5          public void onSuccess(IMqttToken asyncActionToken) {
6              // we are now successfully disconnected
7          }
8
9          @Override
10         public void onFailure(IMqttToken asyncActionToken,
11                               Throwable exception) {
12             // something went wrong, but probably we are disconnected anyway
13         }
14     });
15 } catch (MqttException e) {
16     e.printStackTrace();
17 }
```

Obr. 8: Disconnect

5 JMS, AMQP a MQTT

5.1 Posielanie správ pre účely autorizácie

Posielanie správ je základným komunikačným mechanizmom, ktorý má úspech po celom svete. Zasielanie správ je bežnou metódou komunikácie. Existujú 2 základné mechanizmy, ktoré by sme použili na výmenu správ medzi 2 (alebo viacerými) stranami:

- Synchronne posielanie správ
- Asynchronne posielanie správ

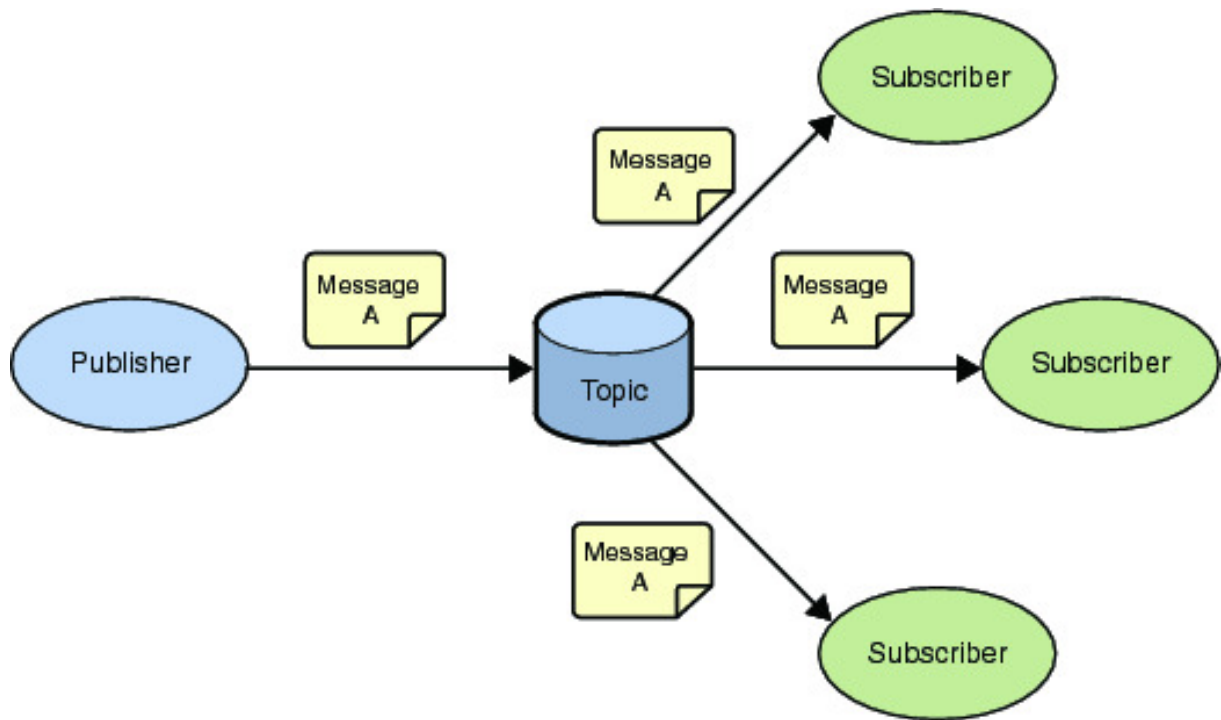
Keďže našou úlohou je vytvoriť autorizačný server, rozhodli sme sa využiť asynchrónne posielanie správ, ktoré je pre náš účel výhodnejšie z niekoľkých dôvodov:

- Účastníci oboch strán konverzácie majú slobodu začať, pozastaviť a pokračovať v konverzačných správach podľa vlastných podmienok
- Asynchrónne posielanie správ rieši problém prerušovanej konektivity
- Škálovateľnosť - krátka správa môže byť odoslaná s veľmi dlhou odpoveďou alebo naopak.

Dostávame sa teda ku asynchrónnym spôsobom komunikácie (z angl. asynchronous messaging), ktoré sa pokúsime porovnať a najvýhodnejší spôsobom neskôr implementovať.

5.2 Java Messaging Service

JMS je jednou z najúspešnejších dostupných technológií asynchrónneho zasielania správ. S nárastom osvojenia si Javy vo veľkých podnikových a korporátnych aplikáciách sa JMS stal prvou voľbou pre podnikové systémy. Definuje API pre budovanie systémov zasielania správ.



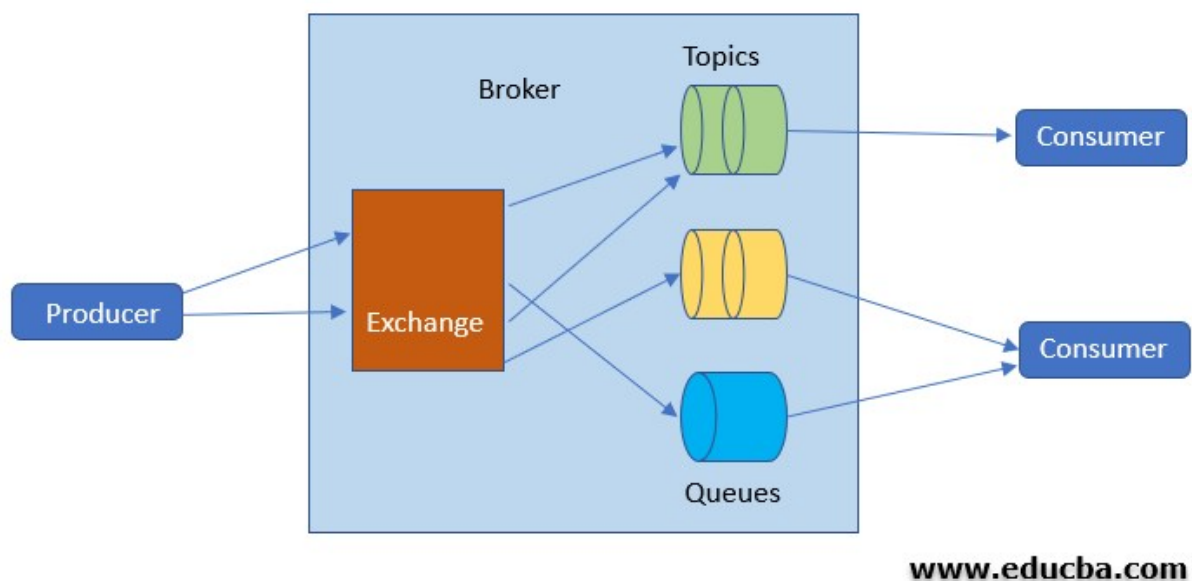
Obr. 9: Java Messaging Service

- Štandardné rozhranie API na odosielanie správ pre platformu JAVA
- Podporuje 2 modely správ: pre queue a topic
- Definuje formát správy (head, properties and body)

5.3 Advanced Message Queueing Protocol

Prišiel problém tzv. interoperability. ide o to, ako môžu 2 programy napísané v 2 rôznych programovacích jazykoch medzi sebou komunikovať cez asynchrónne posielanie správ.

Tu prichádza požiadavka definovať spoločný štandard pre asynchrónne zasielanie správ. AMQP sa zaoberá týmto problémom a prišiel so štandardným protokolom a mnohými ďalšími funkciami na podporu interoperability a mnohých iných funkcií pre odosielanie správ pre moderné aplikácie.



Obr. 10: AMQP protokol

Základné vlastnosti AMQP:

- Interoperabilita medzi viacerými jazykmi a platformami
- Škálovateľnosť
- Na zabezpečenie používa SASL a TLS
- Podporuje klasické fronty správ, ukladanie a posielanie ďalej
- Podporuje servery zabezpečenia proxy

5.4 Využitie ActiveMQ pre účely autorizácie

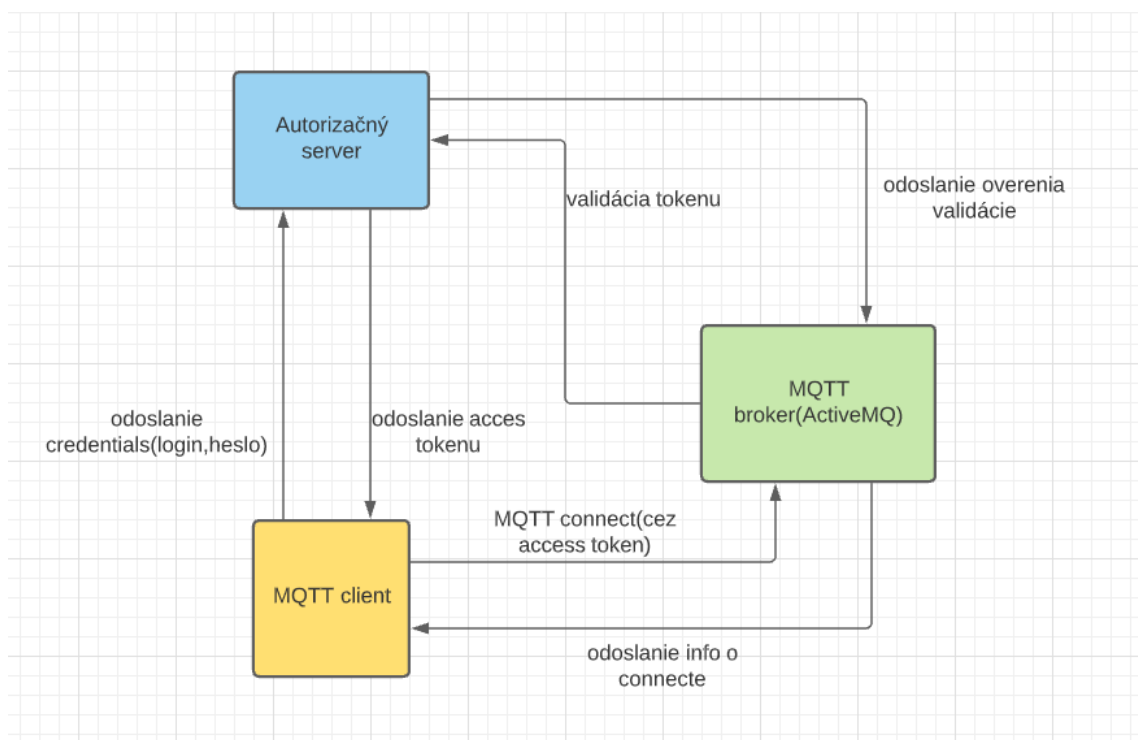
ActiveMQ je open source protokol vyvinutý spoločnosťou Apache, ktorý funguje ako implementácia middleware orientovaného na správy (MOM). Jeho základnou funkciou je posielanie správ medzi rôznymi aplikáciami.

Podporuje JMS, MQTT aj AMQP protokoly.

5.5 Návrh riešenie prostredníctvom ActiveMQ brokera

Riešenie by spočívalo v 6 krokoch:

- 1. odoslanie credentials(login, heslo)
- 2. odoslanie access tokenu (zo strany serveru)
- 3. MQTT connect(prostredníctvom access tokenu)
- 4. Validácia tokenu na strane autorizačného servera
- 5. Odoslanie overenia validácie pre MQTT brokera
- 6. Odoslanie informácie o pripojení pre MQTT klienta



Obr. 11: zdroj: vlastné spracovanie

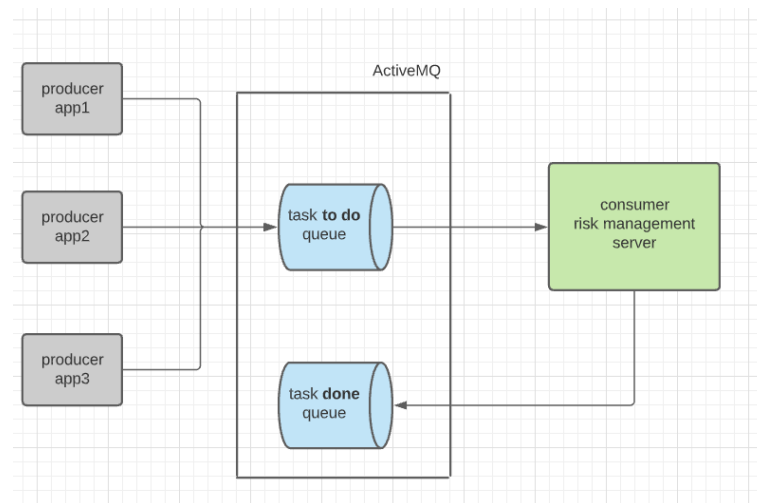
MQTT klienti sú publisheri pre topic obsahujúci autorizačné informácie. Klienti tento topic nemôžu vidieť, ale môžu do neho publishovať. Autorizačný server by predstavoval subscribera pre daný topic a vyberal by z neho informácie potrebné pre overenie autorizácie.

5.6 Architektúra ActiveMQ Brokera

Problém pri pripojení viacerých klientov naraz by mohol spôsobiť, že risk management server by nevedel, ktoré dáta skôr spracovať, resp. ktoré dáta komu a kam priradiť. Preto prichádza otázka či na riešenie pri mqtt protokole využiť queue alebo topic.

5.6.1 Queue vs topic

Queue ActiveMQ je retazec správ, do ktorého prichádza správa a smeruje len k jednému účastníkovi. Topic ActiveMQ je kanál správ, kde správa prichádza a odchádza ku každému účastníkovi. Queue ActiveMQ aj topic ActiveMQ sú miesta, kam sa odosielajú správy. Rozdiel je v tom, kto správu dostane.



Obr. 12: Architektúra ActiveMQ Brokera

Pre naše účely sme sa rozhodli využiť Queue ActiveMQ, z dôvodu, že máme iba jedného consumera a viacero producerov. Môže to byť neefektívne, keby sme chceli program podrobiť budúcemu škálovaniu, avšak z bezpečnostných dôvodov je to výhodnejšie, keďže ďalším rozdielom medzi queue a topic je, že queue sa odošle vždy iba jednému consumerovi, tým pádom si môžeme byť istý, že nikto iný nikdy správu nedostane, iba náš risk management server.

