

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

**SPRING BOOT & SECURITY
IMPLEMENTÁCIA VIAC-FAKTOROVEJ
AUTENTIFIKÁCIE**

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

2022

Názov práce: Spring Boot & Security – viacfaktorová autentifikácia

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. Navrhnite a implementujte vhodný spôsob dvojfaktorovej autentifikácie s využitím mobilnej aplikácie Google Authenticator alebo Microsoft Authenticator. Analyzujte a porovnajte použitie OTP alebo push notifikácie.
7. Vyhodnoťte výsledky implementácie z funkčnej a používateľskej stránky.

Obsah

1	OAuth	1
1.1	História	1
1.2	Bezpečnostné problémy	1
1.2.1	OAuth 1.0	1
1.2.2	OAuth 2.0	1
1.3	OAuth 2.0	1
1.4	Použitie	2
2	SAML	3
2.1	Použitie	3
2.2	SAML provider	3
2.3	Funkcionalita	3
2.4	Výhody	4
2.5	Nevýhody	4
3	Riešenie	5
3.1	Architektúra	5
3.1.1	Autentifikačná platforma	5
3.1.2	Klientske rozhranie	5
3.1.3	Dvojfaktorová autentifikácia	6
3.1.4	Prihlásenie	7
3.1.5	Registrácia	8
3.1.6	Autorizácia	9
3.1.7	Odhlásenie	10
3.1.8	Synchrónne servisy	11
3.1.9	Asynchrónne servisy	12
3.2	Porovnanie vhodných serverov	13
3.2.1	CAS	13
3.2.2	Keycloak	13
3.2.3	Java Spring Boot	14
3.2.4	Výber serveru	15
4	Implementácia	16
4.1	Výsledná architektúra	16
4.1.1	Diagram pre <i>High-level</i> architektúru	16
4.1.2	Diagram klientského rozhrania	17
4.1.3	<i>Diagram pre device enrolment</i>	18

4.1.4	Diagram autorizácie	19
4.1.5	Diagram prihlásenia	20
4.1.6	Diagram registrácie	21
4.2	Viac-faktorová autentifikácia	22
4.2.1	Authy aplikácia	22
4.2.2	Authy One Time Password	22
4.2.3	Authy push notifikácie	25
4.3	Registrácia	28
4.3.1	Implementácia	30
4.3.2	Vytvorenie tokenu	30
4.3.3	Overenie tokenu	30
4.3.4	Jednorazová URL	30
4.3.5	Verifikácia vstupov	31
4.4	Prihlásenie	32
4.4.1	Postup implementácie	33
4.4.2	Druhy prihlásenia	33
4.5	Systémové roly	34
4.5.1	Vytvorenie používateľských rolí	34
5	Prínos	35
	Záver	36

Zoznam obrázkov a tabuliek

Obrázok 1	Proces prihlásenia	7
Obrázok 2	Proces autorizácie	9
Obrázok 3	Proces odhlásenia	10
Obrázok 4	Synchrónny proces	11
Obrázok 5	Asynchrónny proces	12
Obrázok 6	Finálny vzhľad <i>high-level</i> architektúry	16
Obrázok 7	Finálny vzhľad klientského rozhrania	17
Obrázok 8	<i>Device enrolment</i> počas registrácie	18
Obrázok 9	Finálny vzhľad <i>high-level</i> procesu autorizácie	19
Obrázok 10	Finálny vzhľad <i>high-level</i> procesu prihlásenia	20
Obrázok 11	Finálny vzhľad <i>high-level</i> procesu registrácie	21
Obrázok 12	Parametre URL	23
Obrázok 13	Možné odozvy po poslaní požiadavky	24
Obrázok 14	Dodatočné parametre pre URL	26
Obrázok 15	Dodatočné parametre pre URL	27
Obrázok 16	Prvý krok pri registrácii nového užívateľa	28
Obrázok 17	Jednorazová URL odoslaná na email registrujúceho sa užívateľa	28
Obrázok 18	Registračný formulár	29
Obrázok 19	Prihlasovanie užívateľa	32
Obrázok 20	Úspech prihlásenia	32

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.

