Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Zabezpečená webová aplikace

Bakalářská práce

Patrik Marýška

Vedoucí práce: Ing. František Drdák, CSc.

České Budějovice 2019

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: **Patrik Maryška**
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika

Katedra: Ústav aplikované informatiky

Školitel: ing. František Drdák, CSc.
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF: .....
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant: .....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: **Zabezpečená webová aplikace**
Cíle práce:

1. Stanovit bezpečnostní požadavky na zajištění obsahu webové aplikace např. agenda představenstva společnosti (porady, smlouvy, ekonomické výsledky) z hlediska uživatelských přístupů a komunikačních protokolů eventuálně šifrování uložených dat. Zvážit použití PKI.
2. Provést implementaci vzorové webové aplikace v technologii Java Spring tak, aby splňovala výše uvedené požadavky.
3. Provést hodnocení (formou penetračních testů), do jaké míry bylo stanovených požadavků dosaženo.

Základní doporučená literatura:

1. Dokumentace k technologii Spring
2. Libor Dostálek: Velký průvodce infrastrukturou PKI

Financování práce :.....

Vedoucí práce : ing. František Drdák, CSc podpis : .....

U externích vedoucích fakultní garant práce.....podpis : .....

Garant oboru bak.. studia (nepožaduje se u zaměření „příprava na mag. studium biologie)

..... podpis : .....

Vedoucí katedry: Ing. Rudolf Vohnout, PhD podpis .....

Případný souhlas vedoucího ústavu AV .....podpis : .....

V Českých Budějovicích dne 02.05.2019

Převzal/a dne..... podpis : .....

Bibliografické údaje

Marýška P., 2019: Zabezpečená webová aplikace. [Secure web application. Bc.Thesis, in Czech] – 87 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce se zabývá praktickou implementací zabezpečené webové aplikace za použití vhodných bezpečnostních mechanismů. Tato aplikace je určena pro firmu k řízení vnitropodnikové dokumentace. Implementace je provedena pomocí frameworku Spring Boot na backendu a pomocí frameworku jQuery na frontendu. Další náplní práce je provedení zhodnocení zabezpečení aplikace pomocí penetračních testů.

Anotation

This bachelor thesis deals with the process of creating a secure web application with appropriate security mechanisms for managing intracompany documentation within a given company. The Spring Boot framework is used at the backend, and jQuery framework is used at the frontend. The paper goes on to evaluate the security level of the application using penetration testing.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 5.12 2019

Podpis: _____

Poděkování

Děkuji vedoucímu bakalářské práce, Ing. CSc. Františku Drdákovi, za cenná doporučení, trpělivost a poskytnutí odborné literatury k tématu. Také děkuji Ing. Petru Břehovskému za cenná doporučení z oblasti penetračních testů.

Obsah

1. **Úvod** ... 1

1.1 Cíl práce ... 1

2. **Analytická část** ... 2

2.1 Analýza aplikace ... 2

2.1.1 UML use case diagram ... 2

2.1.2 Relační model databáze ... 3

2.1.3 Architektura aplikace ... 5

2.2 Bezpečnostní mechanismy ... 6

2.2.1 HTTPS ... 6

2.2.2 Heslo ... 7

2.2.3 Oauth2 ... 8

2.2.4 XSS ... 9

2.2.5 Autorizace ... 10

2.2.6 Validace inputu ... 10

2.2.7 Šifrování dokumentů na disku ... 11

2.2.8 SQL injection ... 11

2.2.9 Cookies ... 11

2.2.10 Logování ... 11

2.2.11 Povolené metody ... 12

2.2.12 CORS ... 12

2.2.13 Bezpečnostní hlavičky na REST serveru ... 12

2.3 Penetrační testy ... 13

3. **Implementační část** ... 15

3.1 Funkcionalita aplikace ... 16

3.1.1 Struktura aplikace ... 16

3.1.2 Inicializace ... 18

3.1.3 Dokumenty ... 19

3.3.4 Uživatelé ... 33

3.3.5 Skupiny ... 35

3.3.6 Další funkcionalita 36
3.2 Bezpečnostní mechanismy na serveru 37
3.2.1 Oauth2 37
3.2.2 HTTPS 41
3.2.3 Autentizace 41
3.2.4 Autorizace 43
3.2.5 Heslo 43
3.2.6 Chyby 45
3.2.7 CORS 45
3.2.8 Managment end-pointy 46
3.2.9 Logování 46
3.2.10 Šifrování na disku 47
3.2.11 Zakázané a povolené metody 48
3.2.12 Validace inputu 48
3.2.13 Obrana proti útoku XSS 50
3.2.14 Obrana proti útoku SQL injection 50
3.2.15 Obrana proti útokům hrubou silou 50
3.3 Bezpečnostní mechanismy na frontendu 51
3.3.1 Oauth2 51
3.3.2 XSS 54
3.3.3 Další bezpečnostní prvky 54
3.4 Konfigurace webového serveru IIS 55
4. **Testování** 58
4.1 – Testování sběru informací (OTG-INFO) 58
4.2 - Testování správy konfigurace (OTG-CONFIG) 60
4.3 – Testování správy identit (OTG-IDENT) 62
4.4– Testování autentizace (OTG-AUTHN) 64
4.5 – Testování autorizace (OTG-AUTHZ) 66
4.6 – Testování řízení relace (OTG-SESS) 69
4.7 - Testování validace vstupu (OTG-INPVAL) 69

4.8 – Testování, jak aplikace pracuje s chybami (OTG-ERR) 72
4.9 - Testování slabé kryptografie (OTG-CRYPST) 73
4.10 - Testování business logiky (OTG-BUSLOGIC) 73
4.11 - Testování klientské strany (OTG-CLIENT) 75
4.12 Výsledky testů 78
5. **Závěr** 79
6. **Seznam použité literatury** 80
7. **Přílohy** 83
I. Obsah DVD 83
II. Seznam obrázků 83

1. Úvod

Webová aplikace je aplikace, která je uložena na vzdáleném serveru a je dostupná klientům za použití internetu přes rozhraní webového prohlížeče. Webovou aplikaci obvykle tvoří několik částí – webový server, aplikační server a databáze.

Velkou výhodou webových aplikací je jejich snadná údržba. Uživatel také nemusí instalovat žádný speciální softwarový balík k ovládání programu, protože mu postačí pouze internetový prohlížeč.

Webová aplikace může být navržena pro velké množství užití a může ji využívat každý, od organizace až po běžného uživatele, který má možnost denně pracovat s různými typy webových aplikací. Velmi používaným typem webových aplikací jsou jednoznačně e-commerce aplikace, internetové obchody, platební brány a platební procesory, dynamické webové prezentace, diskuzní fóra i aplikace na straně veřejné správy.

Webové aplikace mohou být velmi zranitelné a jejich úroveň zabezpečení by měla odpovídat obsahu dané aplikace. Při vývoji aplikace, která bude pracovat s citlivými informacemi (obchodní transakce, rodná čísla klientů atd), by mělo být dbáno na zabezpečení s mnohem větší důležitostí.

1.1 Cíl práce

Hlavním cílem mé bakalářské práce je praktická implementace bezpečné webové aplikace za použití vhodných bezpečnostních mechanismů. Tato aplikace bude běžet v lokální síti. Na straně serveru bude použit framework Spring Boot a programovací jazyk Java 12. Na tomto serveru bude vystavěno API v podobě REST služeb. Klientská část aplikace bude implementována pomocí frameworku jQuery.

Aplikace bude určena pro firmu k řízení vnitropodnikové dokumentace, jakožto jednoho ze základních nástrojů řízení firmy. To znamená, že uživatel, který bude mít právo nasdílet dokument, si bude moct navolit, jací uživatelé budou muset daný dokument v jím určeném časovém horizontu schválit a kteří uživatelé budou mít povinnost se seznámit s daným dokumentem po schválení. Aplikace bude umožnovat rozesílání hromadných emailů na základě různých událostí. Základní události budou schválení dokumentu, zablokování dokumentu, vytvoření dokumentu, expirace dokumentu a obdržení nového dokumentu. Funkcionalita různého vyhledávání dokumentů, základní administrace uživatelů a dokumentů bude také implementována. Dokumentace může obsahovat velmi citlivé informace (smlouvy, různé ekonomické výsledky atd). Povolené typy dokumentů budou soubory typu PDF.