6 API security

Bezpečnosti je dnes jedna z hlavných tém v IT sektore a to rozhodne právom. Systémy dnes obsahujú veľké množstvo API a často komunikujú medzi sebou. Takýto spôsob je efektívne ale môžeme to považovať za zraniteľnosť. Vedenie tímov si často neuvedomuje aká veľká je komunikácia medzi týmito API. Treba si uvedomiť aj fakt že pri projekte je často nutné využiť externú API na ktorú sa spoliehame avšak neuvedomujeme si či je táto API dostatočne zabezpečená. Ak by táto API bola terčom útoku tak v najlepšom prípade stratíme funkcionality aj našej aplikácie, v horšom prípade môže hacker napadnúť náš systém práve cez tento komunikačný kanál [10]. Aby sme našu aplikáciu čo najviac zabezpečili musíme sa vyvarovať týmto častým chybám:

1. Slabá alebo žiadna autentifikácia
2. Šifrovanie citlivých údajov
3. Posielanie viac dát ako je potrebné
4. Zabudnuté endpointy alebo shadow API
5. Podozrivé používanie API

6.1 Slabá alebo žiadna autentifikácia

Ťažko sa tomu verí ale veľká časť útokov na API je z tohto dôvodu. Pritom nieje zložité implementovať napríklad API key alebo Token. Táto funkcionality do značnej miery zabrániť útočníkovi napadnúť náš systém, alebo sa rozhodne pre iný cieľ. Pri starších riešeniach, ak nebola implementovaná takáto funkcionality tak sa rozhodne odporúča doimplementovať primeranú autentifikáciu.

6.2 Šifrovanie citlivých údajov

Rovnako ako v prvom prípade to znie triviálne, ale napríklad bankové údaje ako číslo účtu by rozhodne mali byť šifrované počas celej komunikácie. Základ je použiť HTTPS čo však nešifruje samotné dáta. Samotný druh šifrovania môže byť ľubovoľný, ale SHA-256 alebo posielanie dát v JWT tokene.

6.3 Posielanie viac dát ako je potrebné

Netýka sa to len SOAP implementácie ale aj v RESTful implementácií sa často stáva že je nutné aby boli vrátené podobné objekty napríklad preukazy pre hľadajú osobu. Takýmto spôsobom posielame viac údajov ako by sme chceli a ak si to analytik pri návrhu nevšimne vzniká bezpečnostná diera. Počas vývoja sú používané nástroje ako napríklad Swagger,

kde sú zobrazené všetky voliteľné polia pre kontrétny request. Ak sa niektoré rozširujúce parametre nevyužívajú a sú nasadené do produkčného prostredia, môže to viesť k úniku informácií.

6.4 Zabudnuté endpointy alebo shadow API

V predchádzajúce sme načrtli čo sa môže stať ak programátori “zabudnú po sebe upratať”. V tomto prípade sa stáva že sú zabudnuté niektoré API ktoré sa používali pri testovaní napríklad databázy. Tieto endpointy často nie sú ani v dokumentácii a tester ich nekontrolujú. V inom prípade nie sú skontrolované všetky HTTP metódy pre endpoint. Na endpoint je posielaný POST na vytvorenie ale môže byť zabudnutý PUT. Tieto chyby ak sú odhalené môžu viesť k útoku a firma si to vôbec nevšimne.

6.5 Podozrivé používanie API

Tie najdôležitejšie API by mali byť dobre zabezpečené proti podozrivému správaniu. Jedná sa o endpointy ktoré budú s najväčšou pravdepodobnosťou cieľom útokov. Medzi takéto patria: Autentifikácia, Reset hesla, fakturácia , platby a všetky citlivé transakcie [11]. Samoregulačné opatrenia ktoré vieme podniknúť sú:

1. geopolitické obmedzenie: blokovať requesty z krajín mimo nášho zaujmu
2. blokovanie Opakovaných requestov vyžadujúce veľké množstvo dát
3. rozoznávanie podozrivého správania: brute force útok na login endpoint
4. implementovať Rate limiting: obmedziť množstvo requestov – lepšia stabilita

7 OpenCV - face recognition

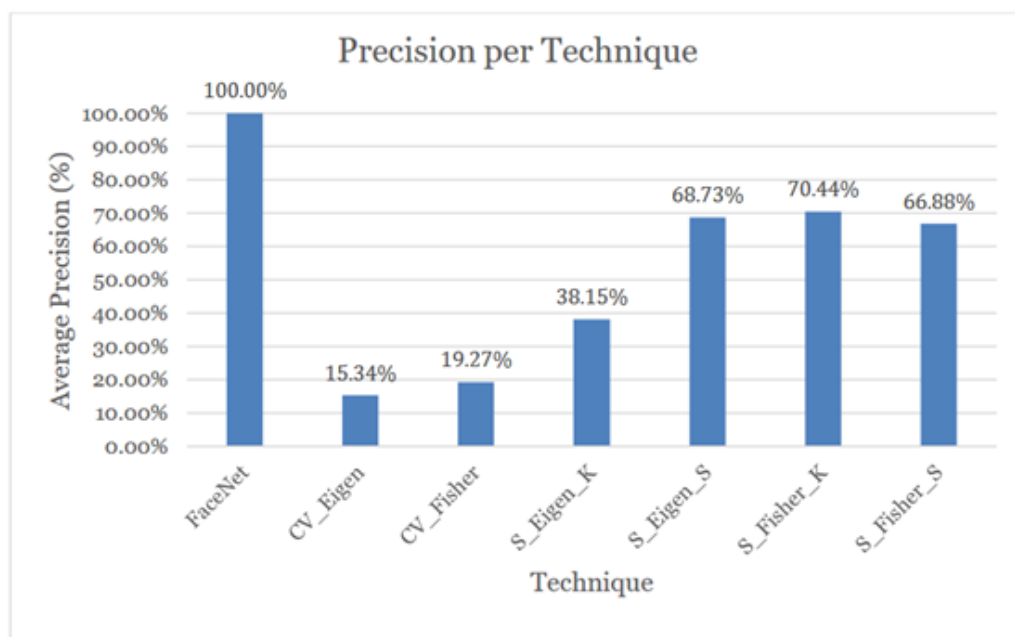
Definícia detekcie tváre je problém počítačového videnia pri lokalizácii a lokalizácii jednej alebo viacerých tvárí na fotografii. Umiestnenie tváre na fotografii znamená nájsť súradnice tváre na obrázku, zatiaľ čo lokalizácia znamená vymedzenie rozsahu tváre, často pomocou ohraničujúceho rámčeka okolo tváre.

Detekciu tvárí na fotografii môžu ľudia ľahko vyriešiť, aj keď to bolo z historického hľadiska pre počítače vzhľadom na dynamickú povahu tvárí náročné. Tváre musia byť napríklad rozpoznávané bez ohľadu na orientáciu alebo uhol, ktorým sú obrátené, úroveň svetla, oblečenie, doplnky, farbu vlasov, vlasy na tvári, make -up, vek a podobne [12].

Pri danej fotografii systém detekcie tváre vydá nula alebo viac ohraničujúcich políčok, ktoré obsahujú tváre. Zistené tváre potom môžu byť poskytnuté ako vstup do nasledujúceho systému, napríklad systému rozpoznávania tvárí [13]. Existujú možno dva hlavné prístupy k rozpoznávaniu tvárí: metódy založené na funkciách, ktoré na vyhľadanie a detekciu tvárí používajú ručne vyrobené filtre, a metódy založené na obrázkoch, ktoré sa holisticky učia extrahovať tváre z celého obrázka.

Pre naše riešenie by bola využitá knižnica OpenCV v integrácii s Android aplikáciou napísanou v Kotlin. OpenCV je open-source knižnica pre rôzne platformy, ktorá obsahuje súbor algoritmov pre prácu s počítačovým videním a obrazom a ich spracovanie. V prípade kedy chceme využívať face recognition pre účely autentifikácie je potrebné poskytnúť fotografie užívateľa [14]. V ideálnom prípade by sme mali mať 10 až 20 obrázkov na osobu, ktorú chceme rozpoznať a treba zohľadniť nevýhody, obmedzenia a spôsob, ako dosiahnuť vyššiu presnosť rozpoznávania tváre. Na tomto datasete by sa natrénovala neurónová sieť v rámci OpenCV knižnice a tá by nám potom pri spätnom overovaní vracala potvrdzujúcu informáciu o totožnosti užívateľa.

8 Metódy pre rozpoznávanie tváre



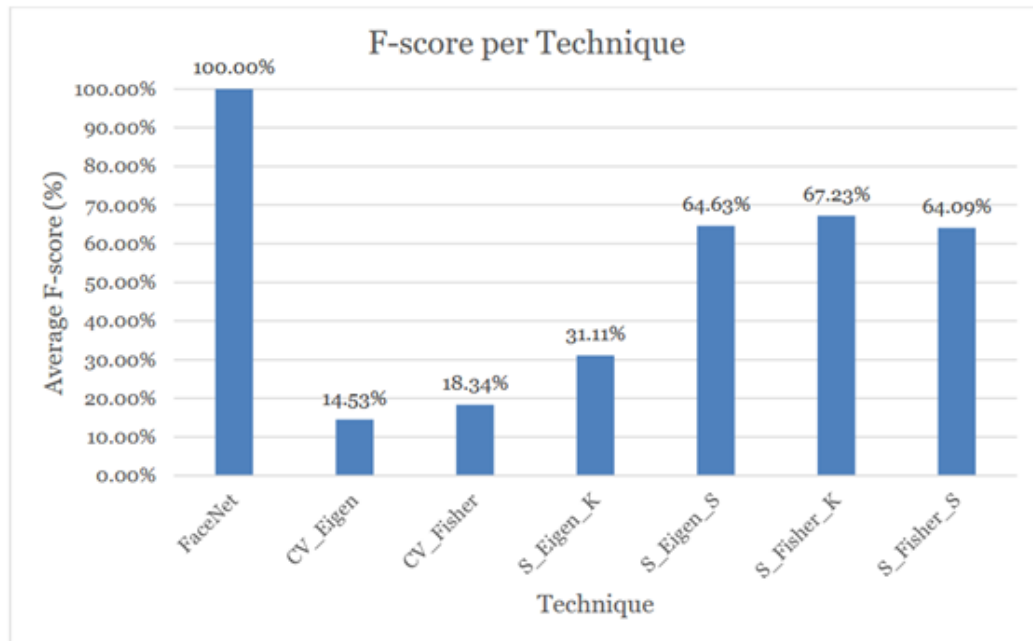
Obr. 13: Porovnanie konvolučných techník rozpoznania tváre

Tabulky zobrazujú experiment na porovnanie konvolučných techník rozpoznávania tváre - Neurónová sieť (CNN), Eigenface s klasifikátormi K-Nearest Neighbors (KNN) a podporujú vektorové stroje (SVM) a Fisherface s klasifikátormi KNN a SVM pod rovnaké podmienky s obmedzenou verziou súboru údajov LFW. Knižnice Python scikit-learn a OpenCV, ako aj implementácia CNN FaceNet. Výsledky ukazujú, že FaceNet je najlepšia technika vo všetkých metrikách okrem času predpovede. FaceNet dosiahol F-skóre 100 %, zatiaľ čo implementácia OpenCV Eigenface pomocou Najhoršie dopadla SVM s 15,5 %. Technika s najnižším predpovedným časom bola scikit- naučiť sa implementáciu Fisherface s SVM. Presnosť je vypočítaná pomocou delenia totálnych predikcií so všetkými pozitívnymi predikciami, ktoré sú relevantné alebo nie. Matematicky: $TP/TP+FP$

8.1 F-Score

F-score je pravdepodobne najlepšia metrika pre účely porovnania úspešnosti techniky, v ako aj správnych predikciách tak aj v nerobení nesprávnych predikcií. Matematická formulácia pre F-score je:

$$(Precision * Recall / Precision + Recall) * 2$$



Obr. 14: Porovnanie konvolučných techník rozpoznania tváre cez F-Score

8.2 Použité techniky

Tri techniky použité v tejto práci boli Eigenface, Fisherface a CNN a klasifikátory boli SVM a KNN. Aby sa tieto techniky otestovali, knižnice OpenCV a bol použitý scikit-learn, ako aj FaceNet. Každá technika bola testovaná pomocou toho istého súboru údajov s 10-násobnou krížovou validáciou a desiatimi nasadenými cyklami na sklade. Najlepšou technikou pri porovnávaní metrického výkonu je FaceNet. Tri zo štyroch Techník scikit-learn by sa mohli použiť v niektorých systémoch, kde presnosť nie je najvyššia dôležitosť, ako je systém označovania tvárí, kde by mohlo dôjsť k chybným predpovediam bez toho, aby ste to urobili akékoľvek významné poškodenie. Testy ANOVA s hladinou významnosti 0,05 dokazujú, že existuje významný rozdiel vo výkonnosti medzi technikami vo všetkých metrikách. Tukey post hoc test s hladinou významnosti 0,05 ukazuje, že FaceNet funguje výrazne lepšie na všetkých metriky výkonnosti. FaceNet pri používaní dosahuje najlepší čas tréningu zo všetkých techník v experimente predtrénovaný model. Test ANOVA dokazuje, že existuje významný rozdiel v tréningu času medzi všetkými technikami. Tukey post

hoc test ukazuje, že FaceNet má výrazne kratší čas tréningu ako všetky ostatné techniky. Test ANOVA dokazuje, že medzi všetkými existuje významný rozdiel v čase predpovede techniky. Tukey post hoc test ukazuje, že Eigenface s implementáciou SVM z OpenCV má výrazne horší čas predpovede v porovnaní so všetkými ostatnými technikami. nie je možné dokázať významný rozdiel medzi ostatnými technikami v čase predpovede. Dokonca hoci v implementácii Fisherface scikit-learn nemožno dokázať žiadny významný rozdiel používanie SVM má čas predpovede, ktorý je o 8,6 rýchlejší ako FaceNet a je potrebné ho zvážiť ak sú predpovedné časy dôležitejšie ako výkon. Ďalšie merania na väčšie súbory údajov by mohli viesť k významnému rozdielu v čase predpovede medzi týmito dvoma technikami.