1.1 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.

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 protokolu HTTPS. To znamená že toky nemusia byť šifrované na koncových bodoch, keďže

sú zašifrované už počas prenosu.

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 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. 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.

2.1 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.

2.2 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.

2.3 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.

2.4 Výhody

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

2.5 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

3 Riešenie

3.1 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.

3.1.1 Autentifikačná platforma

Na serveri budú implementované 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 posielat požiadavky ako obeť.

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.

3.1.2 Klientske rozhranie

Autentifikácia klient prebieha v Android aplikácii, 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.

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.

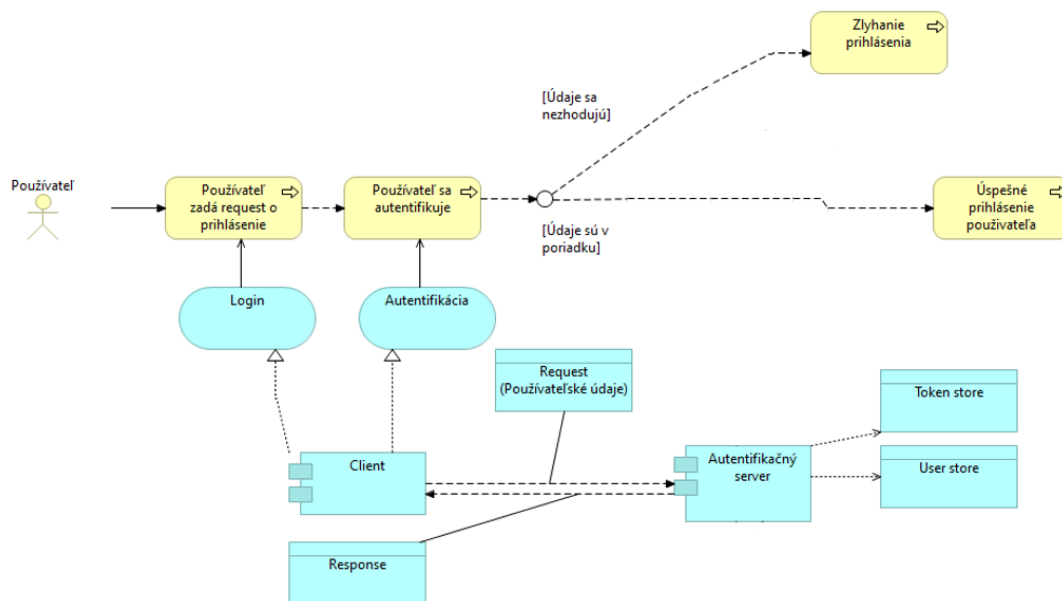
Certifikát Autentifikácia založená na certifikáte je použitie digitálneho certifikátu na identifikáciu používateľa alebo zariadenia pred udelením prístupu k zdroju, sieti, aplikácii atď. V prípade overenia používateľa sa často používa v koordinácii s tradičnými metódami, ako sú napríklad používateľské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

3.1.3 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.

3.1.4 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.



