SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

TÍMOVÝ PROJEKT

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

2022

Obsah

Zoznam obrázkov a tabuliek

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 ocra, MQTT, OpenCV

12.11.2021

- Implementovať landing page na zadanie emailovej adresy. Jednorazová URL linka pri registrácii s možnosťou nastaviť platnosť URL. Spraviť zvlásť tabuľku v databáze pre registrovaných, kde budeme rozlišovať, či sa registácia 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 informácii 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 informácii - browser a verzia, OS, IP, device fingerprint.

7.1.2022

- Ešte viac informácii 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 nájdene 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.

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 funkcionalita+dizajn, Retrofit kostra pre Android app, Nasadenie spring na server

13.12.2021 - 19.12.2021

- Stránka o nás, Implemntácia Google Auth - login + registrácia

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

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 informácii ako sú napríklad

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

Jedným z ďalších cieľov práce je implementovať funkcionalitu pri registrácii a prihlásení tak, aby sa daný používateľ vedel autentifikovať cez konkrétnu aplikáciu ako napríklad Google Authenticator alebo Microsoft Authenticator, čiže buď pomocou jednorázového 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 funkcionalita prejsť testovaním, či spĺňa všetky podmienky na úspešné zahájanie produkcie.

1 OAuth

OAuth je otvorený štandardný autorizačný protokol alebo rámec, ktorý umožňuje možnosť zabezpečeného prístupu. Pomocou tohto prístupu je užívateľovi umožnené pristúpiť k službám pomocou prihlásenia sa cez iné služby ako je napríklad Facebook alebo Google, bez toho aby užívateľ odovzdal heslo používanej službe. V prípade, že príde k úniku na používej 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 vyvýjal implementáciu Twitter OpenID. Počas tohto vývoja spolu s ďalšími prišli k záveru, že neexistujú žiadne otvorené protokoly pre delegvanie 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 prokotol 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. Ná 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 posielať 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 výrazné 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 svojím rozhraniam API.

2 Oath, Oath ocra

2.1 Oath

OATH skratka znamená Initiative for Open Authetication - 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 otázka. 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:

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

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 aplikácii. 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 autentifikácii 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 velkosť kódu a existujú obmedzené šírky pásma. Slúži k vzájomnej interakcií a komunikácií zariadení vrámci IoT.

4.1 Základné vlastnosti MQTT

- Asynchrónny protokol
- Kompaktné správy
- Podpora úrovní 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 príjmať 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 kódováním 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 správách)

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 udržuje 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.

```
repositories {
    maven {
        url "https://repo.eclipse.org/content/repositories/paho-releases/"
    }
}


dependencies {
    compile('org.eclipse.paho:org.eclipse.paho.android.service:1.0.2') {
        exclude module: 'support-v4'
    }
}
```

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:

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

Obr. 2: Deklarovanie služby

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

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

Obr. 3: Povolenia

Na vytvorenie a nadviazanie pripojenia MQTT použijeme nasledujúci útržok kódu:

```
String clientId = MqttClient.generateClientId();
MqttAndroidClient client =
        new MqttAndroidClient(this.getApplicationContext(), "tcp://broker.hivemq.com:1883",
                              clientId);

try {
    IMqttToken token = client.connect();
    token.setActionCallback(new IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken asyncActionToken) {
            // We are connected
            Log.d(TAG, "onSuccess");
        }

        @Override
        public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
            // Something went wrong e.g. connection timeout or firewall problems
            Log.d(TAG, "onFailure");

        }
    });
} catch (MqttException e) {
    e.printStackTrace();
}
```

Obr. 4: Connection setup

4.6.2 Login/Password connection

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

```
options.setUserName("USERNAME");
options.setPassword("PASSWORD".toCharArray());

IMqttToken token = client.connect(options);
```

Obr. 5: Login/Password

4.6.3 Publish

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

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

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 subscribtion úspešne vytvoriť alebo zlyhalo:

```
String topic = "foo/bar";
int qos = 1;
try {
    IMqttToken subToken = client.subscribe(topic, qos);
    subToken.setActionCallback(new IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken asyncActionToken) {
            // The message was published
        }

        @Override
        public void onFailure(IMqttToken asyncActionToken,
                              Throwable exception) {
            // The subscription could not be performed, maybe the user was not
            // authorized to subscribe on the specified topic e.g. using wildcards

        }
    });
} catch (MqttException e) {
    e.printStackTrace();
}
```

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.

```
try {
    IMqttToken disconToken = client.disconnect();
    disconToken.setActionCallback(new IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken asyncActionToken) {
            // we are now successfully disconnected
        }

        @Override
        public void onFailure(IMqttToken asyncActionToken,
                              Throwable exception) {
            // something went wrong, but probably we are disconnected anyway
        }
    });
} catch (MqttException e) {
    e.printStackTrace();
}
```

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:

- Synchrónne posielanie správ
- Asynchrónne posielanie správ

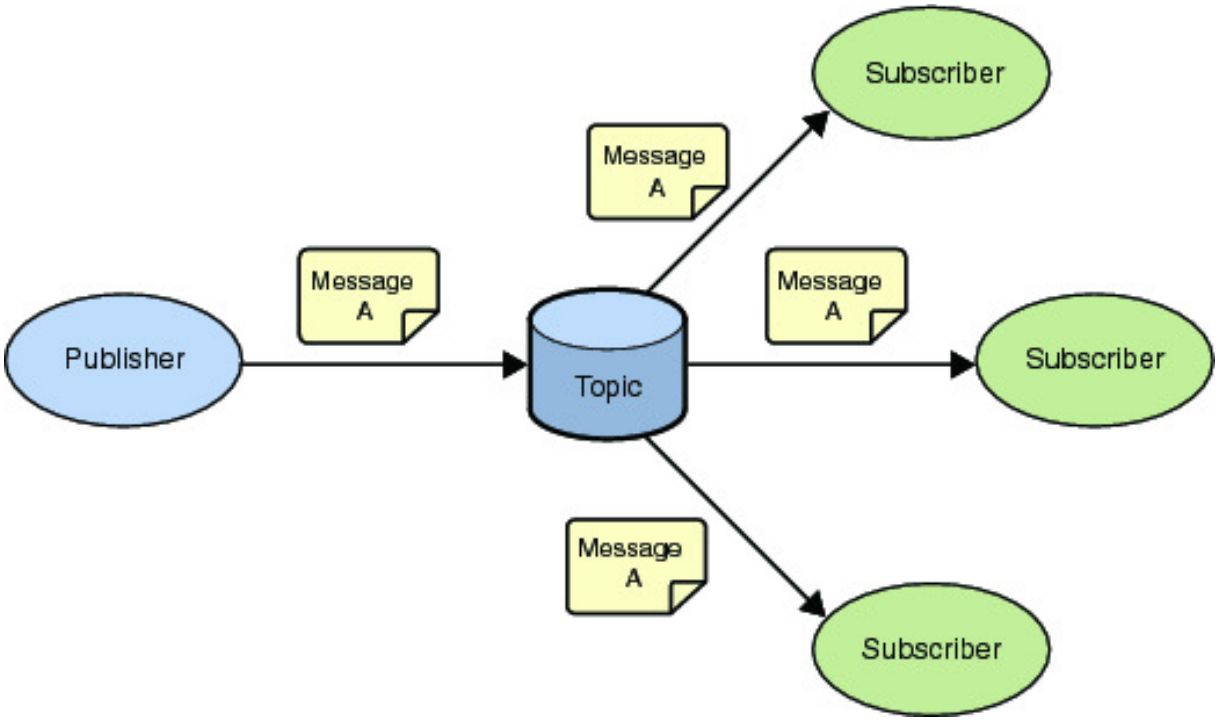
Keďže našou úlohou je vytvoriť autorizačný server, rozhodli sme sa využiť asynchrónne posielanie správ, ktoré je pre náš účel výhodnejšie z niekoľkých dôvodov:

- 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. asynchronnous messaging), ktoré sa pokúsime porovnať a najvýhodnejší spôsom 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.



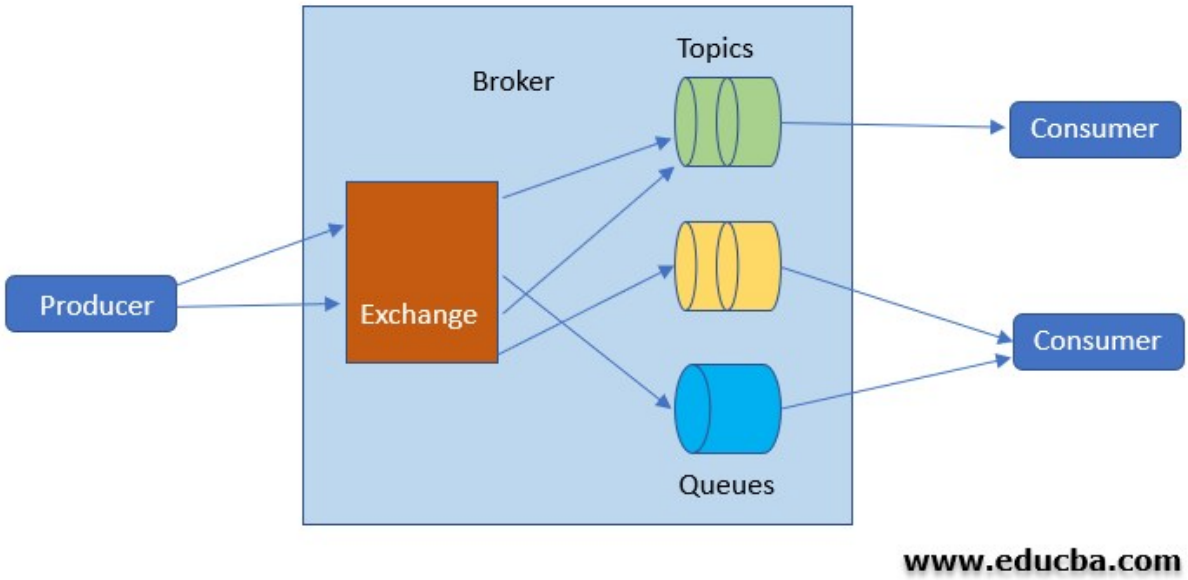
Obr. 9: zdroj: https://www.javatpoint.com/jms-tutorial