Models	Average Training Time (sec)
FaceNet	0.24
opencv_eigen_svm	3.68
opencv_fisher_svm	4.19
scikit_eigen_knn	0.38
scikit_eigen_svm	0.43
scikit_fisher_knn	1.02
scikit_fisher_svm	0.93

Obr. 15: Čas trénovania

Models	Average Prediction Time (sec)
FaceNet	0.0067
opencv_eigen_svm	0.0982
opencv_fisher_svm	0.0075
scikit_eigen_knn	0.0070
scikit_eigen_svm	0.0034
scikit_fisher_knn	0.0029
scikit_fisher_svm	0.0007

Obr. 16: Čas predikovania

8.3 FaceNet

FaceNet je systém na rozpoznávanie tvárí vyvinutý v roku 2015 výskumníkmi zo spoločnosti Google, ktorý vtedy dosiahol najmodernejšie výsledky v rade referenčných súborov údajov na rozpoznávanie tvárí. Systém FaceNet môže byť široko používaný vďaka viacerým implementáciám modelu s otvoreným zdrojom od tretích strán a dostupnosti vopred pripravených modelov. Systém FaceNet možno použiť na extrahovanie vysoko-kvalitných prvkov z tvárí, nazývaných vloženie tváre, ktoré sa potom dajú použiť na tréovanie systému identifikácie tváre. Google FaceNet popísal vo svojom dokumente z roku 2015 s názvom:

„FaceNet: Jednotné vkladanie pre rozpoznávanie a klastrovanie tvárí“

Je to systém, ktorý na základe obrázku tváre vytiahne z tváre vysokokvalitné črty a predpovedá vektorovú reprezentáciu týchto črt so 128 prvkami, nazývanú vloženie tváre. FaceNet, ktorý sa priamo učí mapovanie z obrázkov tvárí do kompaktného euklidovského priestoru, kde vzdialenosti priamo zodpovedajú miere podobnosti tvárí. Model je hlboká konvolučná neurónová sieť tréovaná pomocou funkcie straty tripletov, ktorá podporuje, aby sa vektory pre rovnakú identitu stali podobnejšie (menšia vzdialenosť), zatiaľ čo sa očakáva, že vektory pre rôzne identity budú menej podobné (väčšia vzdialenosť). Dôležitou inováciou v tejto práci bolo zameranie sa na tréovanie modelu na priame vytváranie vložení (namiesto ich extrahovania z medzivrstvy modelu). Tieto vloženia tváre sa potom použili ako základ pre tréovanie klasifikačných systémov na štandardných referenčných súboroch údajov na rozpoznávanie tváre, čím sa dosiahli najmodernejšie výsledky. Je to robustný a efektívny systém rozpoznávania tvárí a všeobecná povaha extrahovaných vložiek tváre prepožičiava prístup k rôznym aplikáciám.

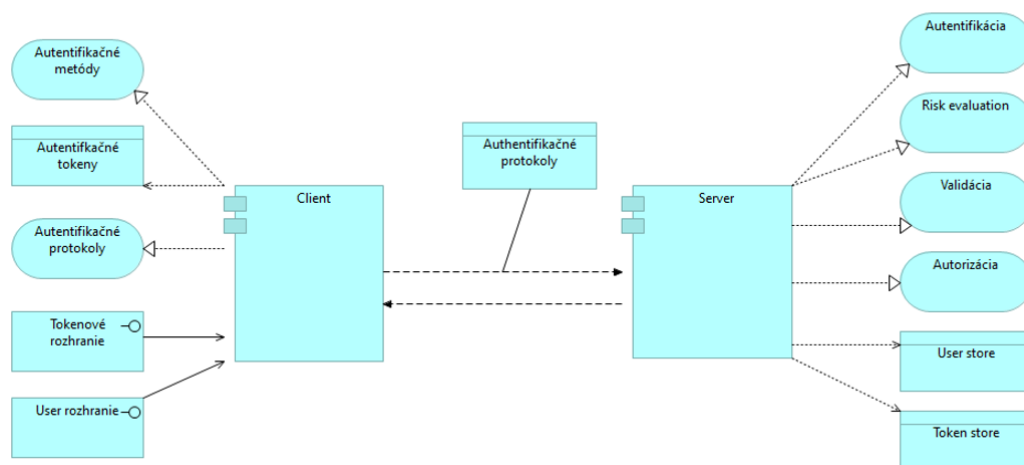
9 Návrh riešenia

9.1 Architektúra

V architektúre zohráva autentifikácia významnú rolu. Ak by došlo k chybnému návrhu náprava by mohla stáť veľa času a úsilia. V našom prípade však návrh nebude zložitý. Chceme implementovať autorizačné tokeny.

9.1.1 Autentifikačná platforma

Na serveri plánujeme implementovať okrem funkčných API pre chod aplikácie aj tokeny slúžiace na autentifikáciu. JWT token je rozumná voľba lebo je šifrovaný a dajú sa v ňom obsiahnuť aj doplnkové dáta. Klieta vieme naprogramovať aby posielal s požiadavkami token a na základe toho ho autentifikujeme. Nevýhodou je ak by tento token bol odcudzený, útočník s týmto tokenom môže posilať požiadavky ako obeť.



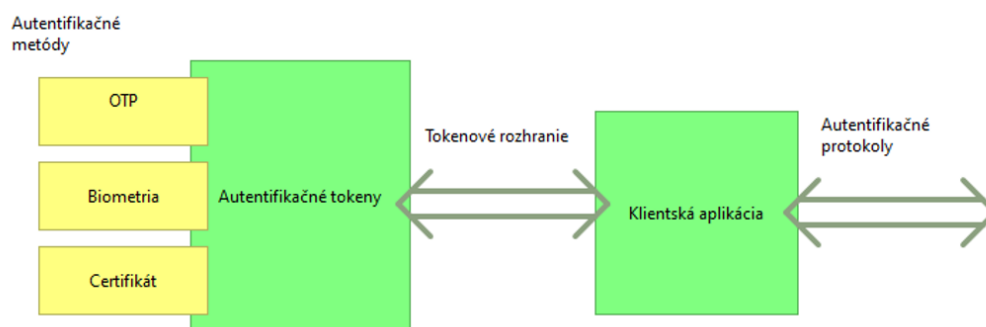
Obr. 17: High level architektúra

Token store Po autentifikácii priradí token k užívateľovi. K tomuto bodu musia mať prístup všetky časti architektúry, ktoré chceme aby boli zabezpečené prihlásením.

User store V user store sa budú uchovávať informácie u používateľoch a ich rolách v systéme. Taktiež ich prihlasovacie meno, heslo a e-mailová adresa.

9.1.2 Klientske rozhranie

Autentifikácia klient prebieha v Android aplikácií, ktorá bude mať viacero foriem. Či už biometria, odtlačok prstu, rozpoznanie tváre alebo OTP, čiže jednorázové heslo. Z bezpečnostného hľadiska je dôležité zabezpečiť aplikáciu čo najlepšie, aby nebolo možné zneužiť jej autentifikačné funkcie.



Obr. 18: Klienske rozhranie

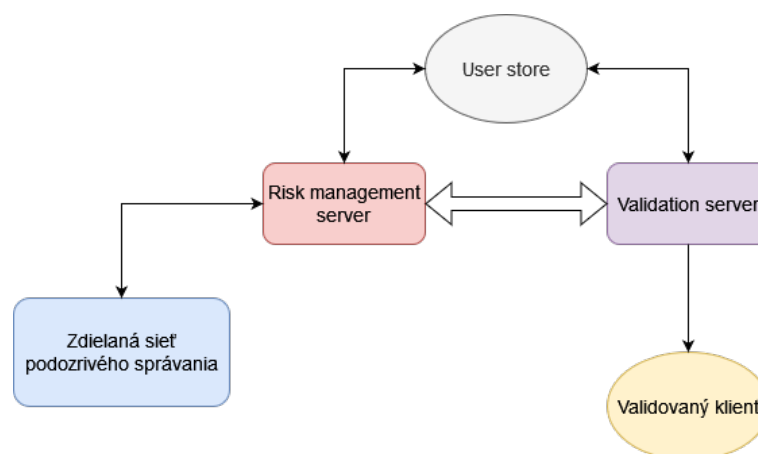
One Time Password Výhodou tohto typu autentifikácie je, že aj keby došlo k jeho ukradnutiu, toto heslo nieje možné zneužiť. Ich krátka platnosť je zároveň aj ich nevýhodou, lebo používatelia si ich nemôžu uložiť lebo by boli neplatné.

Biometria Biometria celkovo je ako autentifikačný prvok účinný. Na jej implementáciu je nutné mať pokrokovú infraštruktúru. Limitom tejto autentifikácie sú hardvérové čítačky. Dostatočne dobré kamery alebo čítačku otlakov prstov. Keďže dnešné smartfóny obsahujú tieto hardvérové prvky, súčasťou našej aplikácie bude aj forma biometrickej autentifikácie.

Certifikát Autentifikácia založená na certifikáte je použitie digitálneho certifikátu na identifikáciu používateľa alebo zariadenia pred udelením prístupu k zdroju, sieti, aplikácii atď. V prípade overenia používateľa sa často používa v koordinácii s tradičnými metódami, ako sú napríklad používateľské meno a heslo.

9.1.3 Risk manažment

Na našom serveri bude implementovaný modul na monitorovanie podozrivého správania. Takisto chceme ohodnocovať jednotlivé transakcie aby sme vedeli nastaviť dostatočné autentifikačné opatrenia. Bolo by zaujímavé pripojiť tento modul do zdieľanej databázy vzorov podozrivého správania. Každá činnosť ktorú bude vykonávať užívateľ prejde spracovaním do API ktorá následne hodnotu číselníku ako veľmi podozrivá táto akcia je a či je potrebná dodatočná autentifikácia alebo dokonca zakáže túto akciu. Môže sa jednať geopolitické alebo historické obmedzenia.



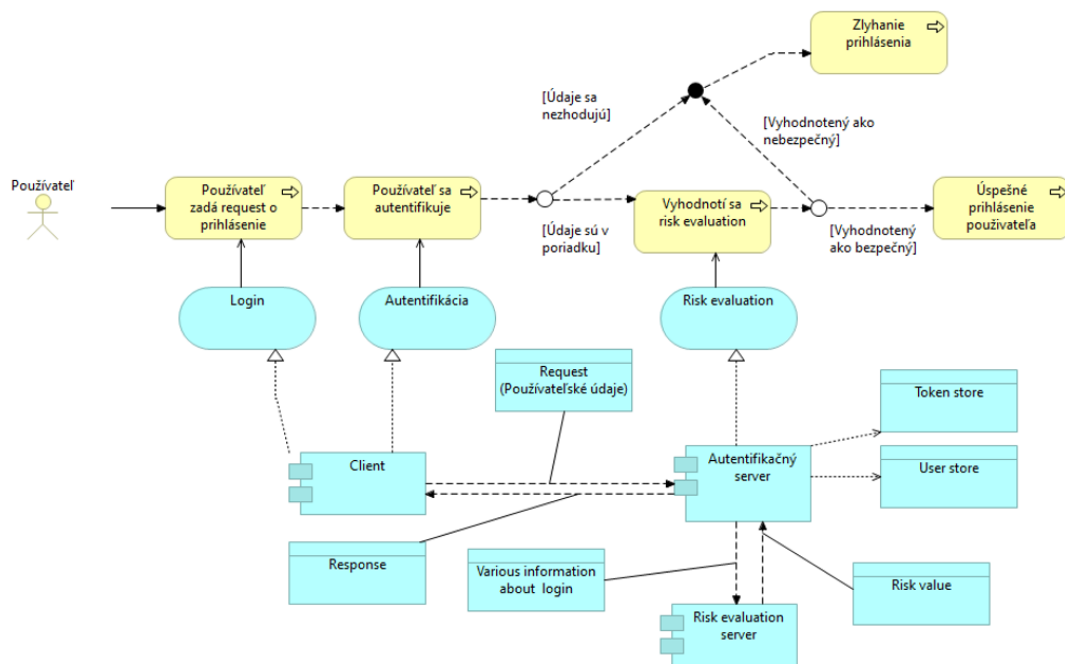
Obr. 19: Risk manažment

9.1.4 Dvojfaktorová autentifikácia

Táto funkcionálnosť je najdôležitejšou súčasťou nášho systému. Je potrebné aby systém umožňoval autentifikáciu z mobilného telefónu a aj stolového počítača či notebooku. Po nastavení aplikácie na konkrétnom zariadení, sa z neho stane autentifikačný zdroj. Budú sa na ňom generovať krátke heslá. V akom budú tvare si ukážeme neskôr v návrhu.

9.1.5 Prihlásenie

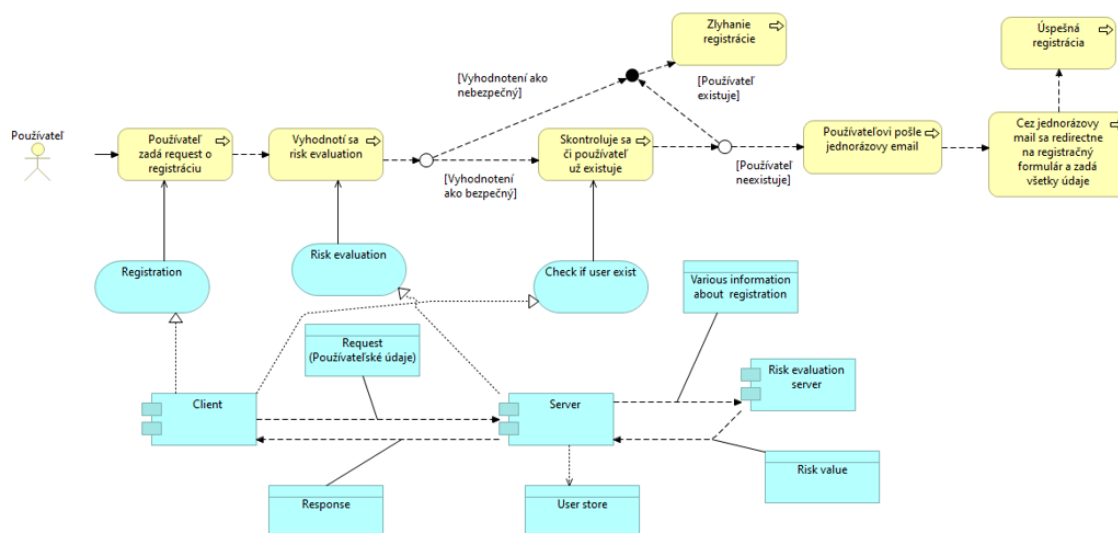
Používateľ zadá request (spolu s používateľskými údajmi) na prihlásenie sa do systému. Používateľ sa následne autentifikuje či sú údaje v poriadku. Ak sú tak sa ešte vyhodnotí risk evaluation. Ak je vyhodnotený ako bezpečný a autentifikácia je úspešná tak používateľa úspešne prihlási. Ak je riziko príliš vysoké alebo sa údaje nezhodovali tak zlyhá prihlásenie a vypíše dôvod.