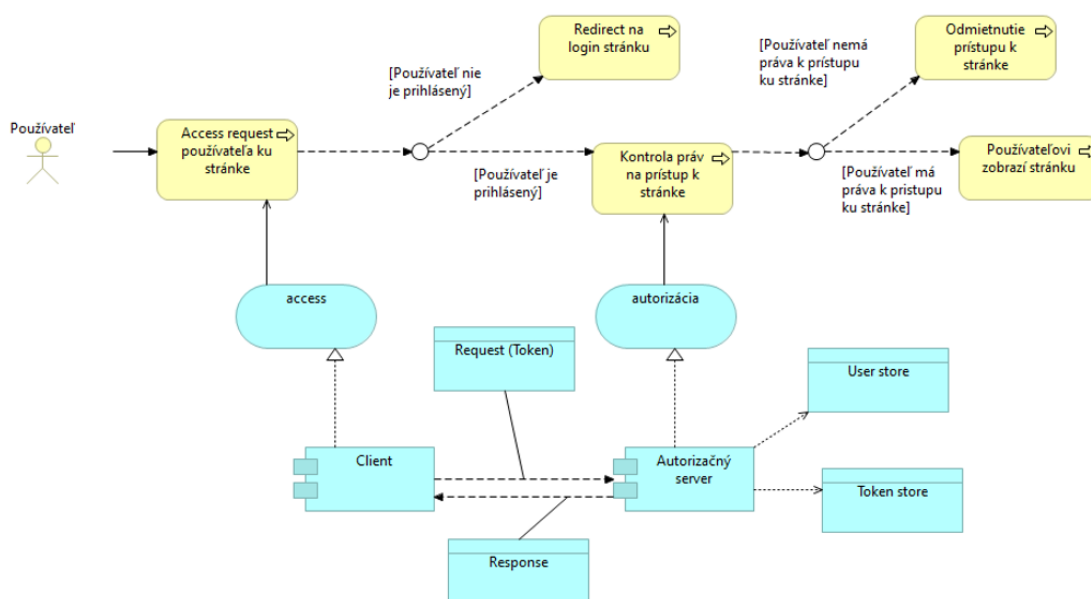
Obr. 1: Proces prihlásenia

3.1.5 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.

3.1.6 Autorizácia

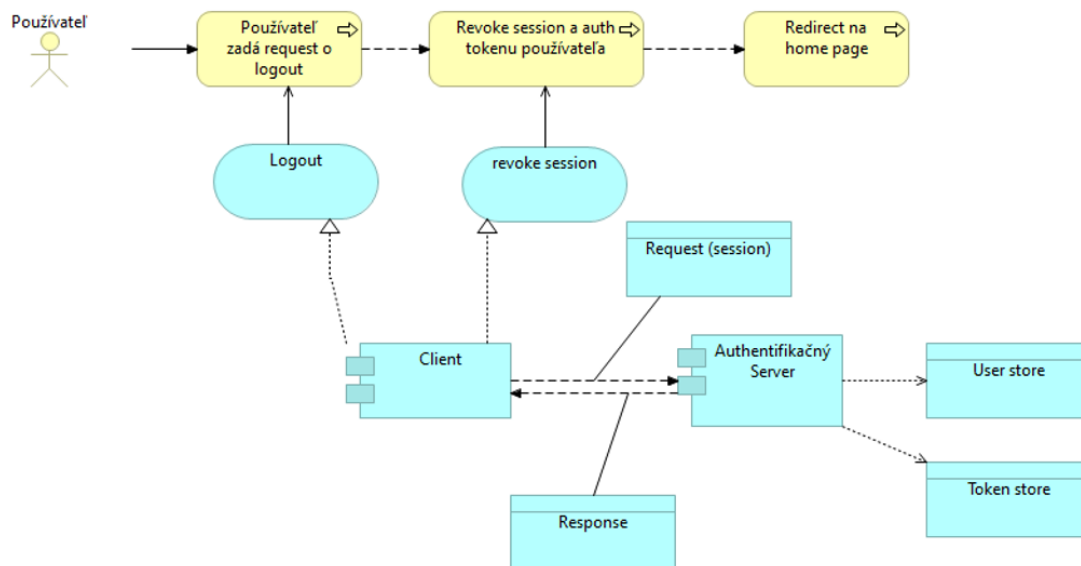
Na obrázku môžeme vidieť postup autorizácie používateľa pri požiadaní o prístup na nejakú stránku. Najprv používateľ zadá request o prístup na stránku, systém overí či je vôbec prihlásený, ak nie je tak ho presmeruje na stránku s loginom. Ak je prihlásený, tak používateľa následne autorizuje a teda skontroluje 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. 2: Proces autorizácie