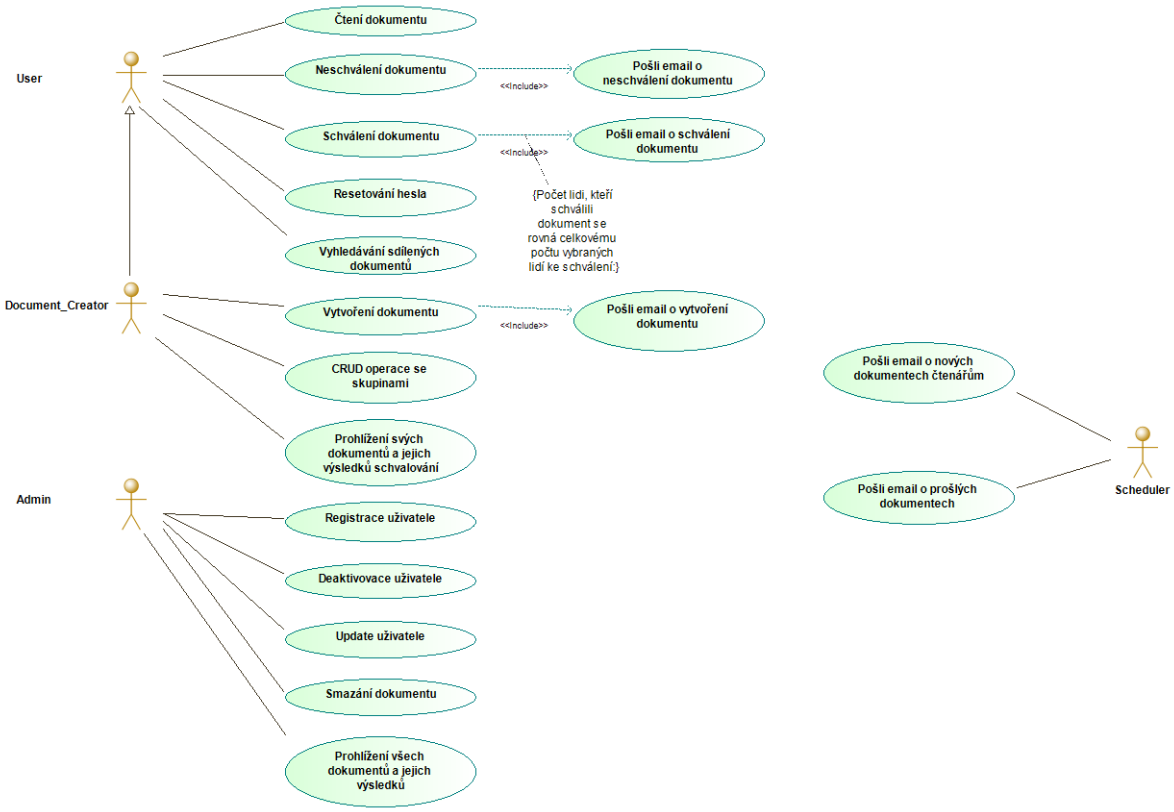
Ve druhé fázi bakalářské práce je cílem provést zhodnocení zabezpečení vytvořené

webové aplikace pomocí penetračních testů.

2. Analytická část

2.1 Analýza aplikace

2.1.1 UML use case diagram



Obrázek 1: UML use case diagram

Na obrázku 1 je zobrazen UML use case diagram. Jsou používáni celkem čtyři aktoři – **User**, **Document_Creator**, **Admin** a **Scheduler**.

Aktor **User** může číst, schvalovat a neschvalovat dokumenty, které s ním byly sdíleny. Operace čtení je uživateli zpřístupněna, pokud je zařazen tvůrcem dokumentu mezi uživatele, kteří jsou povinni se s daným dokumentem seznámit. Uživatel má v tomto případě obeznámení s daným dokumentem potvrdit.

Podobně je tomu i u schvalování a neschvalování dokumentu. Aktorovi **User** je zpřístupněna možnost schvalování dokumentu za předpokladu, že je zařazen tvůrcem dokumentu mezi uživatele, kteří jsou pověřeni daný dokument schválit. Pokud uživatel s dokumentem nesouhlasí, spustí se operace zablokování dokumentu a proběhne odeslání

emailu všem uživatelům, kteří jsou pověřeni schvalováním dokumentu, se zprávou, že daný dokument nebyl schválen a že se stává neaktivním. Pokud uživatel dokument schválí, mohla by se spustit akce odeslání emailu všem uživatelům, kteří jsou pověřeni k procesu schvalování dokumentu se zprávou, že daný dokument byl úspěšně schválen. Tato akce je spuštěna jen za předpokladu, že dokument byl již schválen všemi pověřenými uživateli. Za předpokladu, že tato podmínka není splněna, nedojde k odeslání emailů.

Aktorovi **User** je umožněno dohledávat zpětně dokumenty pomocí globálního vyhledávání podle titulu nebo pomocí historie, kde musí uživatel zadat měsíc a rok. Budou zobrazeny dokumenty, které byly sdíleny s daným uživatelem v zadaném časovém období.

Další funkcí, která je aktorovi **User** zpřístupněna je změna hesla za předpokladu, že je poskytnuto aktuální heslo dvakrát ve správné formě, stejně tak nové heslo musí být zadáno dvakrát stejně. Následně proběhne operace, kde je zkontrolováno, že aktuální heslo je správné a že aktuální a nově chtěné heslo nejsou totožné.

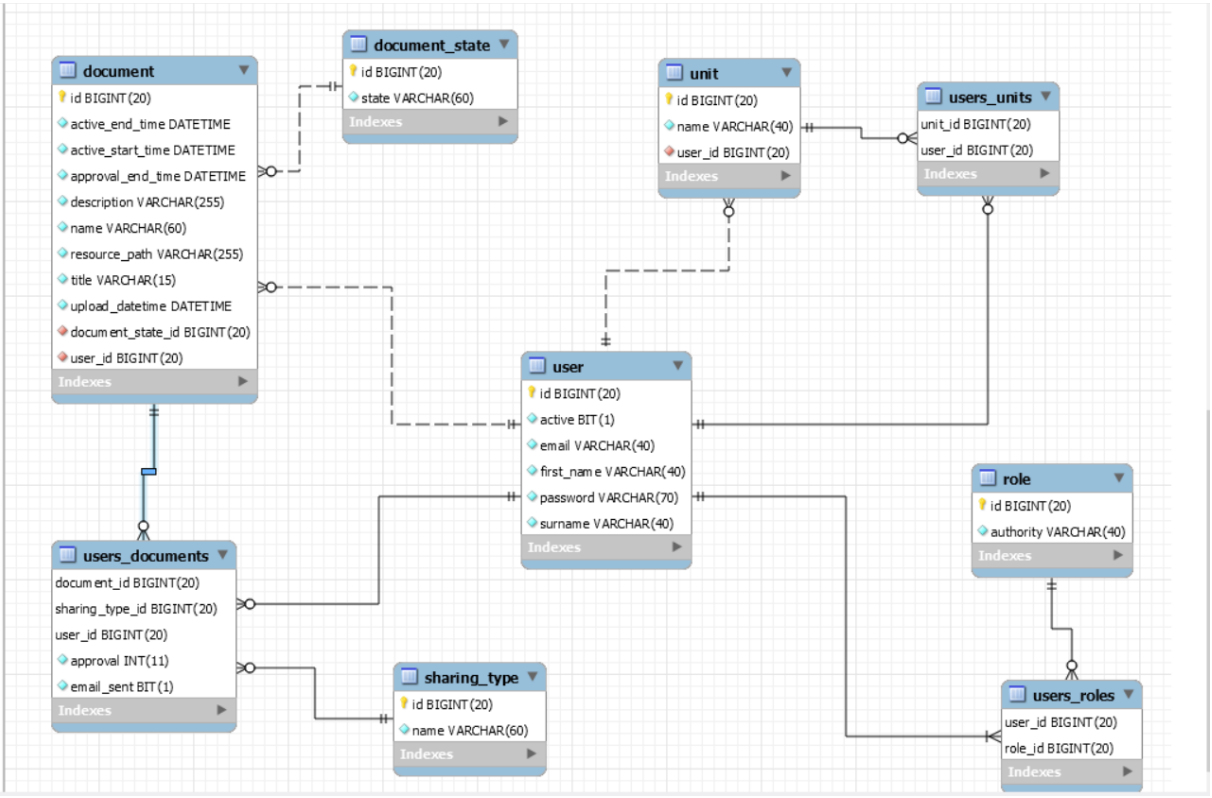
Aktor **Document_Creator** má stejné možnosti jako aktor **User** a navíc může spravovat vlastní skupiny, tvořit dokumenty, prohlížet jím vytvořené dokumenty a výsledky schvalování. Výsledky obsahují informace o tom, kdo s dokumentem souhlasil, kdo nesouhlasil a kdo se prozatím nevyjádřil. Pokud je dokument již schválen, lze dohledat všechny uživatele, kteří dokument přečetli a kteří prozatím ne.

Aktor **Admin** může deaktivovat/aktivovat uživatele. Pokud je provedena operace deaktivace, uživatel je odstraněn ze všech skupin, nedostává již žádné informace prostřednictvím emailové komunikace a v seznamu uživatelů je dohledatelný pouze aktorem **Admin**. Tento aktor může registrovat nové uživatele, aktualizovat informace o uživatelích a mazat dokumenty. Je mu také umožněno čtení všech dokumentů a jejich výsledků.