Obr. 20: Proces prihlásenia

9.1.6 Registrácia

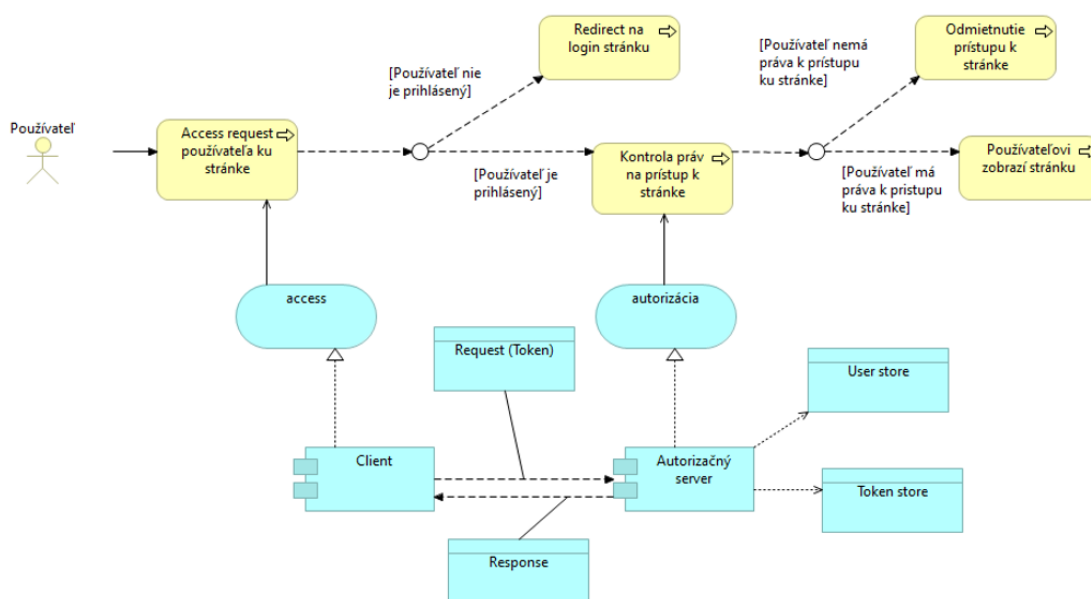
Používateľ zadá request (spolu s používateľskými údajmi) na registrovanie sa do systému. Systém najprv vyhodnotí riziko risk evaluation, ak je vyhodnotený ako nebezpečný tak zlyhá registrácia. Ak je vyhodnotený ako bezpečný tak sa skontroluje, či daný používateľ už neexistuje v databáze(user store). Ak existuje tak zlyhá registrácia. Ak neexistuje tak používateľovi pošle na zadaný email jednorázový email na ktorom môže pokračovať v registrácii. Následne ak vyplní všetky údaje vo formulári tak ho úspešne registruje a vypíše mu message o úspešnej registrácii.



Obr. 21: Proces registrácie

9.1.7 Autorizácia

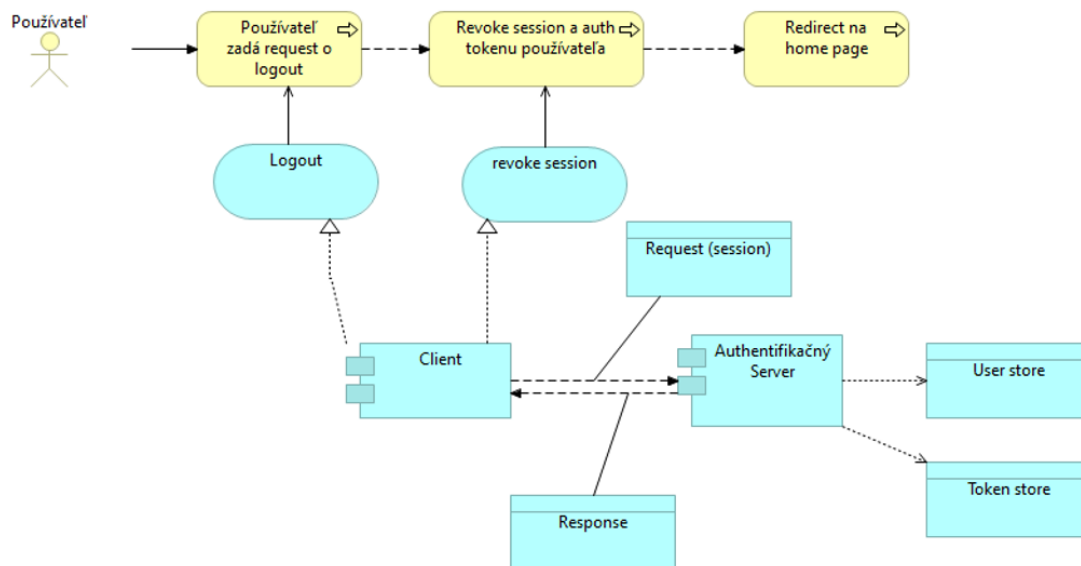
Na obrázku môžeme vidieť postup autorizácie používateľa pri požiadaní o prístup na nejakú stránku. Najprv používateľ zadá request o prístup na stránku, systém overí či je vôbec prihlásený, ak nie je tak ho presmeruje na stránku s loginom. Ak je prihlásený, tak používateľa následne autorizuje a teda skontroluje jeho token, či má práva na prístup k danej stránke alebo nie. Ak nemá práva tak mu odmietne prístup ku stránke a zobrazí nejakú správu o nepovolenom vstupe. Ak má tak mu zobrazí stránku ktorú chcel navštíviť.



Obr. 22: Proces autorizácie

9.1.8 Odhlásenie

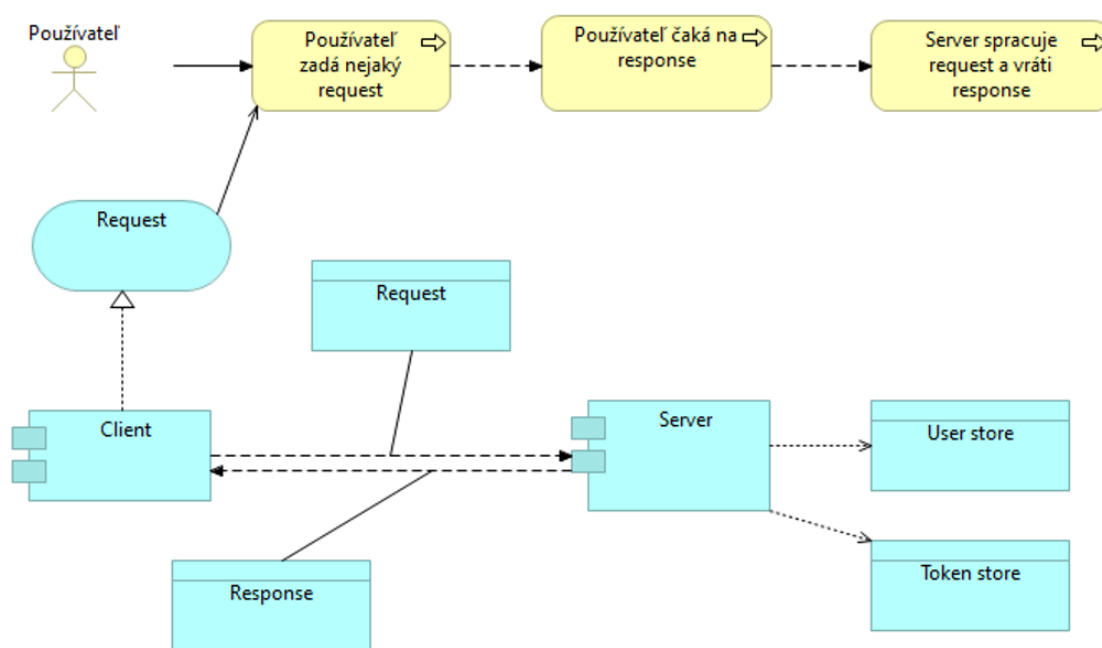
Na obrázku vidíme proces používateľa pre odhlásenie sa zo systému. Najprv používateľ zadá request, že sa chce odhlásiť z účtu. Systém spracuje tento request a odstráni jeho lokálnu session a tiež tú v databáze. Po úspešnom vymazaní session ho presmeruje na home page.



Obr. 23: Proces odhlásenia

9.1.9 Synchronne servisy

Hlavnou črtou synchronných services je, že keď používateľ zadá nejaký request tak čaká kým sa service dokončí, dovedy nemôže vykonať ďalší request. Až keď dostane odpoveď tak môže používateľ pokračovať vo vykonávaní ďalších operácií. Pri synchronných sa zvykne zadávať maximálny čas za ktorý sa má vykonať operácia. V prípade, že by šlo o nekonečný loop aby sa nezastavila prevádzka. Proces synchronného spracovania nejakého všeobecného requestu môžeme vidieť na obrázku nižšie.

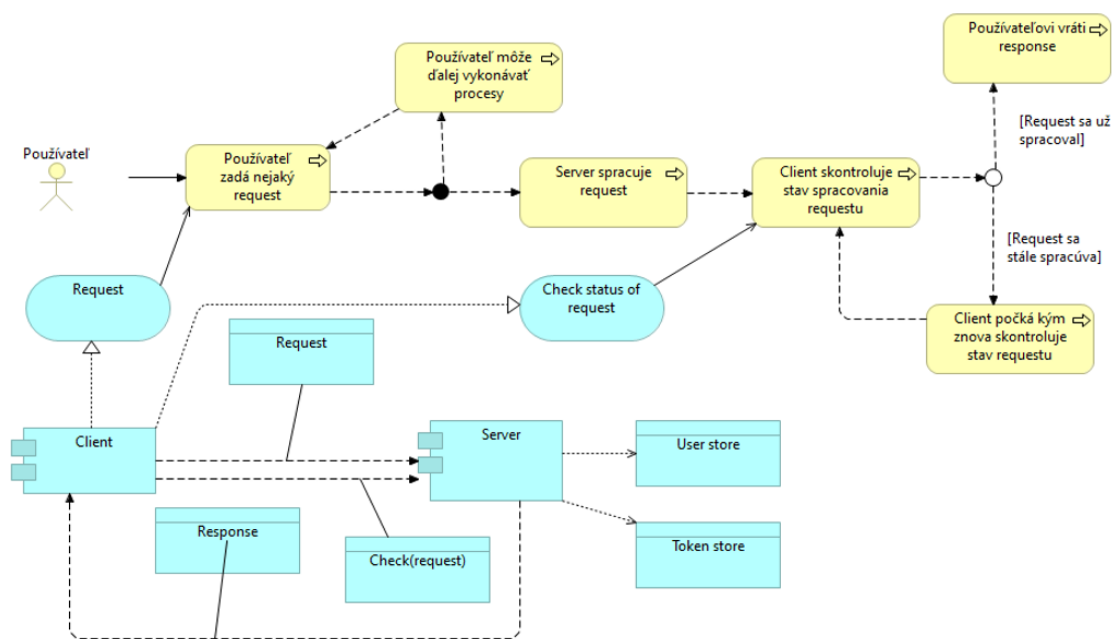


Obr. 24: Synchronný proces

Ako sa už spomínalo používateľ zadá request a čaká na response. Kým používateľ čaká tak nemôže robiť nič iné. Počas toho ako používateľ čaká, server spracuje jeho request a vráti response. Až po tom ako vráti response môže používateľ pokračovať vo vykonávaní operácií a poprípadе zadať nový request

9.1.10 Asynchrónne servisy

Pri asynchrónnych services je hlavnou črtou to, že keď používateľ zadá nejaký request, tak môže ďalej vykonávať iné operácie počas toho ako systém vyhodnotí jeho request a vráti response. V porovnaní so synchrónnymi systémom potrebuje viac času na vyhodnotenie requestu. Pri asynchrónnom by bolo vhodné nastaviť čas po ktorom client skontroluje stav spracovania requestu. Proces celého asynchrónneho vykonávania requestu môžeme vidieť na nasledujúcom diagrame.



Obr. 25: Asynchrónny proces

Na diagrame môžeme vidieť, že najprv používateľ zadá request. Server začne spracovať request používateľa. Počas jeho spracovania môže používateľ ďalej vykonávať rôzne procesy popr. zadávať ďalšie requesty. Počas toho ako server spracúva request tak sa client raz za čas spýta na to aký je stav spracovania requestu. Ak ešte nie je vyhodnotený response tak počká istý čas kým sa znova spýta. Ak je spracovanie requestu ukončené tak vráti používateľovi response.

9.2 Server

9.2.1 CAS

Central Authentication Service je protokol jednotného prihlásenia pre web. Jeho účelom je umožniť používateľovi prístup k viacerým aplikáciám a zároveň poskytnúť svoje prihlasovacie údaje (napríklad používateľske meno a heslo) iba raz. Webovým aplikáciám tiež umožňuje overovať používateľov bez toho, aby získali prístup k bezpečnostným povoleniam používateľa, ako je napríklad heslo. Názov CAS tiež odkazuje na softvérový balík, ktorý implementuje tento protokol.

Výhody

- Kvalitný reporting cez logy
- Pravidelné aktualizácie JAR
- Vhodný pre viacero podstránok

Nevýhody

- Ťažší na implementáciu
 - *First-time* setup je ťažší a nie úplne intuitívny
- Nepravidelné aktualizácie na docker
 - Keďže sa môže meniť cesta ukladania kľúča, tak docker image treba aktualizovať
- Nedostatočná ochrana účtov
 - Ak by sa útočníkovi podarilo získať jeden z hlavných účtov, tak vie využívať aj ostatné účty

9.2.2 Keycloak

Keycloak je *open-source* riešenie pre správu identity a prístupu. Keycloak ponúka Single-Sign On(SSO), SAML aj OAuth2 protokoly, grafické rozhranie pre rôzne nastavenia a iné.

Výhody

- *Open-source*
- Výber medzi autorizačnými protokolmi
- Jednoduchá inštalácia cez Docker
- Ľahká integrácia prihlásenia cez sociálne siete

Nevýhody

- Nie je garantovaná podpora, pretože je to *open-source*
- Náročná implementácia zložitejších *use-cases*
- Zbytočne objemný *open-source*, ak sa SSO nevyužíva
- Zložité prepojenie medzi existujúcimi tabuľkami v databáze

9.2.3 Java Spring Boot

Java Spring Framework (Spring Framework) je populárny *open source*, *enterprise-level framework* na vytváranie samostatných aplikácií, ktoré bežia na *Java Virtual Machine (JVM)*. *Java Spring Boot (Spring Boot)* je nástroj, ktorý urýchľuje a zjednodušuje vývoj webových aplikácií a mikroslužieb s rozhraním Spring Framework prostredníctvom troch základných funkcií:

- Autokonfigurácia
- Názorový prístup ku konfigurácii
- Schopnosť vytvárať samostatné aplikácie

Výhody

- Ľahký štart a vývoj
- Komunita
- Jednoduché testovanie
- Bez potreby XML konfigurácie

Nevýhody

- Slabá kontrola
 - Spring Boot vytvára veľa nepoužívaných závislostí, čo vedie k veľkému súboru
- Zložitý a časovo náročný proces konverzie
 - Napríklad konverzia Spring projektu na Spring Boot
- Nevhodné pre korporátne projekty
- Slabšia bezpečnosť
 - Pre lepšiu bezpečnosť a zminimalizovanie útokov je potrebné doimplementovať funkcionality na vylepšenie bezpečnosti

9.2.4 Výber serveru

Rozhodli sme sa pre **Spring boot server**, pretože pri porovnávaní vyšiel a pôsobil lepšie ako *CAS* alebo *Keycloak* server. *CAS* server sa z tejto trojice najťažšie implementoval, pretože inštrukcie pre implementáciu neboli úplne intuitívne. Pri *Keycloak*-u sme zistili, že sa síce jednoducho používa a implementuje, ale už hocijaká iná implementácia je časovo náročná, pretože *Keycloak* je po *open-source* stránke dosť objemný. Lenže jeho potenciál sa naplno využije iba ak ho využijeme na SSO. Spring boot sa nám ľahko implementoval aj nastavoval. Má super komunitu a ak nastane nejaký problém pri implementácii, v komunite sa už väčšinou tento problém vyriešil a preto ho ľahko nájdeme. Taktiež nevyžaduje XML konfiguráciu. Na ďalšiu stranu ak by sme chceli Spring projekt prekonvertovať na Spring boot projekt tak to by bolo taktiež časovo náročné a preto by sme mali začať už Spring boot projektom.