3.1.7 Odhlásenie

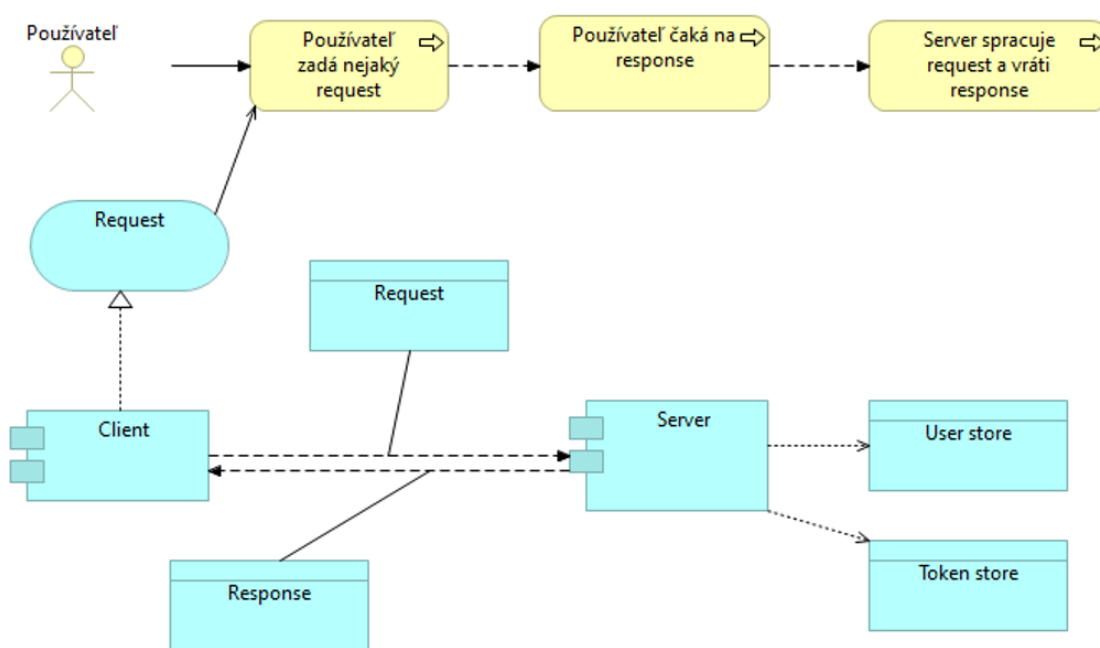
Na obrázku vidíme proces používateľa pre odhlásenie sa zo systému. Najprv používateľ zadá request, že sa chce odhlásiť z účtu. Systém spracuje tento request a odstráni jeho lokálnu session a tiež tú v databáze. Po úspešnom vymazaní session ho presmeruje na home page.



Obr. 3: Proces odhlásenia

3.1.8 Synchronne servisy

Hlavnou črtou synchronných services je, že keď používateľ zadá nejaký request tak čaká kým sa service dokončí, dotedy 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.