Aktor **Scheduler** se stará o posílání emailů ohledně expirovaných dokumentů, u kterých doba ke schválení uplynula. Tyto emaily jsou směrované jen uživatelům, kteří byli pověřeni schvalováním dokumentu. Další funkcí tohoto aktora je odesílání emailů ohledně nových zveřejněných dokumentů čtenářům.

2.1.2 Relační model databáze

Jako systém pro řízení dat je používán MySQL. Na obrázku 2 lze vidět relační model používané databáze.



Obrázek 2: Relační model databáze

Hlavní tabulka v relačním modelu je **user**. Tato tabulka uchovává informace o uživatelích. Tabulka obsahuje unikátní klíč *email*, protože je požadováno, aby *email* byl unikátní hodnotou. Z modelu lze vidět, že uživatel může mít více rolí. K tomuto účelu slouží vazba M:N mezi tabulkou **user** a **role**. Tabulka **users_roles** používá jako primární klíč kombinaci atributů *user_id* a *role_id*. Cizí klíče jsou použity také atributy *user_id* a *role_id*. Používané role lze vidět na obrázku 3.

id	authority
1	USER
2	DOCUMENT_CREATOR
3	ADMIN

id	state
1	Approved
2	Approval
3	Cancelled

id	name
1	approver
2	reader

Obrázek 3: Používané role, stavy a položky v tabulce sharing_type

Uživatel může vytvářet své vlastní skupiny. Všechny skupiny, které uživatel vytvořil jsou uloženy v tabulce **unit**. Mezi tabulkou **user** a **unit** je vazba 1:N. Tabulka **users_units** slouží k ukládání uživatelů ve skupinách. K tomuto účelu je zde použita vazba M:N mezi tabulkou user a unit. Primárním klíčem je zde *user_id* a *unit_id*. Cizí klíče jsou použity *user_id* a *unit_id*. Možnost tvorby dokumentů je znázorněno vazbou 1:N mezi tabulkou **user** a **document**. Dokumenty se neukládají přímo do databáze, je zde pouze uložena cesta k dokumentu na disku v atributu *resource_path*. Atribut *upload_datetime* říká, kdy byl dokument nahrán a zároveň kdy začal proces schvalování. Proces schvalování trvá do data a

času, který je udán v atributu *approval_end_time*. Pokud dojde ke schválení dokumentu, bude vybraným uživatelům dokument k dispozici podle hodnot atributů *active_start_time* a *active_end_time*. Atribut *name* je zde unikátní klíč.

Důležitá je tabulka **document_state**. Tato tabulka indikuje, v jakém stavu se dokument nachází. Jsou používány celkem tři stavy zobrazené na obrázku 3 – *Approved*, *Approval* a *Cancelled*. Pokud je dokument ve stavu *Approved*, znamená to, že dokument je schválen vybranými uživateli a je/bude dostupný uživatelům, kteří jsou vybráni daný dokument přečíst a potvrdit v určitém časovém horizontu. Pokud je dokument ve stavu *Approval*, čeká se na schválení od všech pověřených uživatelů. Za předpokladu, že všichni vybraní uživatelé s dokumentem souhlasili, je stav dokumentu nastaven na *Approved*. Posledním možným stavem je *Cancelled*. Tento stav může nastat dvěma způsoby. Buď jeden z vybraných uživatelů dokument zamítne, nebo jej všichni pověření uživatelé v daném časovém horizontu neschválí.

Dokumenty, které byly sdíleny s uživatelem, lze dohledat v tabulce **users_documents**. Je zde použita vazba M:N mezi tabulkou **user** a **document**. Primární klíč je zde tvořen z atributů *document_id, user_id* a *sharing_type_id*. Je zde používán atribut *email_sent*, který slouží k indikaci, zda byl již uživateli s id *user_id* odeslán email o dostupnosti dokumentu s id *document_id* v aplikaci. Atribut *approval* slouží k uchovávání uživatelova souhlasu/nesouhlasu a potvrzení. Hodnota 2 indikuje, že se uživatel ještě nevyjádřil k tomuto dokumentu. Hodnota 1 říká, že uživatel souhlasí s daným dokumentem, nebo že daný dokument potvrzuje. A hodnota 0 znamená, že uživatel nesouhlasí s daným dokumentem.

Tabulka **sharing_type** obsahuje celkem dvě položky, jak lze vidět na obrázku 3 – *reader* a *approver*. Položka *reader* slouží k označení uživatelů, kteří jsou vybráni ke čtení dokumentu. Ten je dostupný ke čtení až po schválení dokumentu pověřenými uživateli. Pokud dokument není schválen, uživatelé označeni jako *reader* nebudou mít nikdy možnost tento dokument přečíst. Druhá položka *approver* slouží k označení uživatelů, kteří jsou ke schválení daného dokument v daném časovém horizontu pověřeni.

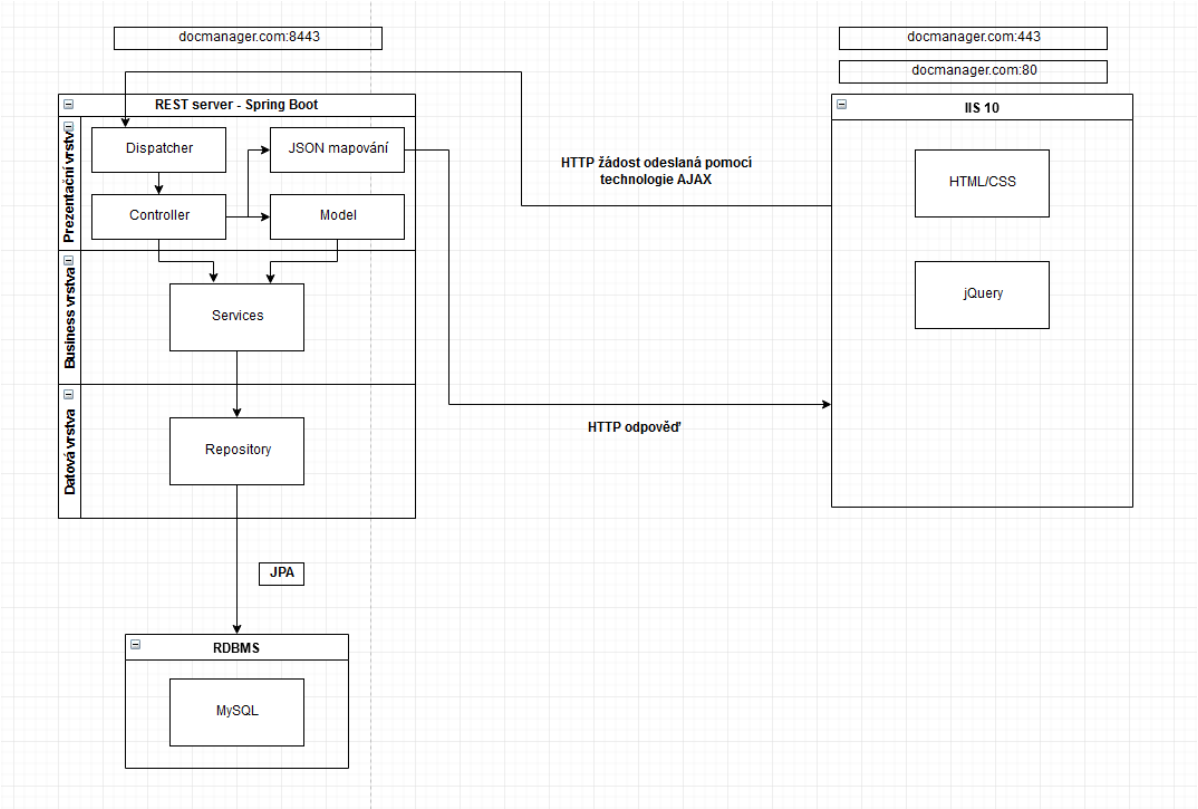
2.1.3 Architektura aplikace

Aplikace je tvořena ze dvou částí. První část aplikace používá framework Spring Boot a programovací jazyk Java 12. Tato část aplikace vystavuje API v podobě REST služeb. Druhá část aplikace používá framework jQuery a komunikuje se serverem pomocí technologie AJAX.

Také je používán webový server IIS verze 10 na portech 443 a 80, na kterém je

vykonávána klientská část aplikace. Serverová část aplikace běží na webovém serveru Apache Tomcat na portu 8443, je podporována komunikace pouze prostřednictvím HTTPS a je rozdělena do tří vrstev – prezentační, business a datové.

V prezentační vrstvě se nachází **Dispatcher,** který implementuje návrhový vzor Front controller a je tedy zodpovědný za směřování příchozích HTTP žádostí ke správným kontrolérům. Do této vrstvy je zařazena i část aplikace vyvíjena dle návrhového vzoru MVC (**Model**, **JSON mapování** a **Controller**). V business vrstvě se nacházejí **Services**. Datová vrstva je zodpovědná za komunikaci s databází pomocí ORM techniky. K tomuto účelu je používán standard JPA.