9.3 Certifikačné authority

V kryptografii je certifikačná autorita alebo certifikačná entita, ktorá vydáva digitálne certifikáty. Digitálny certifikát potvrdzuje vlastníctvo verejného kľúča pomenovaným subjektom certifikátu. To umožňuje ostatným stranám spoliehať sa na podpisy alebo na tvrdenia o súkromnom kľúči, ktorý zodpovedá certifikovanému verejnému kľúču. Každá certifikačná autorita vystupuje ako dôveryhodná tretia strana, ktorej dôveruje subjekt, teda vlastník certifikátu, ako aj strana, ktorá sa na certifikát spolieha. Formát týchto certifikátov je väčšinou určený štandardom X.509 Certifikačných autorít je viacero, no pre našu prácu sme si vybrali na porovnanie **Microsoft Active Directory** a **EJBCA**.

9.3.1 Microsoft Active Directory

Active Directory je adresárová služba vyvinutá spoločnosťou Microsoft pre siete domény Windows. Je súčasťou väčšiny operačných systémov Windows Server ako súbor procesov a služieb. Spočiatku sa služba Active Directory používala iba na centralizovanú správu domén. Active Directory sa však nakoniec stal zastrešujúcim názvom pre širokú škálu služieb súvisiacich s identitou založených na adresároch.

Server s rolou Active Directory Domain Service sa nazýva radič domény. Overuje a autorizuje všetkých používateľov a počítače v sieti typu domény Windows, priraduje a presadzuje zásady zabezpečenia pre všetky počítače a inštaluje alebo aktualizuje softvér. Napríklad, keď sa používateľ prihlási do počítača, ktorý je súčasťou domény Windows, Active Directory skontroluje odoslané používateľské meno a heslo a určí, či je používateľ správcom systému alebo bežným používateľom. Umožňuje tiež správu a ukladanie informácií, poskytuje autentifikačné a autorizačné mechanizmy a vytvára rámec na nasadenie ďalších súvisiacich služieb: Certifikačné služby, Active Directory Federation Services, Ľahké adresárové služby a Služby správy práv. Služba Active Directory používa protokol

LDAP verzie 2 a 3, verziu Kerberos od spoločnosti Microsoft a DNS.

Active Directory poskytuje viacero servisov:

1. *Domain Services*

- *Domain Services* sú základom každej doménovej siete Windows. Ukladajú informácie o členoch domény vrátane zariadení a používateľov, overujú sa ich prihlasovacie údaje a definujú sa ich prístupové práva. Server, na ktorom je spustená táto služba, sa nazýva *domain controller*. *Domain controller* je kontaktovaný, keď sa používateľ prihlási do zariadenia, prístupuje k inému zariadeniu cez sieť alebo spúšťa aplikáciu, ktorá je nainštalovaná v zariadení.

2. *Lightweight Directory Services*

- *Active Directory Lightweight Directory Services*, predtým známy ako *Active Directory Application Mode*, je implementáciou protokolu LDAP. Beží ako služba na serveri Windows.

3. *Certificate Services*

- *Active Directory Certificate Services* vytvára lokálnu infraštruktúru verejných kľúčov. Môže vytvárať, overovať a odvolávať sa na certifikáty verejného kľúča pre interné použitie organizácie. Tieto certifikáty možno použiť na šifrovanie súborov, e-mailov a sieťovej prevádzky (pri použití virtuálnych privátnych sietí, protokolu *Transport Layer Security* alebo protokolu IPsec).

4. *Federation Services*

- *Active Directory Federation Services* je služba s jedným prihlásením. So zavedenou infraštruktúrou môžu používatelia používať niekoľko webových služieb (napr. internetové fórum, blog, online *shopping*, webmail) alebo sieťové zdroje s použitím iba jednej sady poverení uložených na centrálnom mieste, na rozdiel od toho, že by museli byť udelené. vyhradenú sadu poverení pre každú službu. *Active Directory Federation Services* používa mnoho populárnych otvorených štandardov na odovzdávanie poverení tokenov, ako sú *SAML*, *OAuth* alebo *OpenID Connect*.

5. *Rights Management Services*

- *Active Directory Rights Management Services* je serverový softvér na správu práv k informáciám dodávaný so systémom Windows Server. Používa šifrovanie a formu selektívneho odmietnutia funkčnosti na obmedzenie prístupu k dokumentom, ako sú podnikové e-maily, dokumenty Microsoft Word a webové stránky, a operácie, ktoré s nimi môžu vykonávať oprávnení používatelia. Tieto operácie môžu zahŕňať napríklad prezeranie, úpravu, kopírovanie, ukladanie ako alebo tlač. Správcovia IT môžu v prípade potreby vytvárať prednastavené šablóny pre pohodlie koncového používateľa. Koncoví používatelia však stále môžu definovať, kto môže pristupovať k predmetnému obsahu, a nastaviť, čo môžu robiť.

Výhody

1. Centralizované ovládanie a monitorovanie
2. Bezproblémová používateľská skúsenosť

Nevýhody

1. Bezpečnostné riziká
2. Sieť sa stáva nadmerne závislou od AD
3. Môže byť drahé

9.3.2 EJBCA

EJBCA je softvér certifikačnej autority pre infraštruktúru verejného kľúča (PKI) slobodného softvéru, ktorý spravuje a sponzoruje švédsko-zisková spoločnosť PrimeKey Solutions AB, ktorá vlastní autorské práva na väčšinu kódovej základne. Zdrojový kód projektu je dostupný za podmienok Lesser GNU General Public License (LGPL). Softvér EJBCA sa používa na inštaláciu súkromne prevádzkovanvej certifikačnej autority. Toto je rozdiel od komerčných certifikačných autorít, ktoré prevádzkuje dôveryhodná tretia strana. Od svojho vzniku sa EJBCA používa ako softvér certifikačnej autority pre rôzne prípady použitia, vrátane elektronickej verejnej správy, správy koncových bodov, výskumu, energetiky, eIDAS, telekomunikácií, sietí a na použitie v malých a stredných podnikoch.

Výhody

1. Menšie bezpečnostné riziká
2. Vhodné pre menšie projekty
3. Bezplatný softvér

Nevýhody

1. Ťažšie používanie
2. Menej nástrojov pri vytváraní certifikátu

9.3.3 Výber najlepšej autority

Vzhľadom na vlastnosti vyššie popísaných certifikačných autorít sme sa do nášho projektu rozhodli implementovať EJBCA. Rozhodnutie padlo najmä vďaka flexibilitě, škálovateľnosti a pomerne jednoduchou integráciou do rozličných aplikačných prostredí, ktorými EJBCA disponuje. Taktiež sme sa snažili zvoliť si bezplatnú cestu. V prípade budúceho škálovania projektu EJBCA taktiež viac vyhovuje. Prostredníctvom management systému EJBCA sme schopní vitálne pridávať a odoberať rozličné druhy certifikácií.

10 Implementácia

V implementačnej časti si postupne popíšeme a ukážeme ako sa nám podarilo nasadiť výslednú architektúru nášho systému. Procesy prihlásenia, registrácie, autorizácie a autentifikácie boli upravené tak, že ak si klient zvolil viac faktorov na autentifikáciu, vedel by intuitívne pokračovať ďalej.

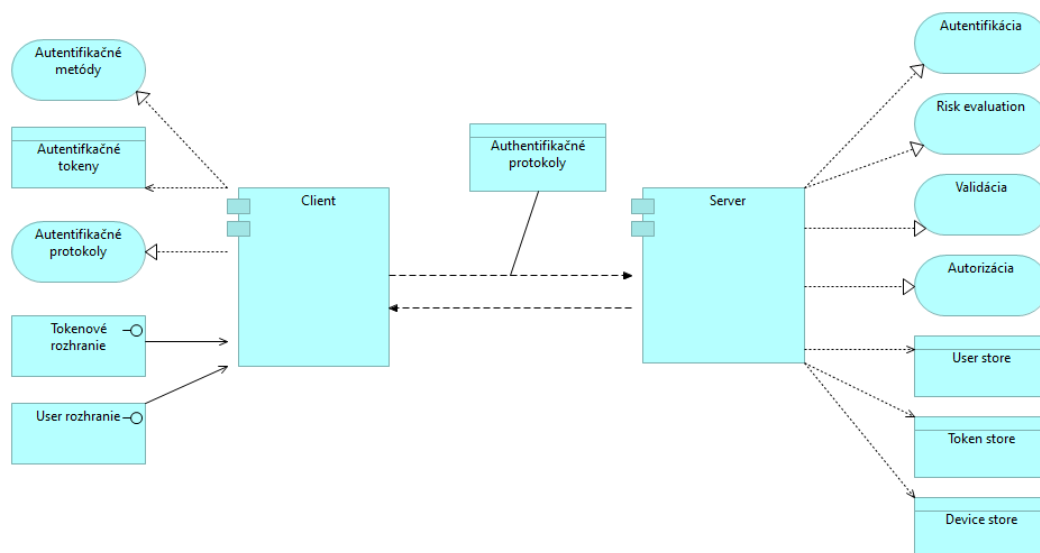
10.1 Výsledná architektúra

Architektúra vo svojej finálnej podobe pozostáva z:

- autentifikačnej a klientskej platformy
- risk serveru
- jednotlivých formulárov pre prihlásenie a registráciu do systému

10.1.1 Diagram pre *High-level* architektúru

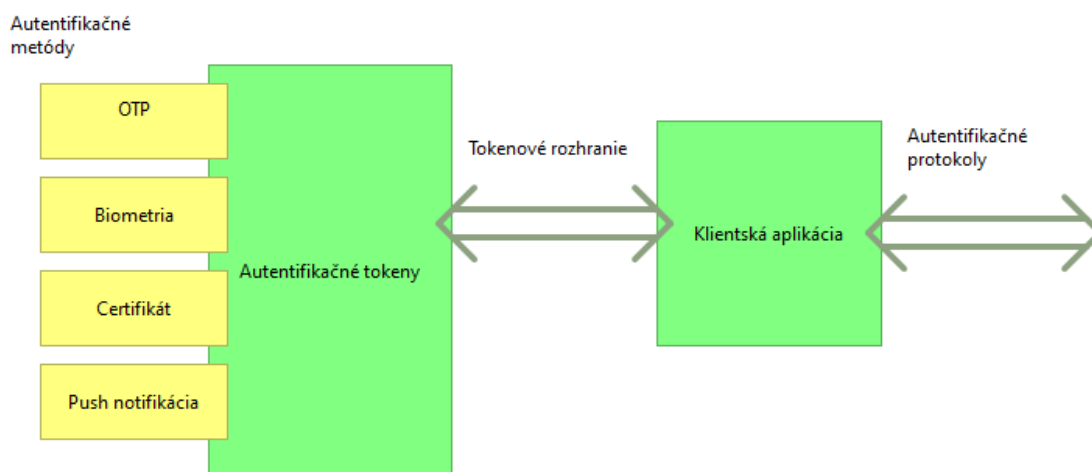
Oproti návrhu riešenia sme do architektúry ešte potrebovali pridať aj *device store*, ktorý drží informácie o klientskom zariadení, ktoré používateľ použil pri registrácii do systému. Ponechali sme *user a token store*, pretože autentifikačný server by nedokázal fungovať na takom autentifikačnom princípe, akom by sme pre finálny výsledok potrebovali. Zvyšné časti z návrhu sme ponechali, pretože sme počas implementácie zistili, že boli vhodne navrhnuté.