- 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: zdroj: https://www.educba.com/amqp-vs-jms/

Základné vlastnosti AMQP:

- Interoperabilita medzi viacerými jazykmi a platformami
- Škáľlvateľnosť
- 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

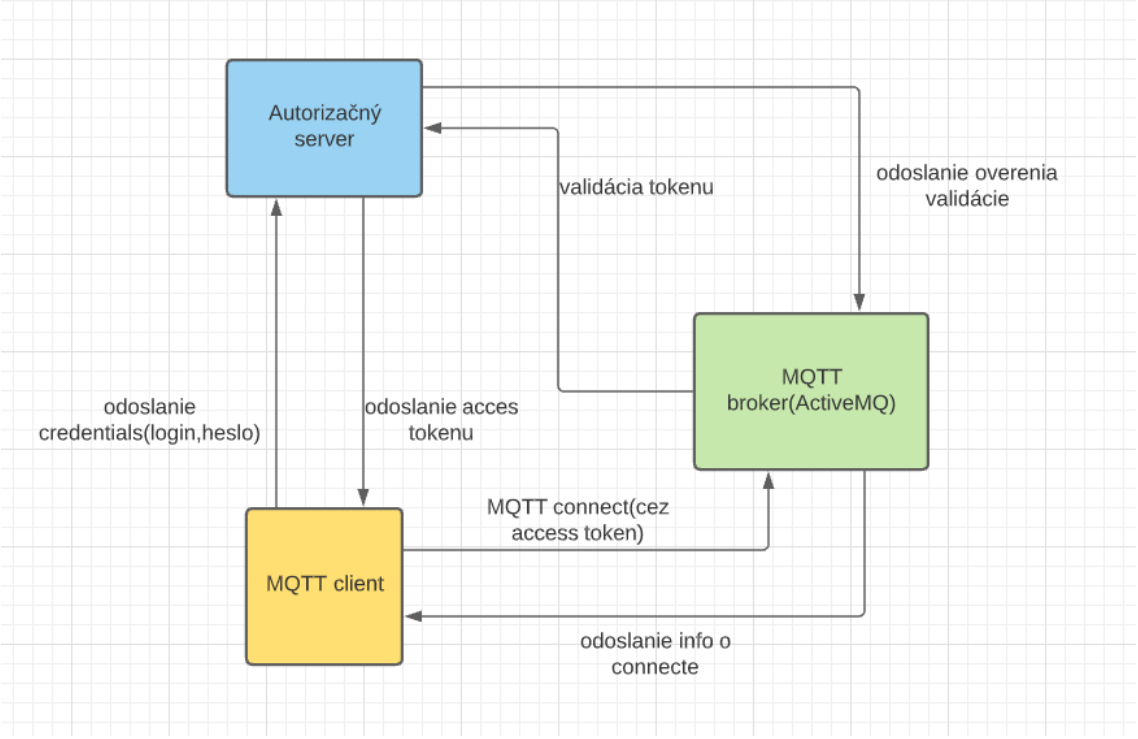
ActiveMQ je open source protokol vyvinutý spoločnosťou Apache, ktorý funguje ako implementácia middleware orientovaného na správy (MOM). Jeho základnou funkciou je posielanie správ medzi rôznymi aplikáciami.

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 pripojený pre MQTT kliena



Obr. 11: zdroj: vlastné spracovanie

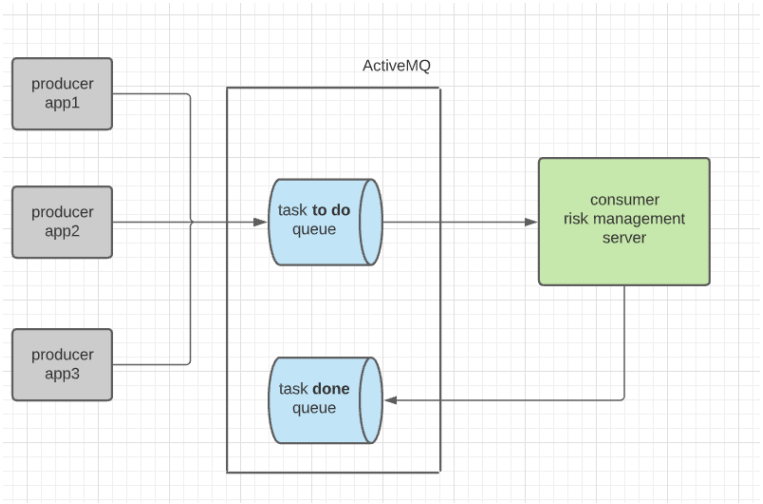
MQTT klienti sú publisheri pre topic obsahujúci autorizačné informácie. Klienti tento topic nemôžu vidieť, ale môžu do neho publishovať. Autorizačný server by predstavoval subscribera pre daný topic a vyberal by z neho informácie potrebné pre overenie autorizácie.