Obrázek 4 Architektura aplikace

2.2 Bezpečnostní mechanismy

Následující kapitola se věnuje rozboru bezpečnostních mechanismů, které jsou využívány.

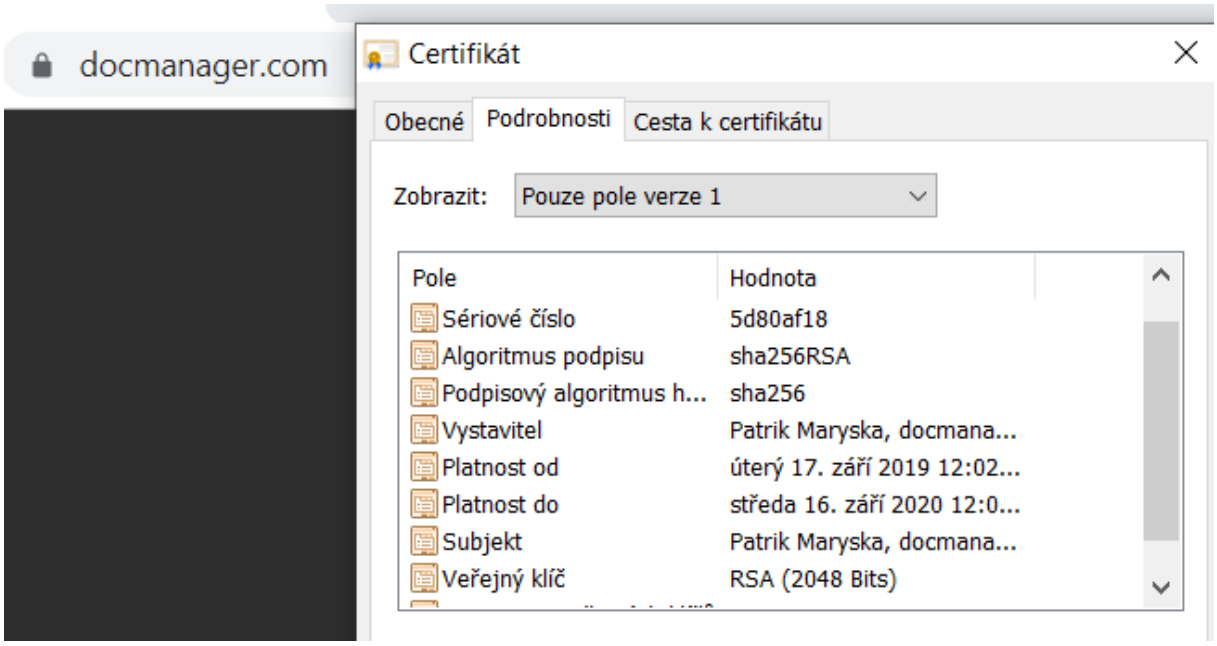
2.2.1 HTTPS

HTTPS hraje velmi důležitou roli ve spojení se zabezpečením webových aplikací. Umožnuje zajistit bezpečné přenosy dat mezi počítači na internetu. Využívá protokol HTTP spolu s protokolem SSL, nebo TLS [3]. Vzhledem k velkému počtu obchodních transakcí na

internetu je kryptografie velmi důležitým článkem k zajištění bezpečnosti transakcí. V roce 2015 proběhlo přes 38,5 bilionů obchodních transakcí na internetu [1].

REST server komunikuje pouze pomocí HTTPS. Jsou zakázané všechny nižší protokoly než TLS1.2. a je zakázáno komunikovat pomocí slabých šifer [2]. Starší protokoly nejsou bezpečné [21]. Je používán certifikát sám sebou podepsaný k šifrované komunikaci, který používá algoritmus SHA256. Tento certifikát byl vytvořen pomocí nástroje *KeyToolExplorer*. K funkčnosti aplikace je nutné, aby tento certifikát byl vložen mezi důvěryhodné certifikační autority.

Webový server IIS používá stejný certifikát jako REST server. Je také zakázáno komunikovat pomocí slabých šifer a je povolena komunikace pouze přes TLS1.2 [2]. Je umožněno přistupovat k webovému serveru IIS přes HTTP i HTTPS, proto je používána hlavička Strict-Transport-Security.



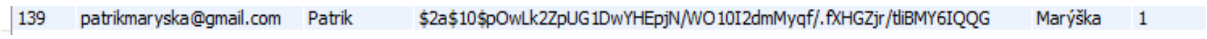
Obrázek 5: Certifikát serveru

2.2.2 Heslo

Hesla jsou jednoznačně nejpopulárnější způsob potvrzení identity webové aplikaci. Z toho důvodu jsou velmi populárním cílem útočníků. Všechny útoky proti heslům jsou založeny na opakovaném hádání hesla. Cílem je uhádnout hodnotu hesla v textové formě [3].

Heslo je uloženo ve formě výsledku hashovací funkce bcrypt do databáze. Velkou výhodou této funkce je, že interně generuje náhodný kus dat, který je přidán na vstup hashovací funkce [3]. Zvýšit zabezpečení můžeme tím, že použijeme hashovací funkci

několikrát za sebou [3]. Je možné dosadit, kolikrát se bude daná funkce opakovat (rounds). Čím větší hodnota, tím déle trvá hashování hesla. Spring používá *bcrypt* s výchozí hodnotou deset [17].



Obrázek 6: Uložené heslo v databázi

Délka hesla a použitá sada má vliv na počet variací, které musí útočník vyzkoušet, aby se dostal ke správnému výsledku.

Je vyžadováno, aby hesla obsahovala alespoň osm znaků, minimálně jeden speciální znak, jedno velké písmeno, jednu číslici a jedno malé písmeno.

Lze to udělat ještě složitější. Pokud budeme mít heslo, které je dostatečně komplexní a budeme vyžadovat, aby došlo ke změně hesla pouze v daných časových intervalech, útočníkova šance na prolomení hesla bude ještě menší [3]. Aplikace neumožňuje změnu hesla v daných časových intervalech. Změna hesla je umožněna kdykoli.

Aplikace má implementovanou obranu proti útokům hrubou silou. Pokud dojde k pěti neúspěšným přihlášením v řadě z jedné IP adresy, dojde k zablokování dané IP adresy na čtyři hodiny.

2.2.3 Oauth2

Tento protokol umožňuje aplikacím třetím stran získat omezený přístup k HTTP službě. Povolení k přístupu je žádáno klientem. Klient může být webová stránka, mobilní aplikace a další [16].

Jsou používány dva typy grantů – *password* a *refresh_token*. Typ grantu *password* umožňuje výměnu uživatelových přihlašovacích údajů za přístupový token [16]. Typ *refresh_token* je používán klienty k výměně obnovovacího tokenu za nový přístupový token, když platnost aktuálního přístupového tokenu vyprší [16].

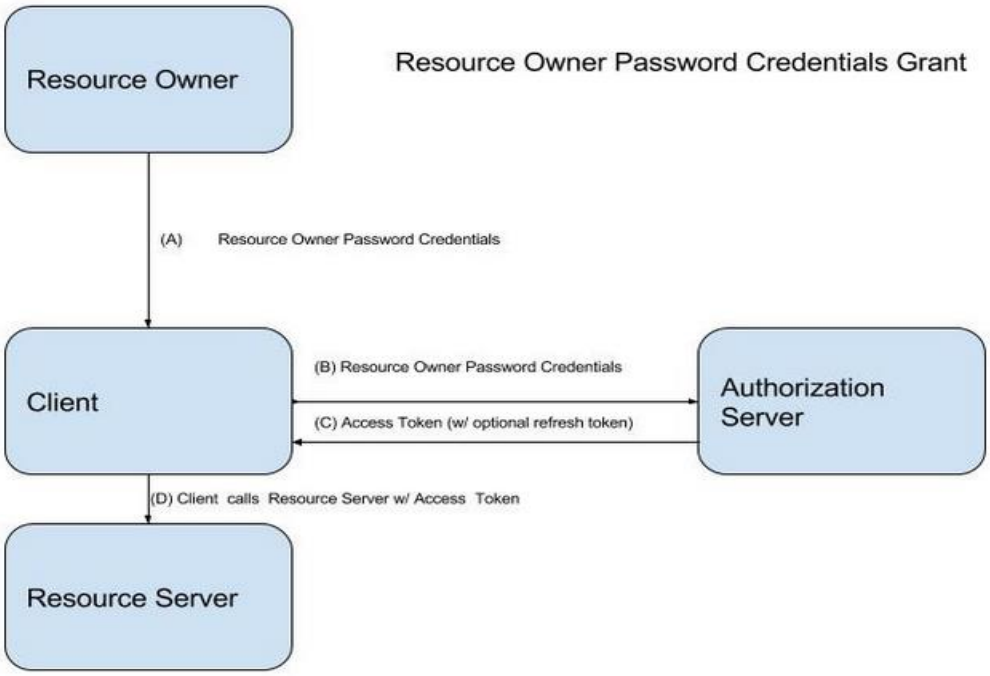
Je tedy používán přístupový token a obnovovací token v aplikaci. Přístupové tokeny jsou používané k přístupu k chráněným zdrojům. Přístupový token je řetězec, který reprezentuje autorizaci vydanou klientovi [16]. Má omezenou životnost, která je definovaná autorizačním serverem. V aplikaci je definována životnost přístupového tokenu na patnáct minut.