Obr. 26: Finálny vzhľad *high-level* architektúry

10.1.2 Diagram klientského rozhrania

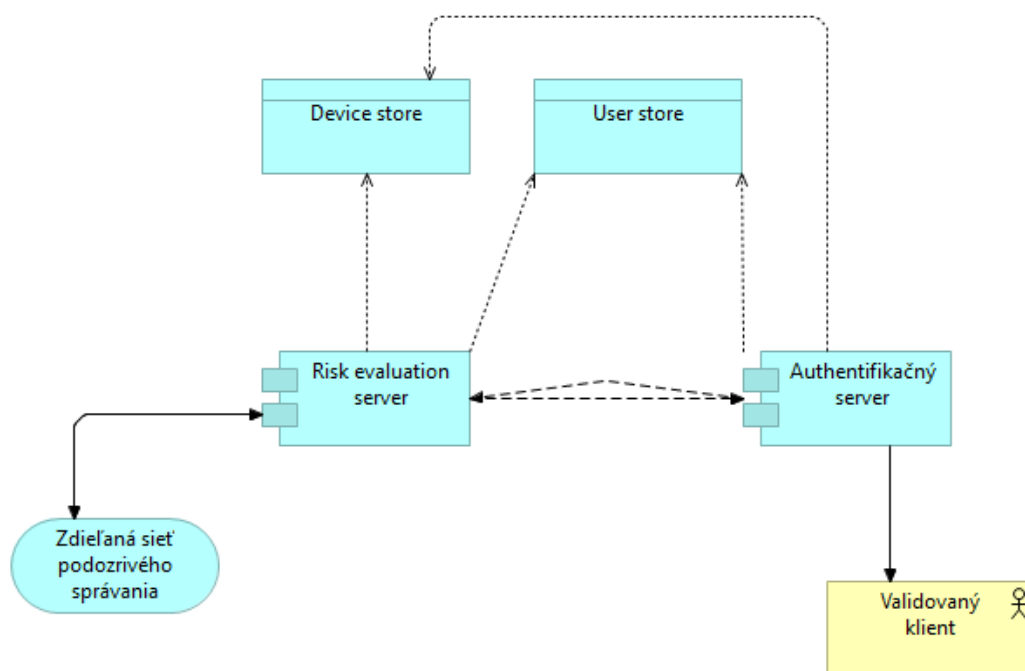
Pri rozhraní pre klienta sme doimplementovali certifikát cez certifikačnú autoritu a taktiež *push* notifikácie. Klient sa po úspešnej autentifikácii prihlási a získa certifikát, ktorého platnosť určuje certifikačná autorita, podľa toho ako je nastavená. *Push* notifikácie sme implementovali cez *Authy* klienta. Notifikácie predstavujú poslednú možnosť autentifikácie pre používateľa. Celkovo si však používateľ má možnosť vybrať z troch autentifikačných metód - *One Time Password*, Biometria a *Push* notifikácia



Obr. 27: Finálny vzhľad klientského rozhrania

10.1.3 Diagram pre manažment rizík

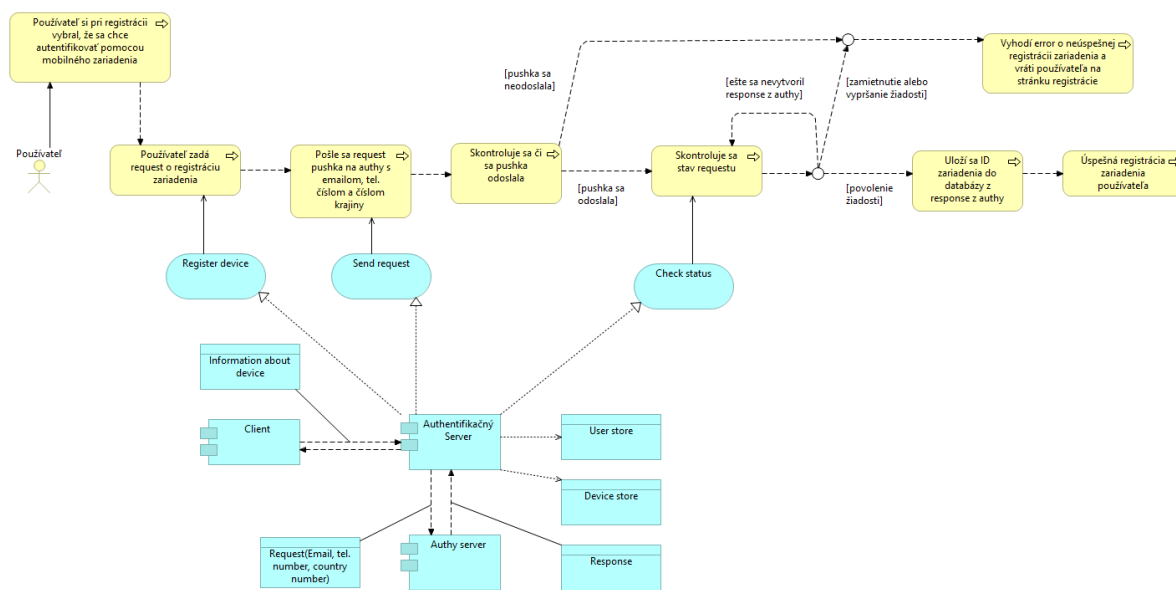
Na spravovanie a manažovanie rizík sme si implementovali *risk-evaluation server*, ktorý nám dané riziká pri prihlásení používateľa vyhodnocuje. Server komunikuje s autentifikačným serverom a taktiež aj s databázou používateľov a zariadení, aby vedel, ktoré zariadenie bolo napárované na používateľa. S databázou používateľov a zariadení musí taktiež komunikovať aj autentifikačný server, ktorý na základe informácii z týchto databáz, je neskôr schopný vyhodnotiť, teda validovať klienta. Zdieľaná sieť podozrivého správania vracia *risk-evaluation* serveru príznaky známeho podozrivého správania.



Obr. 28: Finálny vzhľad fungovania manažmentu rizík

10.1.4 Diagram pre device enrolment

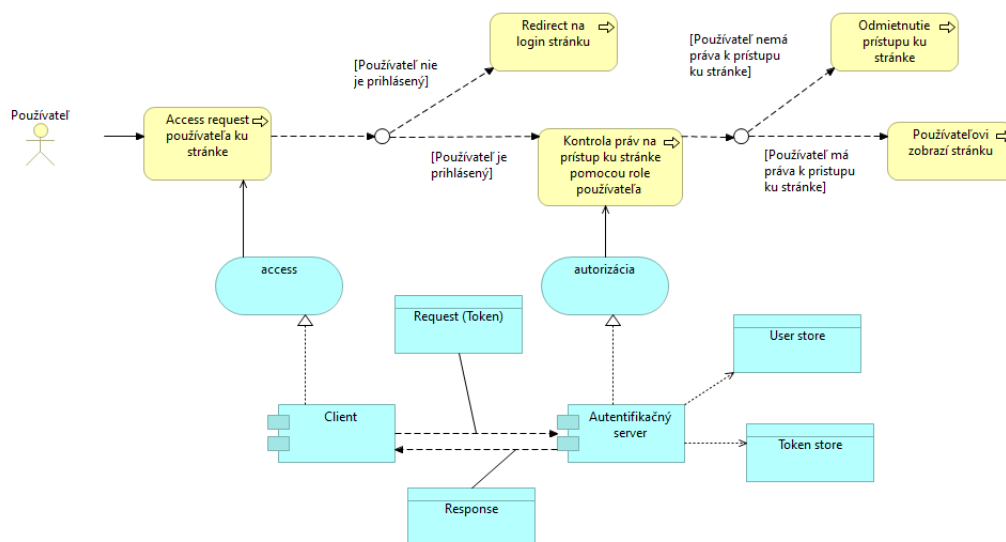
Diagram pre *device enrolment* sme pri návrhu riešenia neuviedli, pretože návrh riešenia nepočítal s rozsiahlejšou registráciou používateľa a jeho zariadenia. Pri implementácii sme však zistili, že by bolo vhodné mať všetky informácie o používateľovi ešte predtým, ako mu schválime celú registráciu a bude schopný sa autentifikovať. Diagram v skratke ukazuje to, že ak si používateľ pri registrácii vyberie, že sa chce autentifikovať pomocou mobilného zariadenia, tak na náš server príde požiadavka o registráciu jeho zariadenia do tabuľky zariadení. Neskôr bude používateľ vyzvaný, aby si nainštaloval aplikáciu *Authy*, cez ktorú sa bude aj pri ďalších prihláseniach autentifikovať. Aplikácia *Authy* nám vráti odozvu, kde vidíme mailovú adresu používateľa, jeho číslo a taktiež jeho číslo krajiny. Používateľ si nastaví či sa chce autentifikovať pomocou *push* notifikácie alebo pomocou OTP, teda jednorazového kódu. Ďalšie procesy, ako kontrola správneho odoslania *push* notifikácie alebo správneho jednorazového kódu sa kontrolujú na pozadí. Všetky ostatné procesy riadi náš autentifikačný server, ktorý komunikuje so všetkými potrebnými koncovými bodmi, ako napríklad databázou, *emphAuthy* serverom a klientom. Medzi klientom a serverom sa vymieňajú informácie o zariadení používateľa. Medzi *Authy* serverom a autentifikačným serverom sa zase posielajú informácie ako mail, číslo krajiny.



Obr. 29: *Device enrolment* počas registrácie

10.1.5 Diagram autorizácie

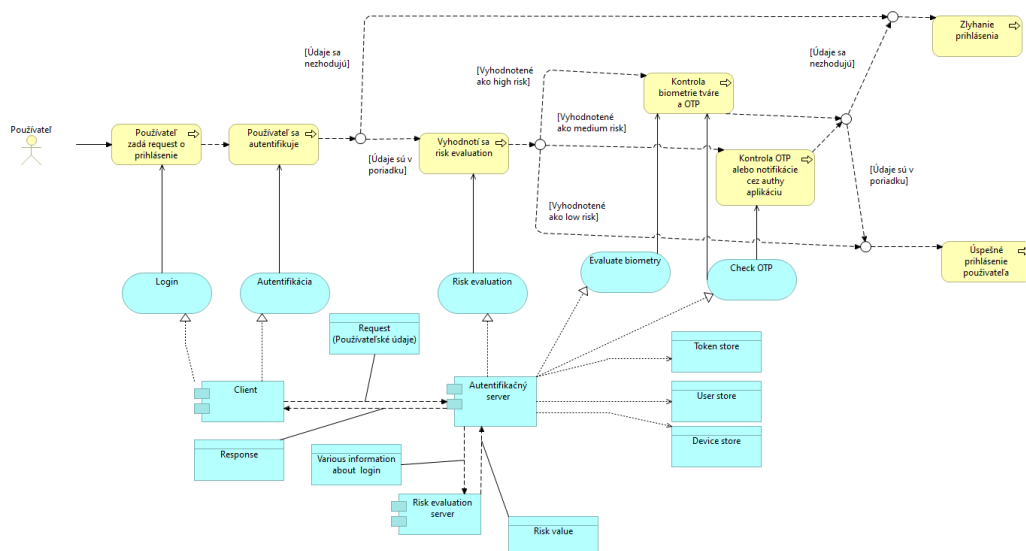
Pri diagrame pre autorizáciu sme urobili pár zmien, hlavne, čo sa týka systémových rol. Zatiaľ, čo v návrhu riešenia sme systémové roly preskočili a nezaoberali sa nimi, teraz sme potrebovali celý autorizačný proces dokončiť tak, aby sme po prihlásení vedeli identifikovať administrátora a bežného používateľa. Ak používateľ nie je prihlásený, presmeruje ho na stránku pre prihlásenie. Ak však sa už v daný deň používateľ prihlásil, najprv sa na pozadí skontrolujú jeho oprávnenia na základe role, ktorú má a až potom ho presmeruje na správnu podstránku. Ak by sa obyčajný používateľ snažil prihlásiť na stránku administrátora, prístup by mu bol zamietnutý. Autentifikačný server na pozadí pracuje aj s tokenom, ktorý je danému používateľovi pridelený a, aj na základe neho prebehne proces autorizácie.



Obr. 30: Finálny vzhľad *high-level* procesu autorizácie

10.1.6 Diagram prihlásenia

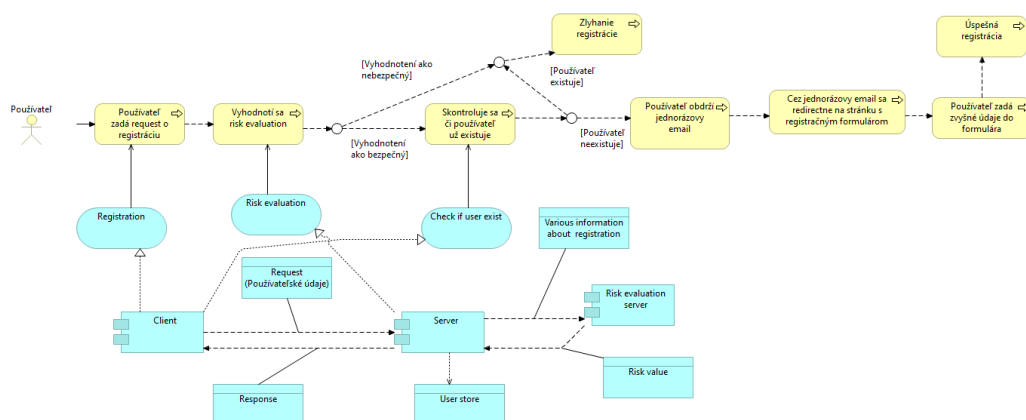
Diagram prihlásenia bol doplnený o viacero vecí. Ak sa používateľ vyhodnotí, že predstavuje nízke riziko, bude mu stačiť sa prihlásiť mailovou adresou a heslom. A však, ak sa na našom *risk-server-i* vyhodnotí prihlásenie používateľa ako rizikové, tak podľa stupňa rizika budú pribúdať faktory pre autentifikáciu. Okrem hesla sa bude musieť používateľ autentifikovať cez aplikáciu *Authy* alebo neskôr aj pomocou tváre. V najjednoduchšom prípade, ak by sa prihlasovacie údaje nezhodovali, prihlásenie by bolo neúspešné. Počas vyhodnocovania rizika pri prihlasovaní sa do systému, autentifikačný server komunikuje s *risk-server-om*, ktorý odošle späť informácie o používateľovi. Podľa jednotlivých informácií (krajina, IP adresa, verzia prehliadača, verzia operačného systému) sa vyhodnotí aj riziko pri prihlasovaní. Ak používateľ prejde všetkými kontrolnými mechanizmami, úspešne sa prihlási do systému.



Obr. 31: Finálny vzhľad *high-level* procesu prihlásenia

10.1.7 Diagram registrácie

Diagram registrácie je pomerne jednoduchý a preto sa registrácia aj jednoducho implementovala. Používateľ už pri samotnej registrácii, ku ktorej musel najprv získať prístup cez jednorazovú *url*, zadá požiadavku o registráciu po vyplnení registračného formulára. *Risk-server* na pozadí vyhodnotí riziko. Ak sa riziko vyhodnotí ako dostatočne veľké, registrácia zlyhá. V opačnom prípade sa používateľ vyhodnotí ako bezpečný alebo priateľný voči riziku a registrácia prebehne úspešne.



Obr. 32: Finálny vzhľad *high-level* procesu registrácie

10.2 Viac-faktorová autentifikácia

Pri viac-faktorovej autentifikácii je potrebné spomenúť všetky faktory, ktoré boli implementované pre zaistenie plnohodnotnej autentifikácie. Na celý proces autentifikácie využívame aplikáciu od spoločnosti *Twilio - Authy*.





10.2.1 Authy aplikácia

Vďaka aplikácii *Authy* a jej podpore pre viacero zariadení sa tokeny viac-faktorovej autentifikácie automaticky synchronizujú s akýmkoľvek novým zariadením, ktoré autorizujete. A ak dôjde k strate, krádeži alebo vyradeniu zariadenia, je možné rovnako rýchlo zrušiť jeho autorizáciu na akomkoľvek autorizovanom zariadení. Keďže *Authy* je k dispozícii pre mobilné zariadenia so systémom *Android* a *iOS*, ako aj pre *Windows*, *Apple Watch* a dokonca, aj pre každý iný počítač, môžete pomocou *Authy* zostať chránení pred všetkými zariadeniami súčasne.