5.6 Architektúra ActiveMQ Brokera

Problém pri pripojení viacerých klientov naraz by mohlol 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 reťazec 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 bezpešnostných 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 týmov 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 zapezpečená. Ak by táto API bola terčom útoku tak v najlepšom prípade stratíme funkcionalitu aj našej aplikáciu , 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 funkcionalita do značnej miery zabrániť útočnikovi napadnúť náš systém, alebo sa rozhodne pre iný cieľ. Pri starších riešeniach, ak nebola implementovaná takáto funkcionalita 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 rozhodnone mali byť šifrované počas celej komunikácie. Základ je použiť HTTPS čo však nešifruje samotné dáta. Samotný druh šifrovania môže byť lubovolný, 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 podobjekty napríklad preukazy pre hľadanú 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 rozšírujúce parametre nevyužívaju 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 stat 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 niesu ani v dokumentácií a testeri ich nekontrolujú. V inom prípade niesu 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 cielom 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

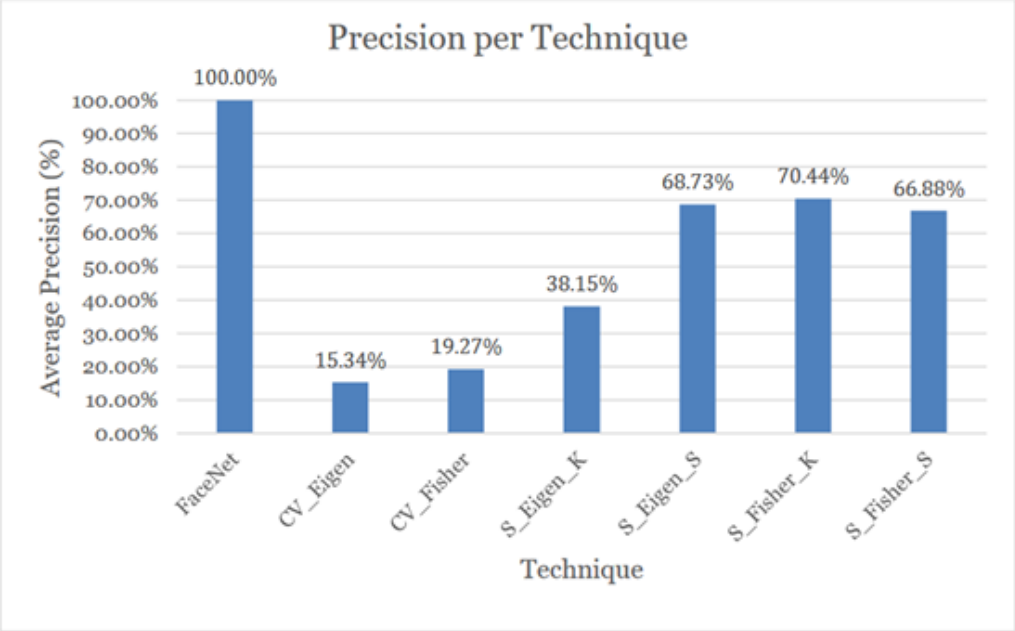
Definícia detekcie tváre je problém počítačového videnia pri lokalizácii a lokalizácii jednej alebo viacerých tvárí na fotografii. Umiestnenie tváre na fotografii znamená nájsť súradnice tváre na obrázku, zatiaľ čo lokalizácia znamená vymedzenie rozsahu tváre, často pomocou ohraničujúceho rámčeka okolo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



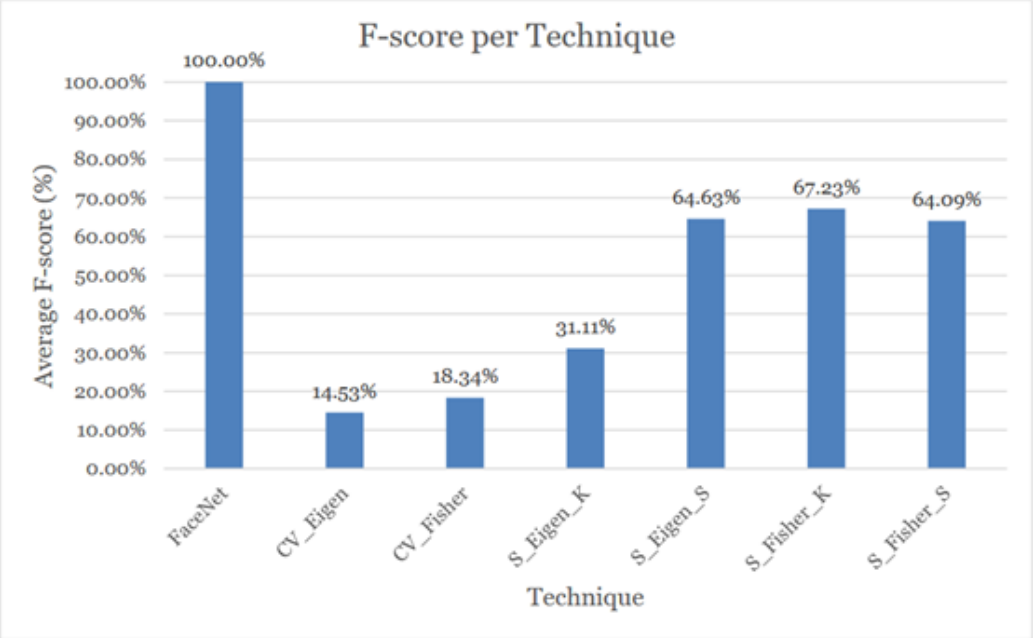
Obr. 13: Porovnanie konvolučných techník rozpoznania tváre