Obnovovací tokeny jsou používány k získání přístupových tokenů. Jsou vydány klientovi autorizačním serverem a jsou určeny k získání nového přístupového tokenu, za předpokladu, že stávající klientův přístupový token se stane neplatným. Jsou používané pouze pro komunikaci s autorizačními servery a nejsou nikdy posílány na resource server [16].

Životnost obnovovacího tokenu je definovaná na třicet minut v aplikaci. Na obrázku 7 je zobrazen tento proces. **Resource Owner** je v případě této aplikace uživatel. **Client** je aplikace, která se snaží získat přístup k uživatelskému účtu.

Je využíván standard JSON Web Token (JWT). JWT je otevřený standard, který definuje způsob, jak bezpečně přenášet informace mezi stranami ve formátu JSON. Tyto informace mohou být ověřeny a důvěřovány, protože jsou digitálně podepsané. JWT může být podepsáno pomocí tajemství nebo asymetricky pomocí RSA nebo ECDSA [10]. JWT tokeny by měly být přikládány do žádostí posílané na server tak, že se přidá hlavička *Authorization: Bearer [JWT token]*.

Aplikace digitálně podepisuje JWT token na autorizačním serveru pomocí privátního klíče a algoritmu RSA256. K tomuto účelu byl vygenerován certifikát podepsaný sám sebou typu JKS pomocí nástroje *KeyToolExplorer*. K ověření validity JWT tokenu je používán veřejný klíč na straně resource serveru. Je vráceno chybové hlášení za předpokladu, že dojde ke kompromitaci JWT.



Obrázek 7: Password Credentials Flow [23]

2.2.4 XSS

XSS je útokem, který je v dokumentu OWASP Top 10 z roku 2017 na sedmém místě [5]. Celkem aplikace může čelit třem typům XSS – reflected, stored a DOM based.

Reflected XSS se objevuje, když aplikace obdrží data, která obsahují útočníkův script v

HTTP žádosti a následně aplikace vkládá tato data nezabezpečeně do okamžité odpovědi. Tento útočný script není uložen v aplikaci a pouze postihuje uživatele, kteří otevřou nebezpečný vytvořený odkaz nebo jinou webovou stránku [8].

Stored XSS nastává, když webová aplikace přijímá vstup od uživatele, který může být potencionálně nebezpečný a následně tento vstup ukládá pro pozdější použití. Pokud tato data nejsou správně filtrována, tak tato nebezpečná data se zobrazí jako součást webové stránky a za předpokladu, že obsahují nějaký škodlivý script, se spustí v prohlížeči jiných uživatelů s právy webové aplikace [8].

Útok DOM based XSS je velmi podobný útoku Stored XSS. Rozdíl je, že dochází k zakomponování útočníkova skriptu do webové stránky na straně klienta, nikoli na straně serveru [8].

Jsou kontrolována všechna příchozí data na frontendu a jsou převáděny potencionálně nebezpečné znaky na HTML entity. Potencionálně nebezpečné znaky jsou amspersand, dvojité uvozovky, levá/pravá ostrá závorka, apostrof a zakódované nebezpečné znaky. Tyto převody jsou uskutečněny voláním správné funkce používané knihovny *xss-filters.js*.

Je využíváno několik bezpečnostních hlaviček, které jsou nutností v prvotní obraně proti útokům typu XSS.

- X-XSS-Protection:1; mode=block – tato hlavička povoluje XSS filtrování a pokud je detekován útok XSS, prohlížeč zablokuje načtení stránky [4].
- Content-Type-Options: nosniff – tato hlavička sděluje prohlížeči, aby ověřoval, zda je správně nastavený MIME type v hlavičce zdroje [11].
- Content-Security-Policy – pomocí této hlavičky je určena politika, odkud se mohou které zdroje nahrávat [5].

2.2.5 Autorizace

Autorizace je prováděna na základě přidělených rolí. Role uživatele se po autentizaci nacházejí v přístupovém tokenu a dle těchto rolí je rozhodnuto, zda uživatel je oprávněn přistoupit k danému end-pointu. V aplikaci je určeno, která URL vyžadují, aby byl uživatel autentizován. Používané role jsou rozebrány v kapitole 2.1.2 a komplexnější rozbor autorizačního systému je popsán v kapitole 3.2.4.

2.2.6 Validace inputu

Validace vstupních dat je nesmírně důležitým obranným mechanismem. Mělo by platit pravidlo, že by program nikdy neměl důvěřovat datům, které přichází od uživatele. Validace

těchto dat je tedy nesmírně kritická funkcionalita bezpečné aplikace.

Prvotní validace je provedena již na frontendu. Tuto validaci lze velmi snadno obejít, proto validace na serveru musí být více důkladná. Validace vstupu je uskutečněna několika způsoby. První způsobem je použití různých regulárních výrazů k filtraci vstupu. Druhým způsobem je použití různých anotací, které nabízí Spring framework. Pomocí těchto anotací lze přesně definovat minimální a maximální délku, povolené znaky, jak by měla vypadat zpráva vrácena uživateli o chybě a další varianty. Aplikace používá různé další funkce, které se starají o validaci inputu. Více je toto téma rozebráno v kapitole 3.2.12.

2.2.7 Šifrování dokumentů na disku

Příchozí dokumenty jsou šifrovány pomocí symetrického klíče a algoritmu AES. Velikost použitého klíče je 256 bitů. Tyto dokumenty jsou po zašifrování uloženy na disk.

Je nutné, aby k tajnému klíči měl přístup pouze uživatel, pod kterým poběží aplikace v rámci operačního serveru a administrátor. Ostatní uživatelé nesmí mít přístup k tajnému klíči.

2.2.8 SQL injection

Útok typu injection je v žebříku OWASP Top 10 z roku 2017 na prvním místě [6]. Velmi kritickou chybou je tvoření SQL dotazů pomocí spojování řetězců, nepoužívání parametrizovaných dotazů a špatná validace vstupu [6].

Prvotním obranným mechanismem proti SQL injection je validace vstupu. Druhotným mechanismem použitým v této aplikaci je používání parametrizovaných dotazů při komunikaci s RDBMS.

2.2.9 Cookies

Je používána technologie Oauth2 s tokeny místo cookies. Cookies jsou na serveru vypnuty. Není tedy nutné řešit útok typu CSRF.

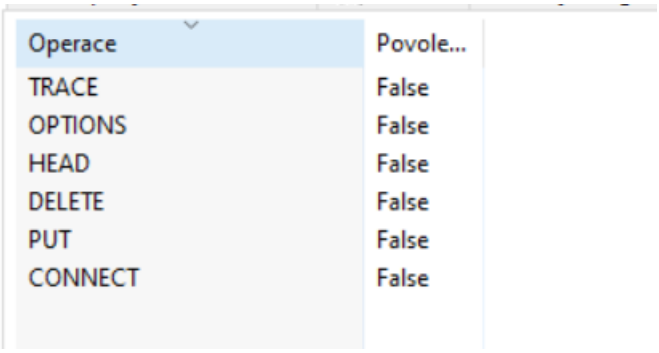
2.2.10 Logování

Jsou logovány úspěšné/neúspěšné pokusy o přihlášení a odhlášení do souboru *login.log*. Dále jsou logovány operace s dokumenty – tvorba dokumentu, úspěšný/neúspěšný přístup k dokumentu, smazání dokumentu a úspěšné/neúspěšné schválení dokumentu do souboru *doc_logs.log*. Do souboru *users.log* jsou logovány operace s uživateli a jejich skupinami –

smazání, vytvoření a aktualizování uživatelů a skupin. Automaticky jsou vytvářeny každý den nové logovací soubory pro tyto tři skupiny. Do souboru *logfile.log* se zapisují globální chyby.

2.2.11 Povolené metody

Na webovém serveru IIS jsou povoleny metody GET a POST. Ostatní metody jsou zakázány, jak lze vidět na obrázku 8.



Operace	Povole...
TRACE	False
OPTIONS	False
HEAD	False
DELETE	False
PUT	False
CONNECT	False

Obrázek 8: Zakázané metody na webovém serveru IIS

Server potřebuje pro svoji správnou funkcionalitu metody GET, POST, DELETE, PUT a OPTIONS z důvodu, že je používán CORS standard.