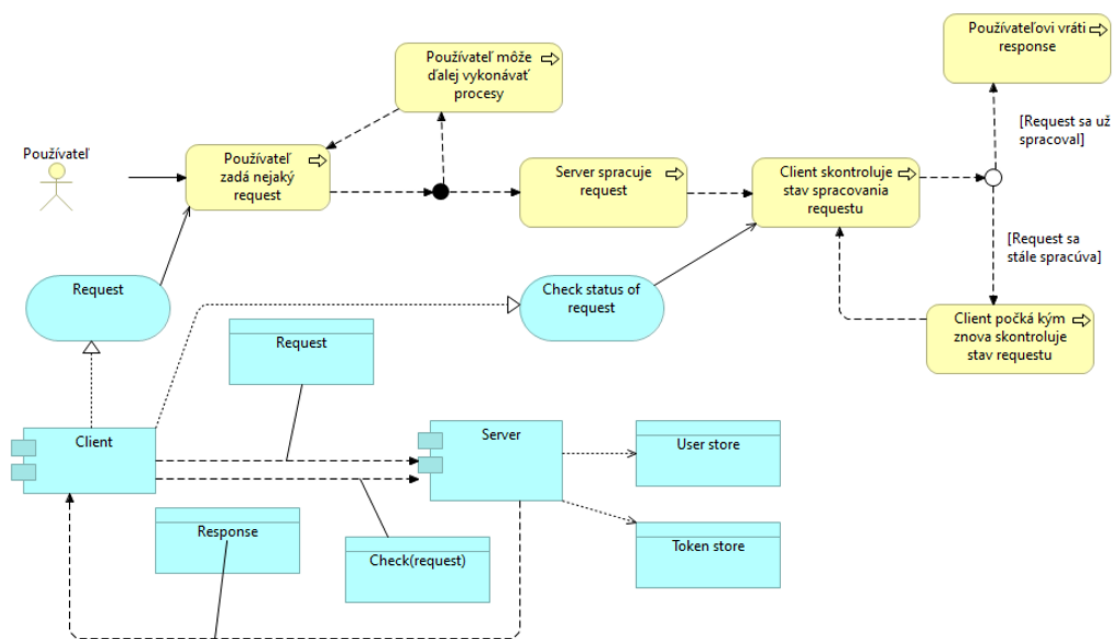


Obr. 4: Synchronný proces

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

3.1.9 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. 5: 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.

3.2 Porovnanie vhodných serverov

3.2.1 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

3.2.2 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

3.2.3 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

3.2.4 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.

4 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.

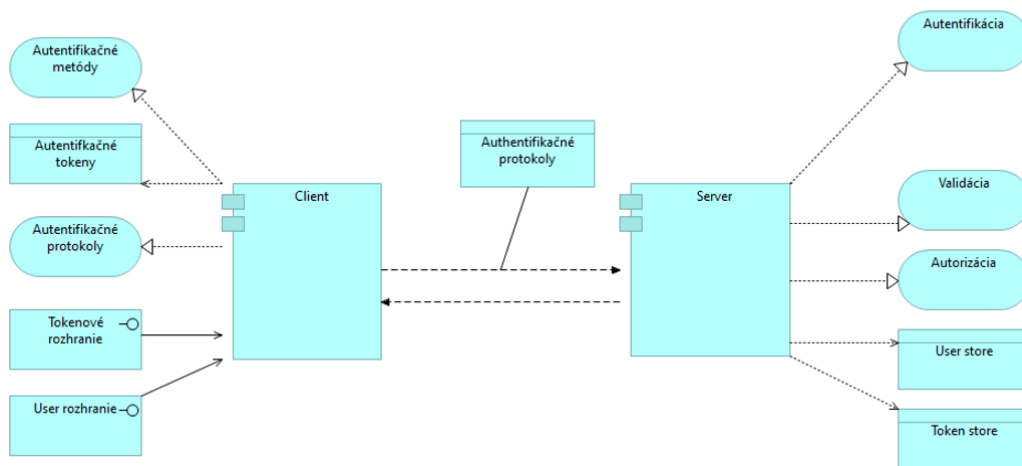
4.1 Výsledná architektúra

Architektúra vo svojej finálnej podobe pozostáva z:

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

4.1.1 Diagram pre *High-level* architektúru

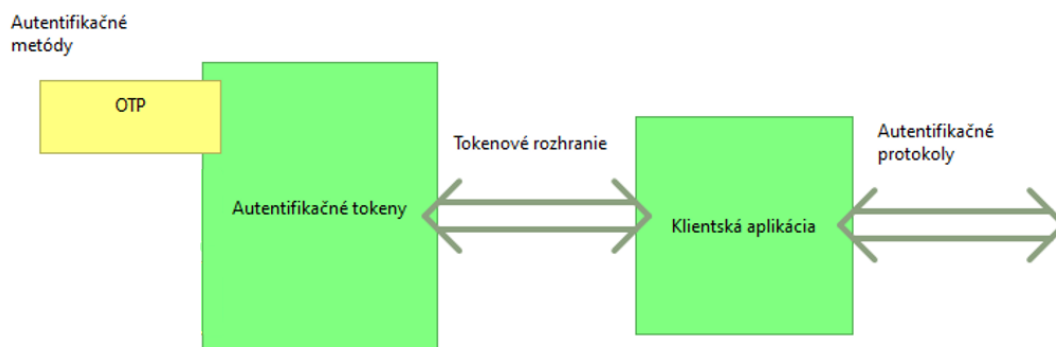
Oproti návrhu riešenia sme do architektúry ešte potrebovali pridať aj *device store*, ktorý drží informácie o klientskom zariadení, ktoré používateľ použil pri registrácii do systému. Ponechali sme *user a token store*, pretože autentifikačný server by nedokázal fungovať na takom autentifikačnom princípe,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