Tabuľky 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 metrila pre účely porovnania úspešnosti techniky, v ako aj správnych predikciáchtak 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 súbor údajov s 10-násobnou krížovou validáciou a desiatimi nasadenými cyklami na sklad. Najlepšou technikou pri porovnávaní metrík výkonu je FaceNet. Tri zo štyroch Techniky scikit-learn by sa mohli použiť v niektorých systémoch, kde presnosť nie je najvyššia dôležitosti, 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 a 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 čas 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 techniky.

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 vysokokvalitných prvkov z tvárí, nazývaných vloženie tváre, ktoré sa potom dajú použiť na trénovanie 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 čŕt 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 trénovaná 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 trénovanie modelu na priame vytváranie vložení (namiesto ich extrahovania z medzivrstvy modelu). Tieto vloženia tváre sa potom použili ako základ pre trénovanie 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.

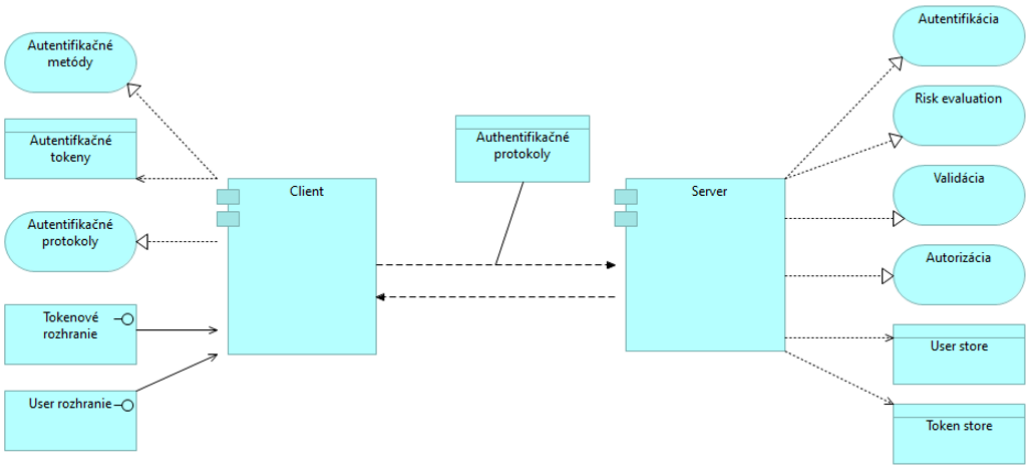
9 Návrh riešenia

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, utočník s týmto tokenom môže posielať požiadavky ako obeť.



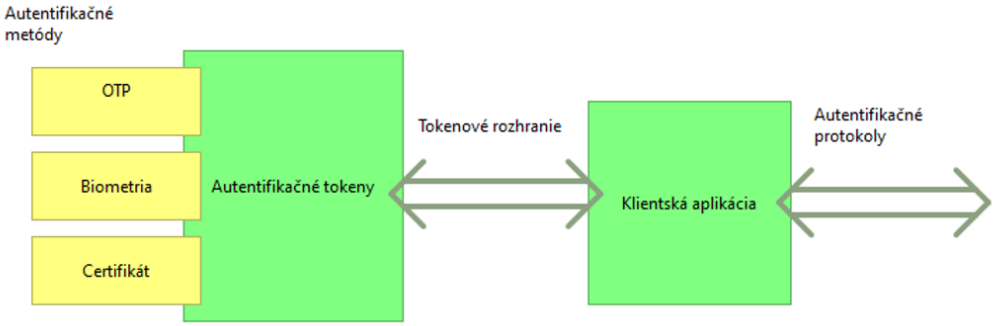
Obr. 17: High level architektúra

Token store Po autentifikáčii priradí token k užívateľovi. K tomuto bodu musia mať prístup všetky časti architektúry, ktoré chceme aby boli zapezpečené 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 hladiska je dôležité zabezpečiť aplikáciu čo najlepšie, aby nebolo možné zneužiť jej autentifikačné funkcie.