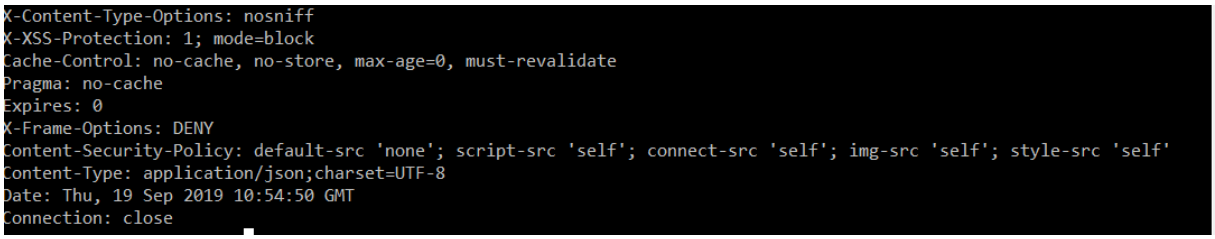
2.2.12 CORS

CORS standard funguje přidáním nových HTTP hlaviček, které nechají server definovat, odkud je povoleno číst informace ze serveru. Funkčnost technologie AJAX je omezena zásadou stejného původu. To znamená, že stránka musí pocházet ze stejného serveru jako server a zároveň musí mít totožný port i protokol [7].

Klíčovou roli zde hrají předběžné požadavky metody OPTIONS. Prohlížeč posílá tento požadavek na server obsahující speciální hlavičky. Prohlížeč se nejdříve zeptá serveru, zda by poskytl dané aplikaci data a na základě této odpovědi prohlížeč o data zažádá. Odpověď serveru může být tvořena z mnoha hlaviček. Nejdůležitější jsou hlavičky, které definují, odkud je povoleno číst informace ze serveru, jaké metody lze použít, jak dlouho se má uchovávat v paměti výsledek předběžného požadavku a jaké hlavičky jsou povoleny. Je možné definovat i další hlavičky, ale aplikace je nepoužívá [7].

2.2.13 Bezpečnostní hlavičky na REST serveru

REST server používá hlavičky ukázané na obrázku 9. Prozatím nevysvětlené hlavičky jsou vysvětleny v kapitole 3.4.



Obrázek 9: Hlavičky REST serveru

2.3 Penetrační testy

Penetrační test je metoda hodnocení zabezpečení počítačových zařízení, kde bezpečnostní expert se snaží najít a zneužít zranitelností v počítačovém systému. Cílem tohoto testu je identifikovat slabá místa v aplikaci, která by mohla být zneužita potencionálním útočníkem.

K testování byl použit dokument OWASP Testing Guide verze 4. Cílem těchto testů je porovnání stavu aplikace oproti řadě definovaných kritérií [8].

Tyto testy jsou rozděleny do několika kategorií. Každá kategorie obsahuje několik testů. Tento dokument popisuje celkem jedenáct kategorií.

1 – Testování sběru informací (OTG-INFO)

Tato sada testů se zaobírá porozuměním konfigurace serveru provozující webovou aplikaci. Aplikační platformy jsou různorodé, ale některé klíčové konfigurační chyby mohou poškodit aplikaci [8]. Je testován únik informací, otisk webového serveru, otisk používaného webového frameworku a otisk webové aplikace. Dále je testována identifikace slabých vstupních bodů v aplikaci a struktura aplikace [8].

2 - Testování správy konfigurace (OTG-CONFIG)

Tyto testy jsou také určené k porozumění konfigurace serveru provozující webovou aplikaci [8]. Jedná se o konfiguraci sítě, konfiguraci aplikační platformy, zpracování souborů s citlivými informacemi, procházení souborů s citlivými informacemi, výčet infrastruktury a administračních rozhraní, testování HTTP metod, testování HSTS a RIA [8].

3 – Testování správy identit (OTG-IDENT)

Tato kategorie se zabývá testováním definice rolí, procesu registrace uživatele, procesu poskytování účtu, hádání uživatelského jména a slabé nebo nevynucené politiky uživatelského jména [8].

4 – Testování autentizace (OTG-AUTHN)

Autentizace je proces ověření digitální identity odesílatele komunikace. Testování autentizačního schématu je založeno na zjištění, jak autentizační proces funguje a získané informace jsou použity k obejití autentizačního mechanismu. Testuje se, zda přihlašovací údaje jsou posílány přes šifrovaný kanál, zda aplikace používá výchozí přihlašovací údaje, slabý zamykací mechanismus, obcházení autentizačního schématu, zranitelnost remember me, zranitelnost cache prohlížeče, slabá politika hesla, slabá bezpečnostní otázka, slabá změna hesla a slabší autentizace přes alternativní kanál [8].

5 – Testování autorizace (OTG-AUTHZ)

Autorizace je koncept povolení přístupu ke zdrojům pouze těm, kteří mají povolení tyto zdroje využívat. Testování autorizace je založeno na porozumění, jak autorizační proces funguje a získané informace jsou použity k obejití autorizačního mechanismu. Toto testování zahrnuje procházení adresářů, obcházení autorizačního schématu, eskalaci privilegií a nezabezpečené přímé odkazy na objekt [8].

6 – Testování řízení relace (OTG-SESS)

V těchto testech, tester kontroluje, zda cookies a jiné session tokeny jsou vytvářeny bezpečně a v neočekávané formě. Útočník, který je schopen odhadnout a vytvořit zranitelné cookies může velmi snadno získat relace legitimních uživatelů [8].

Tyto testy se zaměřují na obcházení session schématu, cookies atributy, session fixaci, vystavené session proměnné, CSRF, funkcionalitu odhlášení, vypršení session a session puzzling [8].

7 - Testování validace vstupu (OTG-INPVAL)

Nejvíce častá chyba webový aplikací je špatná validace vstupu. Tato zranitelnost vede k dalším vážným zranitelnostem ve webových aplikacích. Datům od externí entity nebo klienta by nemělo být nikdy důvěřováno. Cílem testera je testování všech možných cest vstupu k zjištění, zda aplikace dostatečně kontroluje vstupy [8].

Tato kategorie testuje Reflected XSS, Stored XSS, HTTP protokol, znečištění HTTP parametrů, SQL injection, LDAP injection, ORM injection, XML injection, SSI injection, XPath injection, IMAP/SMTP injection, Code injection, Command Injection, Buffer overflow, inkubované zranitelnosti a HTTP splitting/smuggling [8].

8 – Testování, jak aplikace pracuje s chybami (OTG-ERR)

Tato kapitola zahrnuje testy zaměřené na reakce na chyby a Stack Traces [8].

9 - Testování slabé kryptografie (OTG-CRYPST)

Týká se testování slabých SSL/TLS šifer, nedostatečné TLS konfigurace, padding oracle a zda nejsou posílané citlivé informace přes nešifrované kanály [8].

10 - Testování business logiky (OTG-BUSLOGIC)

V této kategorii se testuje validace business dat, zda je možné padělat žádosti, kontrola integrity, časování procesu, zda jsou dodržovány limity na spuštění funkce, obcházení pracovních postupů, obrana proti špatnému používání aplikace, nahrání neočekávaných typů souborů a infikovaných souborů [8].

11 - Testování klientské strany (OTG-CLIENT)

Testování klientské strany je spojeno se spuštěním kódu na straně klienta, typicky v prohlížeči uživatele [8].

Testuje se DOM based XSS, spuštění javascriptového kódu, HTML injection, přesměrování URL, CSS injection, manipulace se zdroji, CORS, Cross site flashing, Clickjacking, WebSockets, Web Messaging a lokální úložiště [8].

Dodatečné testy zaměřené na REST server byly testovány s pomocí OWASP REST Security Cheat Sheet [9].

Vlastní testy

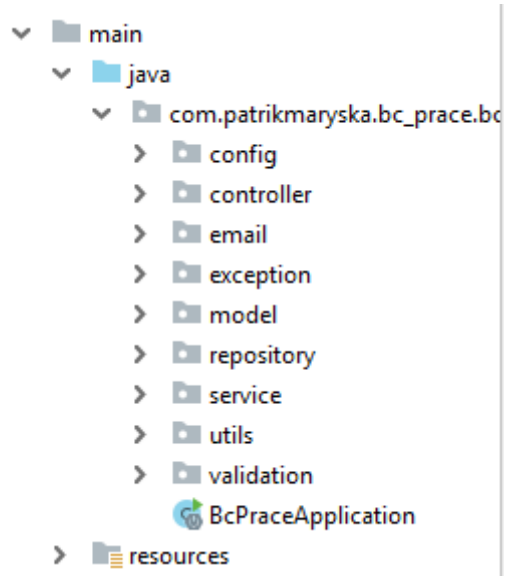
Byly vytvořeny vlastní testy k otestování dalších bezpečnostních mechanismů, které nebyly pokryty předchozími testy. Jedná se o testy, které testují životnost tokenů a odhlášení.

3. Implementační část

Tato kapitola je rozdělena do tří částí. V první části dojde k popisu funkcionality aplikace. Ve druhé části bude popsána implementace bezpečnostních mechanismů. Ve třetí částí bude vysvětlena bezpečná konfigurace webového serveru IIS.