Obr. 6: Finálny vzhľad *high-level* architektúry

4.1.2 Diagram klientského rozhrania

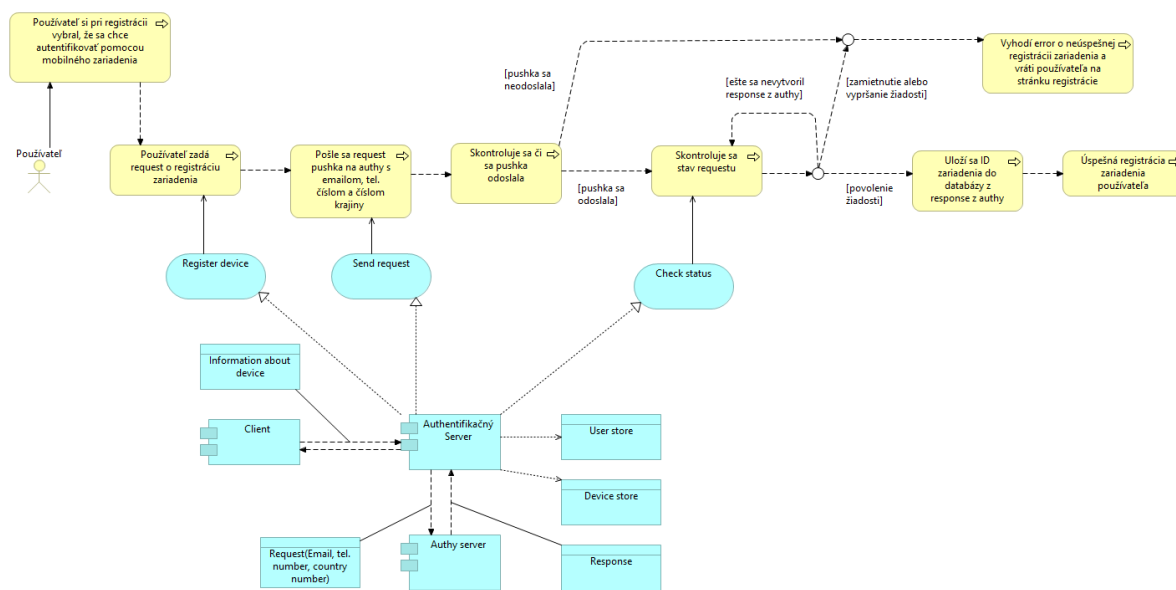
Pri rozhraní pre klienta sme doimplementovali certifikát cez certifikačnú autoritu a taktiež *push* notifikácie. Klient sa po úspešnej autentifikácii prihlási a získa certifikát, ktorého platnosť určuje certifikačná autorita, podľa toho ako je nastavená. *Push* notifikácie sme implementovali cez *Authy* 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. 7: Finálny vzhľad klientského rozhrania