Obr. 18: Klienske rozhranie

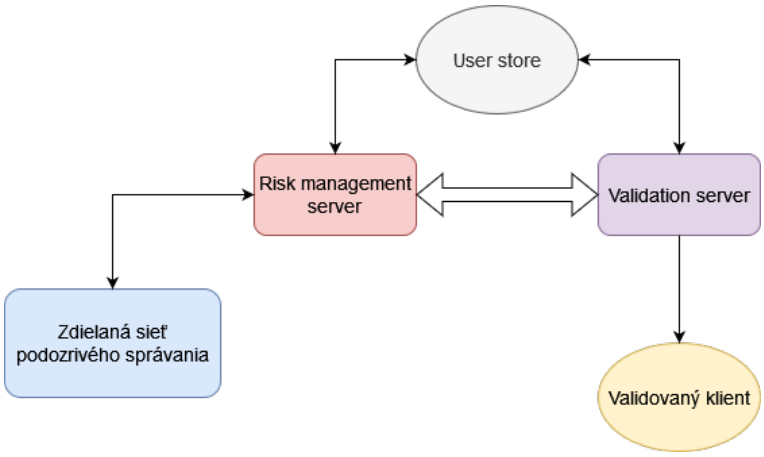
One Time Password Výhodou tohto typu autentifikácie je, že aj keby došlo k jeho ukradnutiu, toto heslo nieje môžné zneužiť. Ich krátka platnosť je zároveň aj ich nevýhodou, lebo používatelia si ich nemôžu uložiť lebo by boli neplatné.

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 otlačkov prstov. Kedže 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 zaujimavé pripojiť tento modul do zdielanej 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.

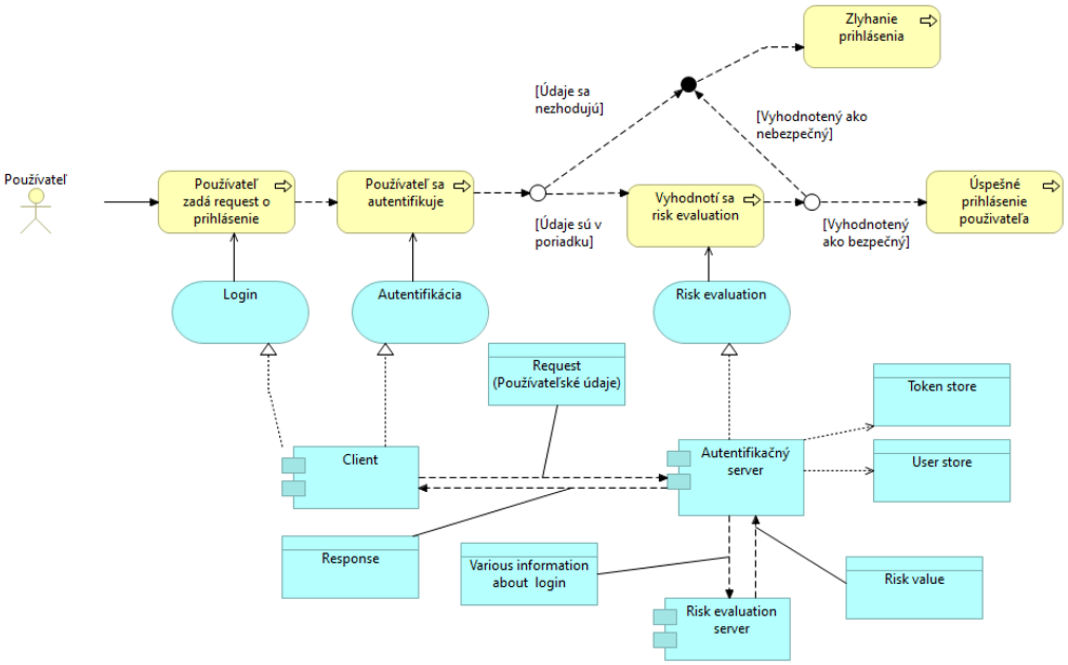


Obr. 19: Risk manažment

9.1.4 Dvojfaktorová autentifikácia

Táto funkcionalita 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.

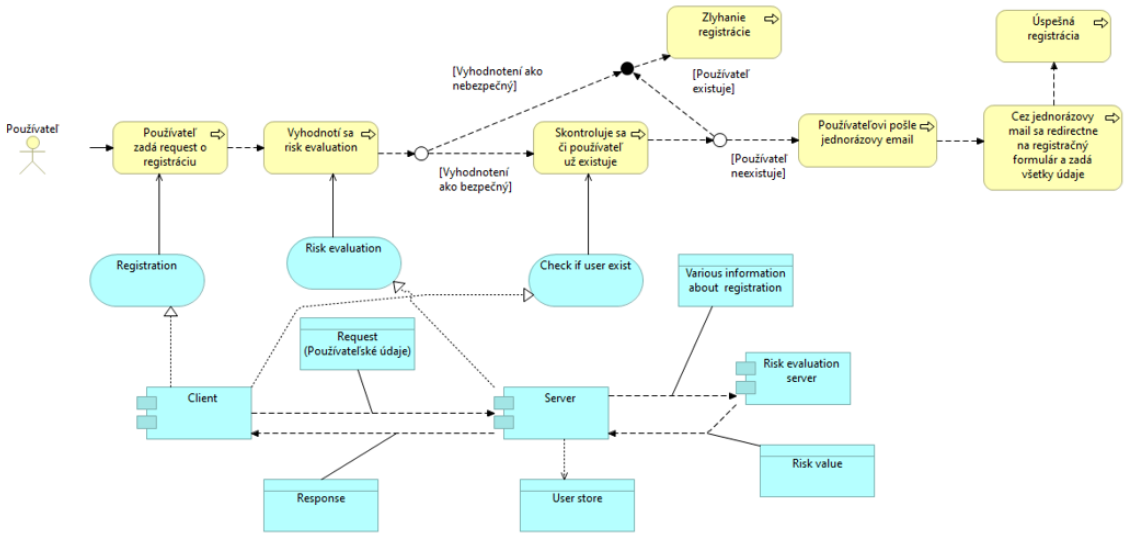
9.1.5 Prihlásenie



Obr. 20: Proces prihlásenia

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.