10.2.2 Authy One Time Password

Keď sa používateľ zaregistruje v aplikácii *Authy* a dostane *AuthyID*, je možné implementovať viac-faktorovú autentifikáciu. Pomocou *Authy API* môžeme poslať jednorazové heslá cez hlasové alebo SMS kanály. *Authy API* sa používa na overenie, či má používateľ prístup k správne telefónnemu číslu (pre hlasové alebo SMS kanály) alebo má prístup k správne dôveryhodnému zariadeniu. Keď by sme zavolali *API* na spustenie SMS alebo hlasovej autentifikácie, automaticky sa skontroluje, či si daný používateľ už predtým stiahol aplikáciu *Authy* alebo či má nainštalovanú aplikáciu, ktorá používa súpravu SDK od *Authy*. Ak má používateľ aplikáciu, rozhranie *API* štandardne nepošle kód pre viac-faktorovú autentifikáciu prostredníctvom SMS alebo hlasu. Namiesto toho sa do zariadenia dostane upozornenie nazývané aj ako *push notification*, ktoré používateľa vyzve, aby spustil aplikáciu, aby získal kód. Ak používateľ nemá žiadny záznam o inštalácii zariadenia, rozhranie *API* bude naďalej odosielať kód prostredníctvom SMS alebo hlasu. Predvolené správanie vieme prepísať a zakaždým vynútiť odoslanie kódu prostredníctvom SMS alebo hlasu. Prepísanie je užitočné, ak používateľ v používateľskom rozhraní aplikácie konkrétne vyberá možnosť „Odoslať SMS“ alebo „Získať kód prostredníctvom hlasového hovoru“. Na obrázkoch nižšie uvidíme aké parametre URL po zaslaní požiadavky na *Authy API* potrebuje a taktiež akú odozvu vráti späť.

Parameters

Name	Description
<code>force</code> Boolean (optional)	Default is <code>false</code> . Set to <code>true</code> to send one-time passwords over the SMS channel even if the user has an Authy or SDK enabled app installed. Configure default behavior in the console. ( not PII)
<code>action</code> String (optional)	The optional action or context we are trying to validate. ( not PII)
<code>action_message</code> String (optional)	Optional message for the specific action. ( not PII)
<code>locale</code> String (optional)	The language of the message received by user. If no locale is given, Twilio will try to autodetect it based on the country code. English is used if no locale is autodetected. More details below ( not PII)

Obr. 33: Parametre URL

- Možné parametre
 - **force** - Predvolená hodnota je *false*. Ak by sme chceli odosielať jednorazové heslá cez SMS kanál, nastavili by sme na hodnotu *true*,
 - **action** - Voliteľná akcia alebo kontext, ktorý sa snažíme overiť,
 - **action_message** - Voliteľná správa pre konkrétnu akciu,
 - **locale** - Predstavuje jazyk správy prijatej používateľom. Ak nie je zadané žiadne lokálne nastavenie, *Twilio* sa ho pokúsi automaticky zistiť na základe kódu krajiny.

Response

Name	Description
success Boolean	Returns true if the request was successful. (🚫 not PII)
message String	A message indicating what happened with the request. (🚫 not PII)
cellphone String	The country code and last two digits of phone number used to send the message with the rest obfuscated. (🚫 not PII)
ignored Boolean	True if we detected an Authy or SDK enabled app installed and we upgraded the OTP delivery channel from 'SMS' to Push Notification. Authy or SDK users are redirected directly to the requested token. (🚫 not PII)
device String	The type of the last device used by the user. This is only returned when we upgraded delivery from SMS. (🚫 not PII)

Obr. 34: Možné odozvy po poslaní požiadavky

- Možné odozvy
 - **success** - Ak bola požiadavka úspešná, vráti hodnotu *true*,
 - **message** - Správa označujúca, čo sa stalo so žiadosťou,
 - **cellphone** - Kód krajiny a posledné dve číslice telefónneho čísla použité na odoslanie správy,
 - **ignored** - Pravda, ak sme zistili, že je nainštalovaná aplikácia s podporou Authy alebo SDK a inovovali sme kanál doručovania OTP z „SMS“ na Push Notification. Používatelia Authy alebo SDK sú presmerovaní priamo na požadovaný token,
 - **device** - Typ posledného zariadenia používaného používateľom. Toto sa vráti iba vtedy, keď sme upgradovali doručovanie z SMS.

10.2.3 Authy push notifikácie

Pri *Authy* push notifikáciach je potrebné naformátovať správne požiadavku cez metódu POST.

URL požiadavka musí mať nasledovný formát, ktorý je definovaný nasledovne:

- **FORMAT**

- typ: String
- klasický REST API callback cez JSON alebo XML

- **AUTHY_ID**

- typ: Integer
- predstavuje Authy ID daného používateľa, aby bola *push* notifikácia zaslaná správne

Možné dodatočné parametre:

- **message**

- zobrazí sa používateľovi, keď príde *push* notifikácia do telefónu

- **details**

- slovník obsahujúci každý *approval request* detail, ktorý by sme chceli ukázať používateľovi, aby sa vedel rozhodnúť pre schválenie alebo zamietnutie notifikácie

- **hidden_details**

- slovník obsahujúci podrobnosti žiadosti o schválenie, ktoré je skryté pre používateľa.

- **logos**

- slovník obsahujúci prepisovacie logá, ktoré sa používateľovi zobrazia v detailoch transakcie push autentifikácie

- **seconds_to_expire**

- počet sekúnd, počas ktorých je transakcia platná bez odozvy používateľa pred uplynutím platnosti.

Parameters

Name	Description
<code>message</code> String	Shown to the user when the push notification arrives. (🔒 PII)
<code>details</code> Hash (optional) (Max 20 characters for the Key in the key value pair)	Dictionary containing any <code>ApprovalRequest</code> details you'd like to present to the user to assist their decision to approve or deny a transaction. We automatically add a timestamp to transactions. See below for an example on how to use <code>details</code> . (🔒 PII)
<code>hidden_details</code> Hash (optional) (Max 20 characters for the Key in the key value pair)	Dictionary containing the approval request details hidden to user. This information will be preserved in transaction records but not presented to the user, so it may be useful for your business logic and routing. (🔒 PII)
<code>logos</code> Hash (optional)	A dictionary containing override logos that will be shown to user in the push authentication transaction details. By default, we send the logos uploaded through the console. (🔒 not PII)
<code>seconds_to_expire</code> Integer (optional)	The number of seconds a transaction is valid without user response (pending) before expiring. Defaults to <code>86400</code> (one day); <code>0</code> will never expire. This should not be set too low as users need time to evaluate a request. (🔒 not PII)

Obr. 35: Dodatočné parametre pre URL

Možné odozvy:

- **approval_request**
 - *Hash* obsahujúci kľúče a hodnoty pre žiadosť o schválenie
- **uuid**
 - Jedinečné ID žiadosti o schválenie
- **created_at**
 - Dátum a čas, kedy sa vytvorila žiadosť o schválenie
- **status**
 - Sleduje aktuálny stav žiadosti o schválenie medzi čakajúcou na odpoveď používateľa, schválením, zamietnutím alebo vypršanou platnosťou

Response

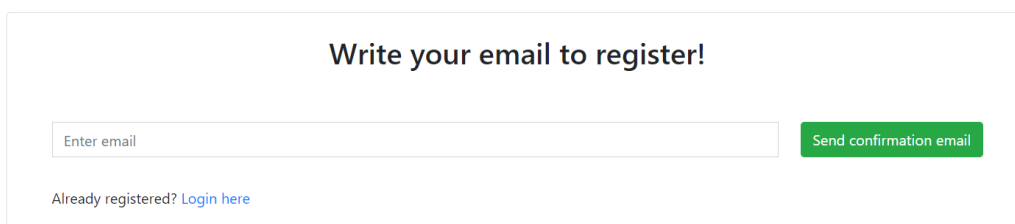
Name	Description
<code>approval_request</code> Hash	Hash containing the keys & values for the <code>ApprovalRequest</code> . (🔒 PII)
<code>uuid</code> String	Unique transaction ID of the <code>ApprovalRequest</code> . You'll need the <code>uuid</code> to query the request status or tie future callbacks to this <code>ApprovalRequest</code> . (🔒 not PII)
<code>created_at</code> Datetime	The date and time that we created the <code>ApprovalRequest</code> . (🔒 not PII)
<code>status</code> String	Tracks the current state of the <code>ApprovalRequest</code> between <code>pending</code> a user response, <code>approved</code> , <code>denied</code> , or <code>expired</code> . (🔒 not PII)

Obr. 36: Dodatočné parametre pre URL

10.3 Registrácia

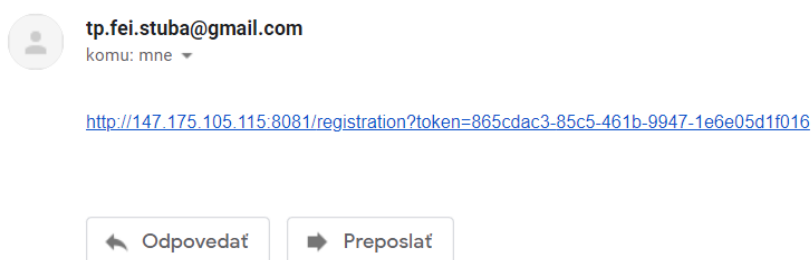
Pre účely evidencie nových užívateľov aplikácie sme vytvorili registračný systém využívajúci jednorazovú URL a jedinečný token. Systém prebieha štandardne v nasledujúcich krokoch:

- Užívateľ si zvolí možnosť registrácie na webovej stránke
- Následne napíše svoj email a potvrdí jeho odovzdanie



Obr. 37: Prvý krok pri registrácii nového užívateľa

- Na zadanú emailovú adresu je mu odoslaná jednorazová URL



Obr. 38: Jednorazová URL odoslaná na email registrujúceho sa užívateľa

- Po kliknutí na adresu je presmerovaný na registračný formulár. Vo formuláry spracovávame informácie ako krstné meno, priezvisko, email a telefónne číslo. Všetky z týchto údajov sú povinné okrem telefónneho čísla. Užívateľ má priamo pri registrácii voľbu dvojstupňovej autentifikácie alebo face recognition, teda prepojenia tváre užívateľa s jeho účtom.
- Korektné vyplnenie a odovzdanie formulára vedie k úspešnej registrácii užívateľa, ktorý sa môže prihlasovať do našej aplikácie

First Name

Last Name

Email

ezzekieqt@gmail.com

Phone number

 370 ▼

Password

Register face

Do you want to use 2FA?

No



Register

Already registered? [Login here](#)

Obr. 39: Registračný formulár

10.3.1 Implementácia

Samotný proces implementácie registrácie začína pri vytvorení dočasného usera v triede *TemporaryUser*. Účel vytvorenia takéhoto usera je z dôvodu implementovaného odoslania dočasnej a jednorazovej URL, kde ešte usera neregistrujeme do našej databázy. Proces pokračuje širokým zastúpením funkcií v triede *UserRegistrationController()*, ktorej cieľom je poskytnúť funkcionality registrácie pre nového používateľa. Nachádza sa tu metóda *showRegistrationForm()*, ktorá zobrazuje userovi registračný formulár, prípadne vráti *badToken* error, ak sa stane, že vypršal *verification token*.

Taktiež sa tu nachádza metóda *registerUserAccount()*, ktorej úloha je korekcia vstupov od usera a následná registrácia, teda uloženie registrovaného do databázy. Funkcia *registerGaAccount()* má za úlohu zaregistrovať usera, ktorý si zvolil možnosť 2FA, teda registrácia navyše ponúkne nastavenie *Authy app*, pre daný user account.

10.3.2 Vytvorenie tokenu

Samotné vytvorenie tokenu prebieha v triede *UserServiceImpl* v metóde *createVerificationToken()*. Metóda pre vytvorenie vyžaduje 2 argumenty. Jedným z nich je objekt *temporaryUser* a druhý je náhodne vygenerovaný string, z ktorého je token následne vytvorený. Metódu *createVerificationToken* voláme v triede *RegistrationListener*.

10.3.3 Overenie tokenu

Overenie tokenu nám umožňuje vytvárať vyššie spomenuté adresy URL, ktoré majú limitovanú životnosť. Tokeny sa generujú na našej webovej aplikácii a pripájajú sa k vygenerovanej URL adrese pre registráciu. Týmto prístupom zabezpečujeme jedinečný prístup k registrácii pre daného užívateľa.

10.3.4 Jednorazová URL

Adresa URL je vytvorená v triede *RegistrationListener* v metóde *confirmRegistration()*, kde dochádza k pripojeniu stringu vygenerovaného tokenu k pôvodnej URL. Tým docieľujeme jedinečnosť URL adresy pre registrovaného usera. Dočasná adresa zanikne hneď po jej použití alebo po uplynutí stanoveného časového obdobia, ktoré sme nastavili na 15 minút. Tento prístup má veľmi široké zastúpenie pri tvorbe externých autentifikačných systémov, a to je aj jeden z dôvodov prečo sme si ho vybrali.

Implementácia Je realizovaná prostredníctvom triedy *VerificationToken*. V triede sú je implementovaný konštruktor, ktorý tokenu automaticky nastavuje životnosť na 15 minút. Taktiež sa tu nachádza metóda *calculateExpiryDate()*, ktorá dokáže vrátiť zostávajúci čas životnosti pre vygenerovaný token. Trieda samozrejme obsahuje všetky potrebné *getter*y a *setter*y.

Token je využívaný v triede *UserRegistrationController()*, kde kontrolujeme, či sa *verification token* nerovná null, teda či nevypršal. Na základe tejto informácie dovoľujeme užívateľovi sa registrovať.

10.3.5 Verifikácia vstupov

Pri registrácií sme taktiež zabezpečili overovanie vstupov od užívateľa. Je to najmä z dôvodu dostatočne bezpečného hesla, jednoduchosti údajov v databáze a taktiež ako prevencia voči útokom na stránku ako sú napríklad XSS či SQL injection útoky.

Jednotlivé obmedzenia pre vstupy sme zvolili nasledovne:

- Nový registrovaný email nesmie existovať
- Krstné meno a priezvisko musia mať dĺžku aspoň 2 znaky a najviac 30 znakov
- Emailová adresa musí spĺňať štandardný formát pre emailové adresy
- Heslo musí obsahovať minimálne 8 znakov, 1 veľké písmeno, 1 malé písmeno, 1 číslo a jeden špeciálny znak
- Telefónne číslo musí obsahovať 10 čísiel

Implementácia Verifikácia je implementovaná v triede *UserRegistrationController* v metóde *showRegistrationForm()* a taktiež v triede *User*, kde kontrolujeme pomocou anotácií, ktoré atribúty musia byť vyplnené, resp. sú povinné pri registrácií. Verifikácia prebieha pomocou overení bežných dĺžok inputov od usera a taktiež pomocou *regexu* pre overenie emailu a taktiež pre overenie hesla.

10.4 Prihlásenie

Po úspešnej registrácii sa pre užívateľa otvorí možnosť prihlásenia sa. Pre úspešné prihlásenie užívateľ musí zadať korektné údaje, ktoré si nastavoval pri registrácii.

User Login Page

Username :

Password:

Log In

Authenticate via face
recognition

New user? [Register here](#)

Obr. 40: Prihlasovanie užívateľa

Po úspešnom prihlásení a verifikácii všetkých nastavených dvojstupňových autentifikácií je užívateľ autorizovaný pre vstup do aplikácie.