4.1.3 Diagram pre device enrolment

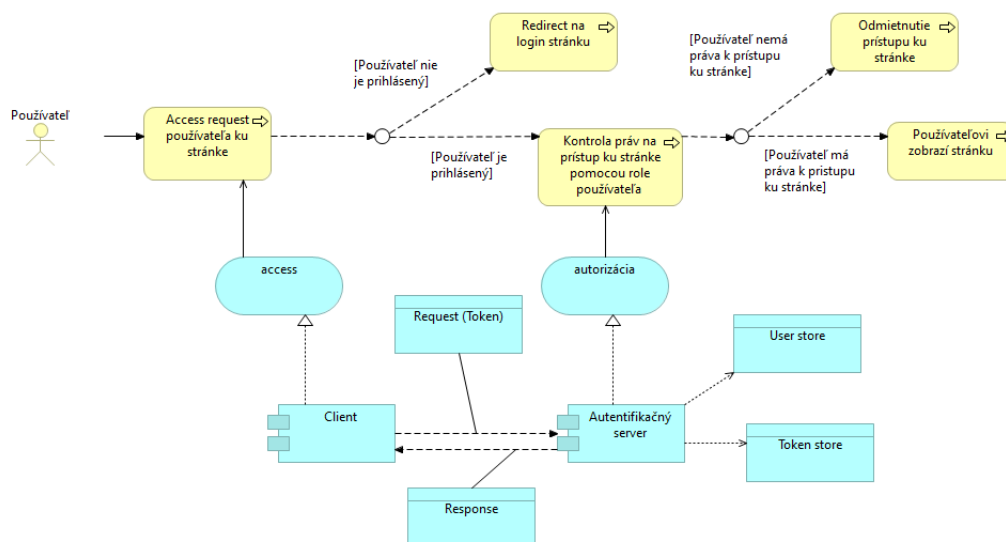
Diagram pre *device enrolment* sme pri návrhu riešenia neuviedli, pretože návrh riešenia nepočítal s rozsiahlejšou registráciou používateľa a jeho zariadenia. Pri implementácii sme však zistili, že by bolo vhodné mať všetky informácie o používateľovi ešte predtým, ako mu schválime celú registráciu a 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. 8: *Device enrolment* počas registrácie

4.1.4 Diagram autorizácie

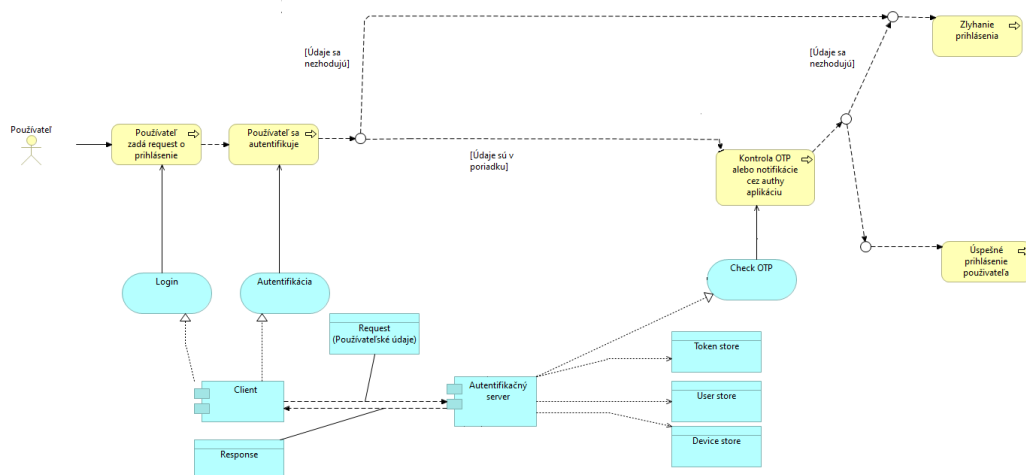
Pri diagrame pre autorizáciu sme urobili pár zmien, hlavne, čo sa týka systémových rol. Zatiaľ, čo v návrhu riešenia sme systémové roly preskočili a nezaoberali sa nimi, teraz sme potrebovali celý autorizačný proces dokončiť tak, aby sme po prihlásení vedeli identifikovať administrátora a bežného používateľa. 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. 9: Finálny vzhľad *high-level* procesu autorizácie