3.1 Funkcionalita aplikace

3.1.1 Struktura aplikace



Obrázek 10: Používané balíčky na REST serveru

Na obrázku 10 lze vidět balíčky, které jsou používány na serveru. Balíček **config** obsahuje konfigurační bezpečnostní třídy. Balíček **controller** obsahuje třídy, které jsou označeny pomocí anotace *@RestController*. Jak již anotace napovídá, jedná se třídy, které fungují jako REST kontroléry. Tyto třídy vystavují různé end-pointy a starají se o obsluhu příchozích požadavků. Jsou používané celkem čtyři kontroléry – *DocumentController*, *UserController*, *UnitController* a *OauthController*. *OauthController* se stará o odhlašování uživatelů a má k dispozici end-point, na kterém lze ověřit validitu přístupového tokenu. Třída *DocumentController* vystavuje end-pointy, které souvisí s dokumenty. *UserController* pracuje s uživateli a *UnitController* se skupinami.

Další důležitý balíček je **service**. Ten obsahuje třídy, které jsou označeny anotací *@Service*. Jedná se o vrstvu, s kterou komunikují kontroléry. Servisní vrstva zpracovává data, která jsou předaná kontrolérem. Zvláštními případy jsou třídy *UserDetailsService*, *LoginAttemptService a SchedulerService*. *UserDetailsService* se stará o autentizaci. Vysvětlení této třídy a její implementace je popsána v kapitole 3.2.1. *LoginAttemptService* slouží k obraně proti útokům hrubou silou. Vysvětlení této třídy je popsáno v kapitole 3.2.15. Velmi důležitá je třída *SchedulerService*, která má na starosti plánování. Servisní třídy po zpracování dat komunikují s třídami označenými *@Repository*. Tyto třídy se nacházejí v balíčku *repository*. Jejich úkolem je komunikace s RDBMS.

Balíček **model** obsahuje třídy, které jsou označeny anotací *@Entity*. Je používána ORM

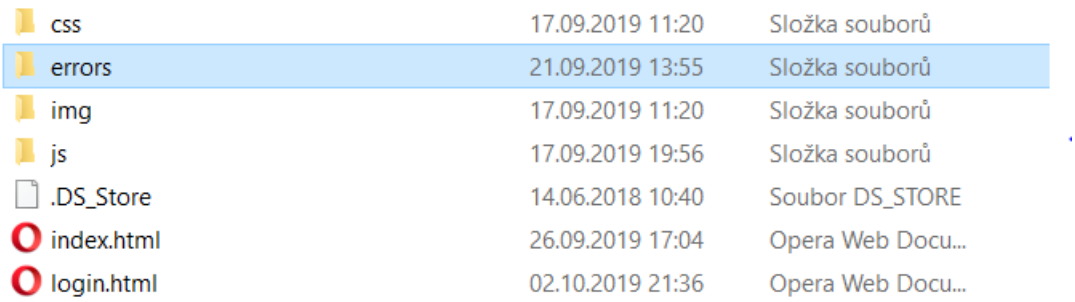
technika pro komunikaci s databází a k tomuto účelu je využíván standard JPA. Třídy v tomto balíčku slouží k uskutečnění ORM. Výjimkou je balíček *RequestBody*, který se nachází v balíčku *model*. Zde se nachází třídy, které slouží k přijímání dat od uživatele, která nejsou zaslaná v URL. Obsahují validační pravidla a potřebné proměnné, aby mohlo dojít k převodu z formátu JSON na Java objekt.

Balíček **exception** obsahuje třídy, které se starají o globální zachycení výjimek. Balíček **validation** obsahuje třídy, které souvisí s obsáhlejší validací dat. Nacházejí se zde třídy, které se starají o politiku hesla. V balíčku **utils** se nacházejí třídy pro specifickou práci s daty. Třída *EncrypterDecrypter* šifruje a dešifruje soubory. Třída *DocumentComparator* slouží ke správnému seřazení dokumentů. Balíček **email** obsahuje třídy, které se starají o posílání emailů.

Třída *BcPraceApplication* je hlavní třída aplikace, která aplikaci spouští.

Spring-Boot má složku *resources*. Zde je uložen konfigurační soubor *application.properties*. Je zde uložen i soubor *application.yml*, ve kterém je uložen veřejný klíč pro šifrování dokumentů. Ve složce *keystore* se nachází certifikát *dm_fin.p12*.

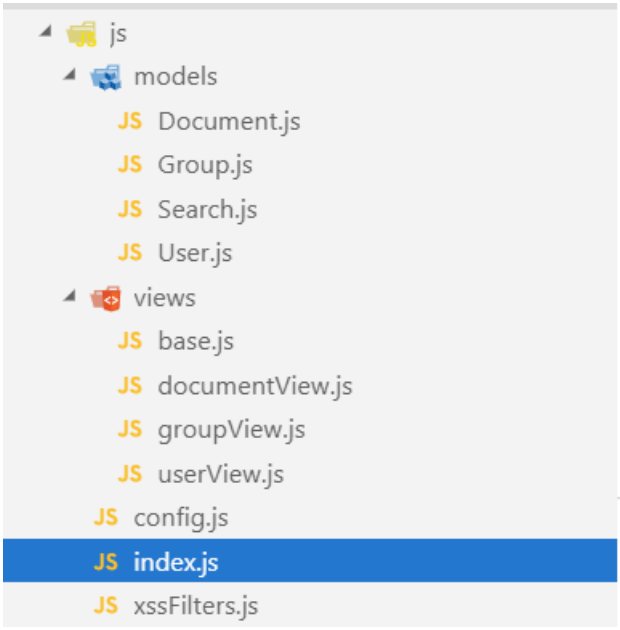
Soubor *pom.xml* obsahuje všechny *dependencies*, které jsou využívány.



Obrázek 11: Frontendová složka

Na obrázku 11 lze vidět potřebné soubory na frontendu. Ve složce *js* se nachází celkem čtyři scripty. Soubor *bundle.js* je hlavní script, který se stará o funkcionalitu po úspěšném loginu. Tento script byl vyvíjen podle architektury MVC, jak lze vidět na obrázku 12.

Na frontendové straně je použit jeden kontrolér a to *index.js*. Soubor *xssFilters.js* je používaná knihovna a *config.js* slouží k základnímu nastavení aplikace. Nachází se zde script *jquery.js*, což je také použitá knihovna. Soubor *login.js* se stará pouze o přihlašovací stránku a *pre-load.js* je script, který se spouští před načtením hlavní stránky *index.html* a posílá dotaz na server, zda je přístupový token validní.



Obrázek 12: Script bundle.js

HTML soubory jsou používány celkem dva – *index.html* a *login.html*. Soubor *index.html* je používán jako hlavní stránka aplikace po úspěšném loginu. Složka *errors* obsahuje stránky, které se zobrazí při určitých chybách, které může vrátit webový server. Složka *css* obsahuje kaskádové styly.

3.1.2 Inicializace

Je vyžadovaná autentizace, aby mohla být aplikace plně využívána. K přihlášení uživatele dochází na stránce *login.html*. Tento proces probíhá způsobem popsaných v kapitolách 3.2.3 a 3.2.1. Předtím, než je načtená hlavní stránka, je zkontrolováno, zda uživatel má opravdu validní token. K tomu účelu slouží script *pre-load.js*, který se dotazuje serveru, zda token je opravdu validní. Tento script je spuštěn v hlavičce HTML souboru. Dotaz lze vidět na obrázku 13. Po potvrzení validity tokenu je dekódován přístupový token a jsou získány informace o přihlášeném uživateli. Proces dekódování lze vidět na obrázku 14. Menu je tvořeno na základě rolí, které jsou zapsané v přístupovém tokenu.

```
function checkTokenValidity(url){
 $.ajax({
    'url': url,
    'type': 'GET',
    'headers': {
      'Authorization': 'Bearer ' + sessionStorage.access_token
    },
    'success': function (result) {
      if(result == "OK"){
        $("body").removeClass("body-initial");
      } else {
        window.location.replace("login.html");
        sessionStorage.clear();
      }
    },
    'error': function (xhr, textStatus, errorThrown) {
      if(xhr.status === 0){
        alert("Could not contact the server. Try it later please.");
        window.location.replace("login.html");
        sessionStorage.clear();
        return;
      }

      if(xhr.status === 401){
        refreshToken();
```

Obrázek 13: Testování validity tokenu

```
const parseJwt = (token) => {
  try {
    return JSON.parse(atob(token.split('.')[1]));
  } catch (e) {
    return null;
  }
};
```