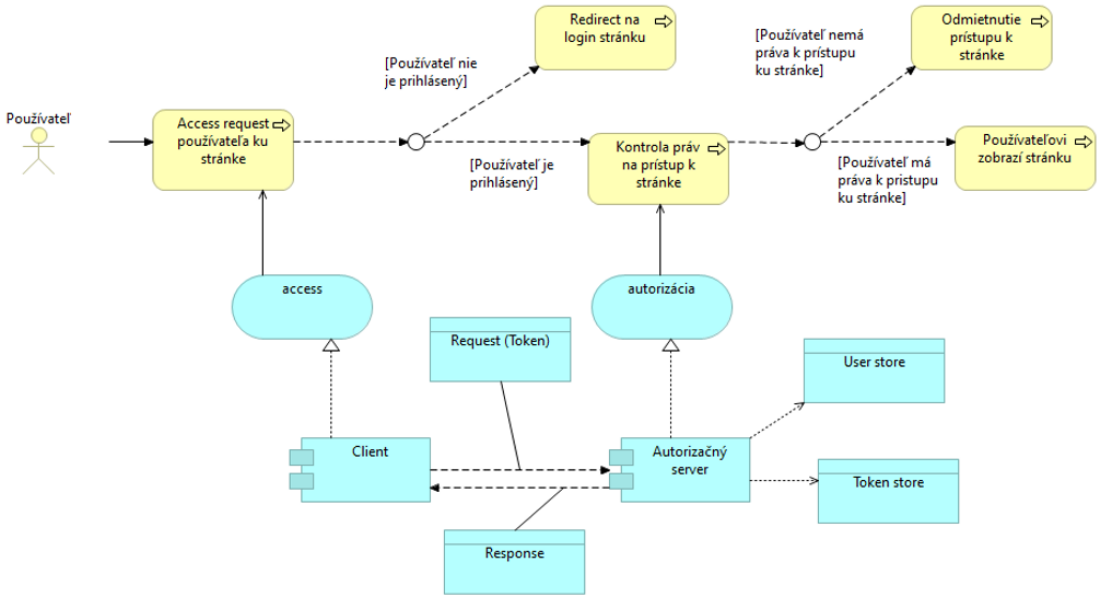
9.1.6 Registrácia



Obr. 21: Proces registrácie

Používateľ zadá request (spolu s používateľskými údajmi) na registrovanie sa do systému. Systém najprv vyhodnotí rizko 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..

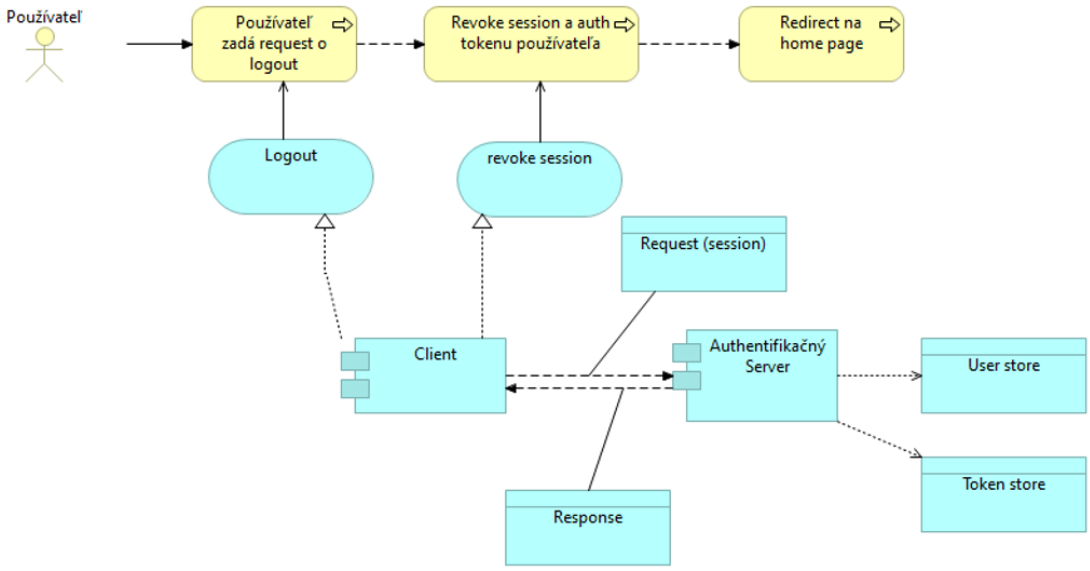
9.1.7 Autorizácia



Obr. 22: Proces autorizácie

Na obrázku vyššie 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.