4.1.5 Diagram prihlásenia

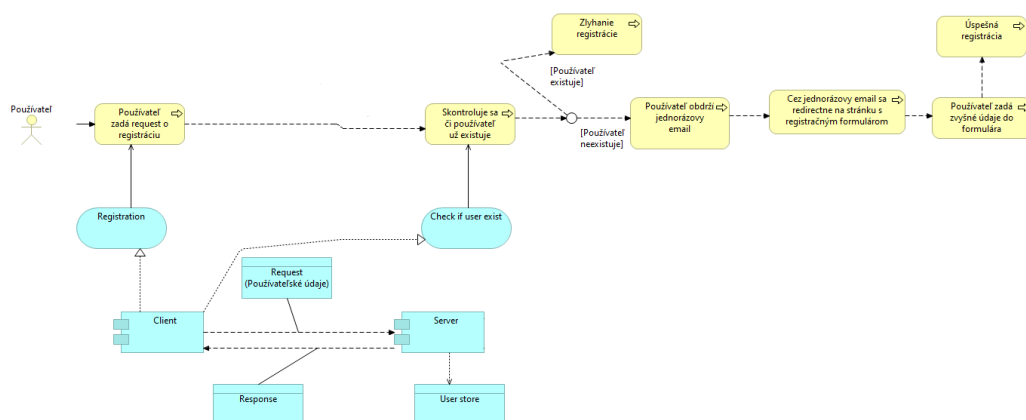
Diagram prihlásenia bol doplnený o viacero vecí. Ak sa používateľ vyhodnotí, že predstavuje nízke riziko, bude mu stačiť sa prihlásiť mailovou adresou a heslom. A však, ak sa na našom *risk-server*-i vyhodnotí prihlásenie používateľa ako rizikové, tak podľa stupňa rizika budú pribúdať faktory pre autentifikáciu. 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. 10: Finálny vzhľad *high-level* procesu prihlásenia

4.1.6 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. 11: Finálny vzhľad *high-level* procesu registrácie

4.2 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*.





4.2.1 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.

4.2.2 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. 12: 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. 13: 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.

4.2.3 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. 14: 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

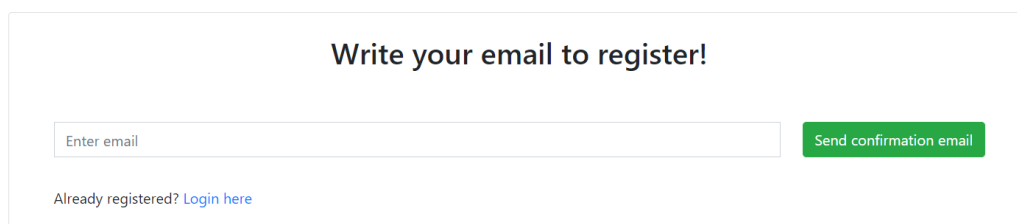
Name	Description
<code>approval_request</code> Hash	Hash containing the keys & values for the <code>ApprovalRequest</code> . (🔒 PII)
<code>uuid</code> String	Unique transaction ID of the <code>ApprovalRequest</code> . You'll need the <code>uuid</code> to query the request status or tie future callbacks 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. 15: Dodatočné parametre pre URL

4.3 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



Write your email to register!

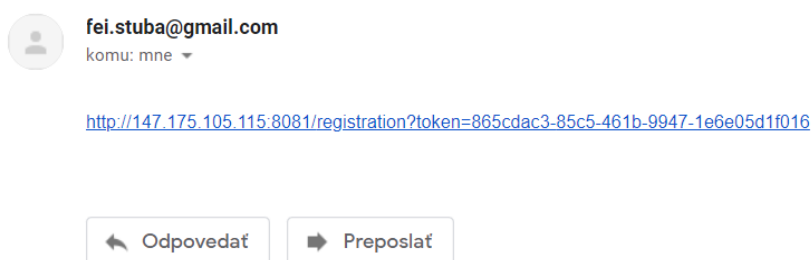
Enter email

Send confirmation email

Already registered? [Login here](#)

Obr. 16: Prvý krok pri registrácii nového užívateľa

- Na zadanú emailovú adresu je mu odoslaná jednorazová URL



Obr. 17: 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

Registration

First Name

Last Name

Please fill out this field.

Email

Phone number

 370 ▼

Password

Do you want to use 2FA?

[Register](#)

Already registered? [Login here](#)

Obr. 18: Registračný formulár

4.3.1 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.

4.3.2 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*.

4.3.3 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.

4.3.4 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 *getteri* a *setteri*.

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.

4.3.5 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.

4.4 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

New user? [Register here](#)

Obr. 19: Prihlasovanie užívateľa

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

You have been logged successfully!

Welcome ezekeieqt@gmail.com

Obr. 20: Úspech prihlásenia

4.4.1 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.

4.4.2 Druhy prihlásenia

Vrámci 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.

4.5 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.

4.5.1 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.

5 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 komunite, 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.