You have been logged successfully!

Welcome ezekeiqt@gmail.com

Obr. 41: Úspech prihlásenia

10.4.1 Postup implementácie

Prebieha v triedach *AuthyLoginController* a *MainController*, kde máme implementované metódy ako *login()* alebo *loginUser*. Prihlásenie prebieha overením a nájdením userovho mailu v našej databáze a následným overením správneho hesla. Ak má používateľ pridanú dvojfaktorovú autentifikáciu, tak je v metóde *loginUser* aj vygenerovaný response, ktorý odošle kód na userov mobil, kde musí potvrdiť prihlásenie. Zároveň je v tejto metóde implementované aj overenie odoslaného *secretCode*, teda kódu, ktorý musí užívateľ napísať do aplikácie. Ak je overenie v poriadku, užívateľ je úspešne prihlásený.

10.4.2 Druhy prihlásenia

V rámci 2FA má užívateľ možnosť vybrať si dvojfaktorové overenia ako je push notifikácia, jednorázové heslo cez *Authy* alebo *face recognition*, teda autentifikácia vykonaná prostredníctvom verifikácie tváre užívateľa.

10.5 Face recognition

Face Recognition modul pozostáva z 2 hlavných komponentov:

- **Frontend**

- obsluhuje vyhotovenie fotiek pre spracovanie na registráciu/autentifikáciu

- **Backend**

- obsluhuje model neurónovej siete FaceNet pre registráciu/autentifikáciu a poskytuje informácie o stave pre Spring Boot aplikáciu

10.5.1 Frontend stack

Využíva sa Vue 2, ktorý je *javascript framework* pre tvorbu webových aplikácií poskytujúci užívateľovi rozhranie pre registráciu a autentifikáciu pomocou Face Recognition.

10.5.2 Frontend metódy pre registráciu a autentifikáciu

Podľa url parametra *mode* (1/2) sa rozhranie rozhoduje čo za akciu bude vykonávať a v rámci toho používa 2 metódy: **recordImage** a **downloadImage**. Okrem toho je tu generická **takePhoto** metóda, ktorá transformuje snímok z kamery do tvaru *dataURL*, ktorý dokážeme poslať backendu na vyhodnotenie.

Obidve metódy *recordImage* a *downloadImage* sú založené na tom že pomocou *AJAX* komunikácie odošlú dáta na relevantný backend endpoint a očakávajú ďalej odpoveď o stave spracovania snímok na zobrazenie.

10.5.3 Backend stack

Využívame dve nasledovné technológie:

- **Flask**
 - webový *framework* v Pythone pre vytvorenie endpointov na komunikáciu s Frontendom a so Spring Boot aplikáciou
- **Keras**
 - je softvérová knižnica typu *open-source*, ktorá poskytuje rozhranie Python pre umelé neurónové siete, v našom prípade obsluhuje FaceNet pre trénovanie alebo samotný *recognition* proces

10.5.4 Backend metódy pre registráciu a autentifikáciu

V rámci backendu sú relevantné dva servisy: `app` a `recognition_handler`. `App` je klasický Flask modul pre definovanie endpointov a `recognition_handler` ovláda neurónovú sieť FaceNet. Pre proces registrácie slúži POST endpoint `register_user`, ktorý očakáva snímok v base64 dataURL tvare, ktorý si spracuje a v rýchlosti skontroluje či sa na snímke nachádza tvár a uloží do priečnika prislúchajúceho pre používateľa. Takto spracuje 20 snímokov (optimálny počet z prieskumu parametrov) a z nich vyextrahuje face embeddingy vo veľkosti 160x160 pixelov, ktoré už sú vhodné na vstup do siete pre trénovanie. Pre proces autentifikácie slúži POST endpoint `recognize_user` ktorý očakáva snímok v base64 dataURL tvare, ktorý si spracuje a pošle ďalej do `main_recognition_handler`. Ten už spúšťa proces rozpoznania tváre pomocou FaceNetu, ktorý nám vráti pravdepodobnosť, že je na fotke užívateľ. Ak je autentifikácia úspešná tak vytvorí “token” vo forme json súboru, na ktorý sa bude dopytovať Spring Boot aplikácia pri overení tohto faktoru.

Pre samotnú komunikáciu so Spring Boot aplikáciou slúžia opäť 2 GET endpointy: `check_registration` a `check_authentication`. Oba očakávajú ako parameter `username` meno používateľa, o ktorého sa jedná. Funkcionalita `check_registration` spočíva v tom, že kontroluje či existuje priečinok pre `username` a, či má dostatok snímok. Tieto kritéria predstavujú hotovú registráciu na strane Face Recognition. Funkcionalita `check_authentication` spočíva v tom, že hľadá aktívny token v priečinku užívateľa a pokiaľ taký nájde tak ho zmaže a potvrdí autentifikáciu. Vo všetkých iných prípadoch vracia neautentifikovaný stav.

10.6 Systémové roly

Implementovali sme systém s rolami alebo aj tzv. *role-based* systém, na základe ktorého má používateľ s danou rolou prístup na určité stránky. Celkovo budeme pracovať s tromi rolami, ale jedna rola je *hybridná* resp. dynamická. Rola sa najprv volá *pre-user*, pretože takto si vieme sledovať, či už daný používateľ je validný a súčasťou celého systému. Ak má spomínanú rolu, tak ešte nie je a až keď celý registračný proces prebehne úspešne, rola sa zmení na *user*. Posledná rola, ktorú sme implementovali bola *admin* rola. Už z názvu vieme vyčítať, že používateľ s admin rolou bude môcť pristupovať na všetky podstránky.

10.6.1 Vytvorenie používateľských rolí

Implementovali sme pomocnú metódu ***createRoleIfNotFound***, ktorá vytvorí roly *ROLE_PRE_USER* a *ROLE_ADMIN*, ak ešte neexistujú. Každú rolu vytvárame cez novú inštanciu triedy *Role* a potom ukladáme do rozhrania *RoleRepository*. Metóda ***createRoleIfNotFound*** nakoniec vráti vytvorenú rolu.

Metóda ***postConstruct*** bola implementovaná tak, aby vytvorila iba jedného *root* admina, na konkrétny mail. Taktiež ešte vytvorí dve nové roly a to *ROLE_ADMIN* a *ROLE_PRE_USER*. Vo zvyšku tela metódy sa už len vytvorí nová inštancia triedy *User* a nastaví sa základné vlastnosti ako e-mail, prvé a posledné meno, telefónne číslo a iné vlastnosti. Nakoniec sa nový používateľ uloží do rozhrania *UserRepository*.

Metódu ***save*** sme implementovali konkrétne pre potenciálnych používateľov, ktorí dostanú na začiatku *ROLE_PRE_USER*. Celá metóda pracuje takmer identicky ako ***postConstruct*** až na to, že už sa pracuje s údajmi z registračného formuláru.

10.7 Bezpečnostná konfigurácia

Pre účely bezpečnosti využívame viacero konfiguračných systémov. Tieto systémy sa nachádzajú v triede *SecurityConfiguration()*.

Pri konfigurácii taktiež využívame metódu ***configure()***, ktorá je napojená na risk server a vyvoláva odozvy risk servera vzhľadom na neoprávnené aktivity na serveri. Metóda taktiež slúži k zamedzeniu prístupu užívateľa k potenciálne citlivým priečinkom uložených na serveri. Pre tento účel v metóde využívame ***antMatchers()***, ktoré obmedzujú manipuláciu s URL adresou.

antMatchers() je jedinečná metóda pre Spring Security, ktorá slúži na manažovanie URL adries, ku ktorým môžu pristupovať užívatelia na základe ich priradených rolí v systéme.

10.7.1 Risk server

Jednou z väčších implementácií bol ***risk-evaluation server***, ktorý vyhodnocuje riziko na základe nejakých údajov, ktoré v sebe nesie celá štruktúra daného používateľa. Údaje, podľa ktorých vyhodnocuje server riziko sú nasledovné:

- IP adresa
- prehliadač a jeho verzia
- operačný systém a jeho verzia
- krajina

Risk-server dokáže vrátiť 4 rôzne hodnoty, kde každá predstavuje iné riziko.

- **Najväčšie riziko - 4**
 - používateľ sa nedokáže prihlásiť a je umiestnený na *blacklist*,
- **Stredné riziko - 3**
 - používateľ sa prihlási, ale bude sa musieť autentifikovať, čiže nie len heslom, ale aj cez aplikáciu *Authy* a nakoniec aj cez tvárovú biometriu,
- **Malé riziko - 2**
 - používateľ predstavuje v tomto scenári malé riziko a preto sa mu stačí okrem hesla autentifikovať iba cez jeden ľubovoľný faktor,
- **Takmer žiadne riziko - 1**
 - používateľ sa prihlási iba s heslom, pretože predstavuje takmer nulové riziko a automaticky dostane aj rolu *ROLE_USER*

11 Prínos

Benefity externých autentifikačných serverov majú v praxi široké zastúpenie. Medzi hlavné výhody patrí šetrenie času a peňazí pre firmy, ktoré nemusia implementovať pre tento účel vlastné systémy. Stačí ak využijú iné externé autentifikačné servery a automaticky majú splnený celý prípad použitia pre prihlasovanie, autentifikáciu, správu a autorizáciu rôznych užívateľov v systéme. Veľkou výhodou takýchto systémov je taktiež jednoduché napojenie na externú aplikáciu, ktorá dokáže služby autentifikačného servera využívať.

Z hľadiska bezpečnosti sú externé autentifikačné servery taktiež výhodné, a to najmä pre klientské aplikácie. Je to z dôvodu šetrenia prostriedkov, pretože o bezpečnostné opatrenia pre ochranu hesiel a údajov sa stará samotný autentifikačný server. Mnohokrát môže byť implementovaný prostredníctvom cloudových centralizovaných serverov, kde je bezpečnosť na veľmi vysokej úrovni. Naša aplikácia taktiež spĺňa množstvo z týchto aspektov externých autentifikačných serverov a to z hľadiska bezpečnosti a aj škálovateľnosti.

Záver

V práci sme si vybrali framework Spring Boot. Túto technológiu sme si z počiatku vybrali najmä kvôli širokému zastúpeniu funkcionality a komunity, ktorá sa problémami Spring Bootu zaoberá, takže je jednoduchšie nájsť riešenia. Počas práce na projekte sa framework javil ako implementačne mierne náročnejší, no fundamentálne nám pokryl všetky potrebné body pre vyhotovenie našej aplikácie. Podarilo sa nám implementovať plne funkčnú aplikáciu pre registráciu, prihlasovanie a viacfaktorovú autentifikáciu užívateľov. Aplikácia je zabezpečená viacerými bezpečnostnými opatreniami a taktiež risk management systémom. Počas implementácie jednotlivých komponentov potrebných pre vyhotovenie nášho projektu sa nám úspešne podarilo implementovať dva typy dvojfaktorovej autentifikácie. Jedným z nich je face recognition a druhým je push notifikácia pre autentifikáciu prostredníctvom mobilného zariadenia. Uvažovali sme aj nad inými možnosťami pre dvojfaktorovú autentifikáciu ako napríklad overenie QR kódom, avšak túto časť sme nakoniec neimplementovali.

Zameranie sme upriamili najmä na dvojfaktorovú autentifikáciu miesto multifaktorovej autentifikácie. Naša aplikácia nevyžaduje od používateľa povinné nastavenie viacerých faktorov autentifikácie, ale je dobrovoľná. Používateľ však ja napriek tomu je schopný zaobstarat si multifaktorovú autentifikáciu a to v prípade, že si zvolí obe možnosti dvojfaktorovej autentifikácie, teda face recognition a aj overenie push notifikáciou. V tom prípade bude od užívateľa vyžadované pred prihlásením úspešne splniť oba druhy autentifikácie, teda sa jedná už o typ multifaktorovej autentifikácie. Pri návrhu sme sa zaoberali otázkami certifikačných autorít, ktoré by sme mohli potenciálne zakomponovať do našej aplikácie, no implementačne sa to javilo ako dosť náročné a je to jedna z úloh, ktoré sme úplne nespĺnili.

Zoznam použitej literatúry

1. RAIBLE, Matt. *What the heck IS OAuth?* 2017. Dostupné tiež z: <https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth>.
2. 8/30/2018, Rob Sobers Updated: *What is OAuth? Definition and how it works*. 2018. Dostupné tiež z: <https://www.varonis.com/blog/what-is-oauth/>.
3. HAMMER-LAHAV, Eran. *Introduction*. 2007. Dostupné tiež z: <https://oauth.net/about/introduction/>.
4. ZHANG, Ti. *OAuth 1.0 VS OAuth 2.0*. 2019. Dostupné tiež z: <https://www.loginradius.com/blog/async/what-is-the-difference-between-oauth1-and-oauth2/>.
5. KOROBKINA, Ann. *OCRA Algorithm Explained*. [B. r.]. Dostupné tiež z: <https://www.protectimus.com/blog/ocra-algorithm-explained/>.
6. M'RAIHI, D. *OCRA: OATH Challenge-Response Algorithm*. [B. r.]. Dostupné tiež z: <https://datatracker.ietf.org/doc/html/rfc6287>.
7. PETTERS, Jeff. *What is SAML and How Does it Work?* [B. r.]. Dostupné tiež z: <https://www.varonis.com/blog/what-is-saml/>.
8. TEAM, The HiveMQ. *Authorization - MQTT Security Fundamentals*. 2015. Dostupné tiež z: <https://www.hivemq.com/blog/mqtt-security-fundamentals-authorization/>.
9. TEAM, The IPC2u. *Co je MQTT a k čemu slouží ve IIoT?* [B. r.]. Dostupné tiež z: https://ipc2u.cz/blogs/news/mqtt-protokol?gclid=Cj0KCQjwkbuKBhDRARIsAALysV5dLG5xYv5d-BZ5zl6hoaAlCREALw_wcB.
10. IYER, Subbu. *Top 5 API Discovery Insights for Security Teams*. [B. r.]. Dostupné tiež z: https://securityboulevard.com/2021/09/top-5-api-discovery-insights-for-security-teams/?fbclid=IwAR1xCME17f8BHJQhg969F_Kgk9qVQLFJ9Eccxo1ybl3yy5cdguli5Trnc.
11. SIMPSON, J. *Everything You Need To Know About API Rate Limiting*. [B. r.]. Dostupné tiež z: <https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting/>.
12. BROWNLEE, Jason. *How to Perform Face Detection with Deep Learning*. [B. r.]. Dostupné tiež z: <https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/>.

13. ROSEBROCK, Adrian. *OpenCV Face Recognition*. [B. r.]. Dostupné tiež z: <https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/>.
14. MENON, Adarsh. *Face Detection in 2 Minutes using OpenCV Python*. [B. r.]. Dostupné tiež z: <https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81>.