9.1.8 Odhlásenie

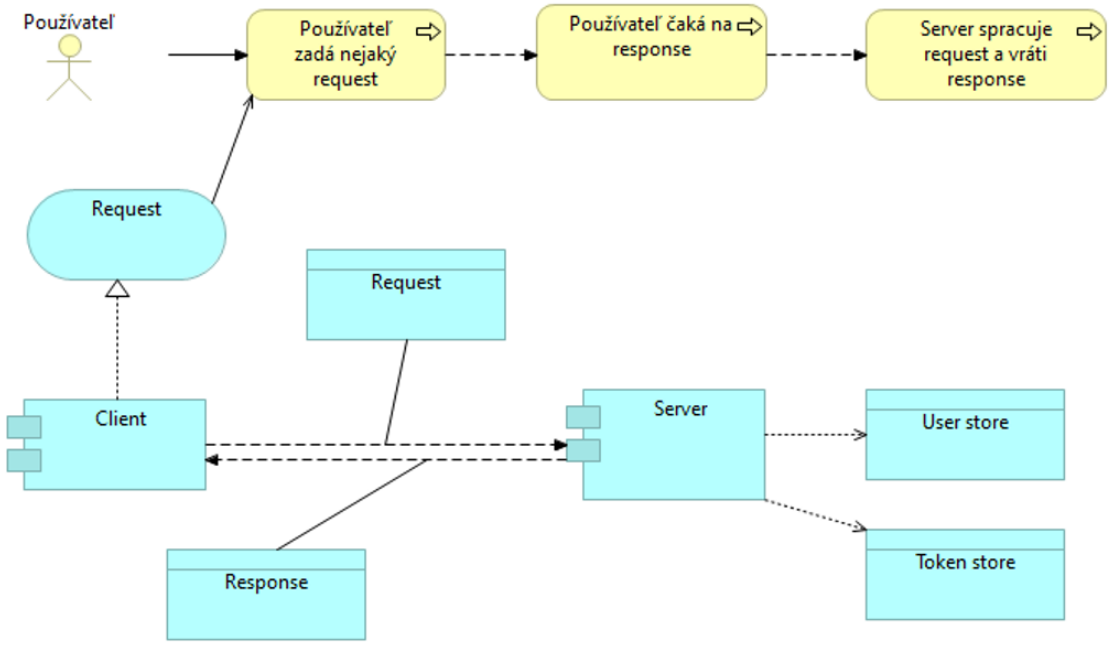


Obr. 23: Proces odhlásenia

Ďalej môžeme vidieť proces používateľa odhlásiť 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.

9.1.9 Synchrónne servisy

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

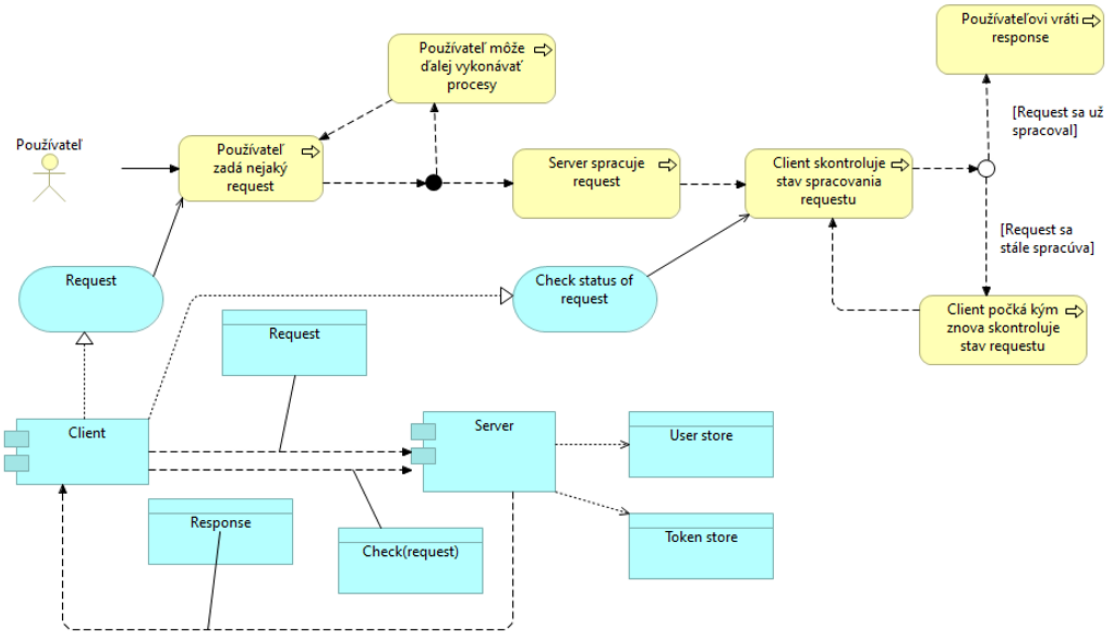


Obr. 24: Synchrónny proces

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 operácii a poprípade 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 systém potrebuje viacej 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 vyhodnotení 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 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 povereniam 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 funkcionalitu 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.

Záver

Bude doplnené

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_Kgk9qVQLFJ9Eccxo1yb3yy5cdguli5Trnc.
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.