Obrázek 14: Dekódování tokenu

3.1.3 Dokumenty

Po úspěšném přihlášení je uživatel přesunut na stránku *index.html*. Ihned je odeslána žádost na server o nové uživatelovy dokumenty. Žádost je odeslána na end-point */documents* metodou GET způsobem popsaným v kapitole 3.2.1. Pokud je uživatel správně autentizován, dojde ke zpracování požadavku a pomocí JPQL dotazu zobrazeného na obrázku 15 je vrácen JSON s novými dokumenty, které byly sdíleny s daným uživatelem. Tento dotaz se nachází ve třídě *DocumentRepositoryCustomImpl* v balíčku *repository*.

```
TypedQuery<Document> q2 = entityManager.createQuery( s: "SELECT d FROM Document d " +
                "INNER JOIN d.documentsForUsers ud WHERE (d.documentState.id = 2 AND ud.sharingType = 1" +
                " AND ud.user.email=:email" +
                " AND current_timestamp between d.uploadDatetime AND d.approvalEndTime AND ud.approval=2)" +
                " OR(d.documentState.id = 1 AND ud.sharingType.id=2 AND ud.user.email=:email AND current_timestamp " +
                "between d.activeStartTime AND d.activeEndTime AND ud.approval=2)"
        , Document.class);

q2.setParameter( s: "email", email);
```

Obrázek 15: JPQL dotaz pro získání všech sdílených dokumentů s uživatelem

Dokumenty musí být speciálně seřazeny. K tomuto účelu slouží třída *DocumentComparator* v balíčku *utils*. Řazení je ukázáno na obrázku 16.

```
@Override
public int compare(Document d1, Document d2) {
    Date date1 = new Date();
    Date date2 = new Date();

    Optional<UsersDocuments> ud1 = d1.getDocumentsForUsers().stream().
            filter(usersDocuments -> usersDocuments.getUser().getId() == user.getId()).findFirst();
    Optional<UsersDocuments> ud2 = d2.getDocumentsForUsers().stream().
            filter(usersDocuments -> usersDocuments.getUser().getId() == user.getId()).findFirst();

    if(ud1.get().getSharingType().getId() == 1){
        date1 = d1.getUploadDatetime();
    }  else {
        date1 = d1.getActiveStartTime();
    }

    if(ud2.get().getSharingType().getId() == 1){
        date2 = d2.getUploadDatetime();
    }  else{
        date2 = d2.getActiveStartTime();
    }

    return date1.compareTo(date2);
}
```

Obrázek 16: Metoda compare

Je nutné dokumenty seřadit takovým způsobem, aby bylo jasné, v jakém pořadí dokumenty dorazily do aplikace. Pokud je uživatel pověřen ke schválení dokumentu, musí probíhat řazení u tohoto dokumentu podle proměnné *uploadDatetime*. V opačném případě podle proměnné *activeStartTime*.

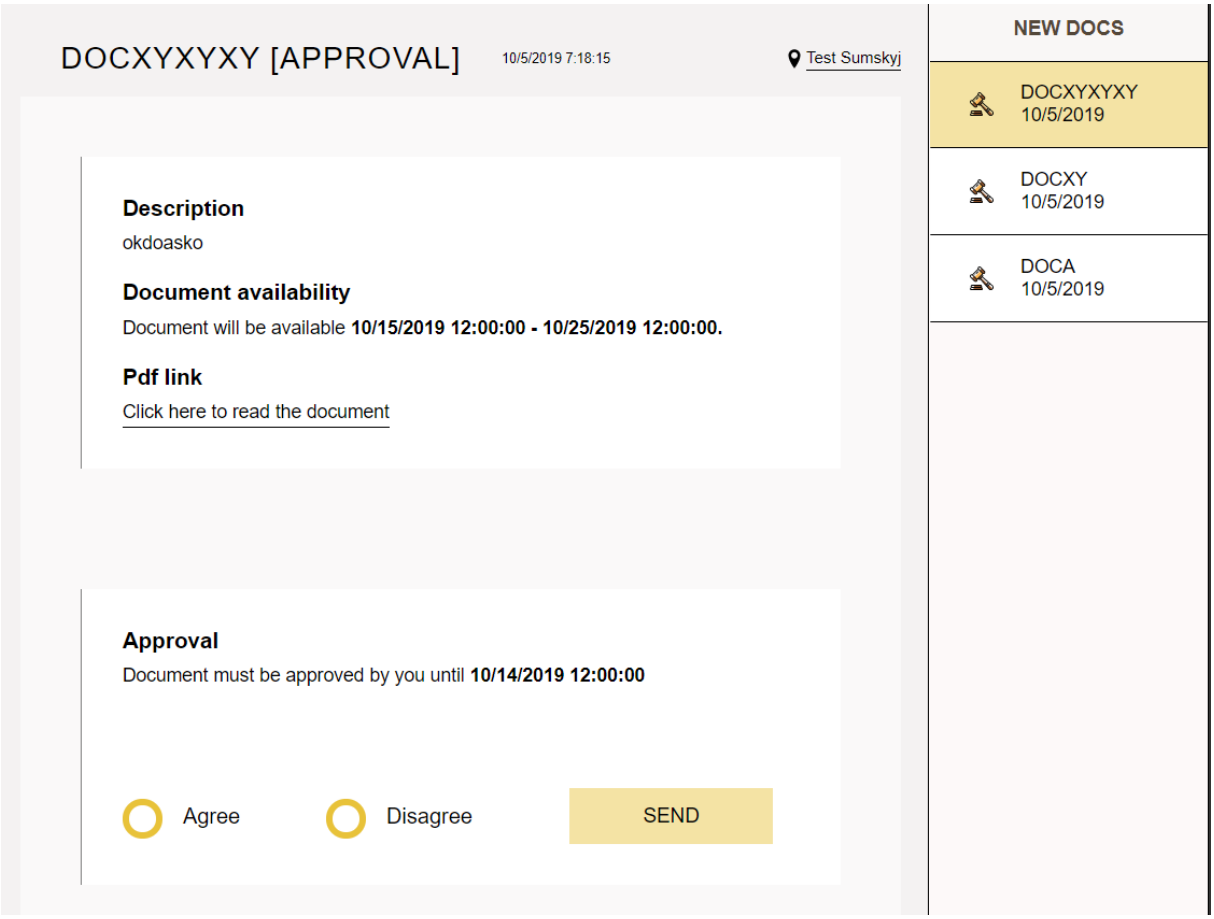
Použití komparátoru je znázorněno na obrázku 17. Na kolekci dokumentů se volá metoda *sort*. V tomto případě je volána i metoda *reversed*, která obrátí řazení.

```
List<Document> dcs = documentRepository.getAllDocsByEmail(email);
User user = userService.getUserByEmail(email).get();

dcs.sort(new DocumentComparator(user).reversed());
```

Obrázek 17: Použití komparátoru

Tato třída je použita pro řazení všech kolekcí, které vrací dokumenty uživateli. V některých případech není použita metoda *reversed*.

Pokud uživatel má k dispozici nové dokumenty, zobrazí se na pravé straně v aplikaci.



Obrázek 18: Načtený dokument k odsouhlasení

Dokument je získán posláním žádosti na end-point */documents/[id dokumentu]* metodou GET. Proces generování PDF dokumentu lze vidět na obrázku 19. Tato metoda je implementována ve třídě *DocumentController* v balíčku *controller*.

```
@GetMapping(value = "/{id}")
@Secured({("ROLE_USER"), ("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public void generateReport(OAuth2Authentication auth, HttpServletResponse response, @PathVariable("id") long id) {
    String email = auth.getName();
    User user = userService.getUserByEmail(email).get();
    Optional<Document> doc = documentService.getDocumentById(id);

    if(doc.isPresent() && documentService.hasUserAccessToDocument(doc.get().getId(), user.getId())){
        try {
            documentService.generateReport(doc.get().getResourcePath(), response);
            docLogger.info("User " + email + " has accessed the document id" + id);
        } catch (NoSuchPaddingException | NoSuchAlgorithmException | IOException | InvalidKeyException | InvalidAlgorithmParameterException e) {
            throw new ResponseStatusException(
                    HttpStatus.INTERNAL_SERVER_ERROR, "Internal server error while generating document.");
        }
```

Obrázek 19: Generování PDF dokumentu

Je nejdříve zjištěno, zda daný dokument existuje a zda k němu uživatel má přístup. Tato metoda je popsána v kapitole 3.2.4. Pokud obě podmínky jsou splněny, je dokument poslán pomocí dvou metod zobrazených na obrázku číslo 20. Tyto metody se nacházejí ve třídě *DocumentService* v balíčku *service*. Nejdříve musí být dokument rozšifrován a poté je zavolána metoda *streamReport*. Proces šifrování a rozšifrováno je popsán v kapitole 3.2.10.

```
public void generateReport(String resourcePath, HttpServletResponse response) throws
    byte[] fileContent =  decryptFile(resourcePath);
    streamReport(response, fileContent, resourcePath);
}

public void streamReport(HttpServletResponse response, byte[] data, String name)
        throws IOException {

    response.setContentType("application/pdf");
    response.setHeader( s: "Content-disposition",  s1: "attachment; filename=" + name);
    response.setContentLength(data.length);

    response.getOutputStream().write(data);
    response.getOutputStream().flush();
}
```

Obrázek 20: Poslání PDF dokumentu

Žádost o PDF a zpracování odpovědi je zobrazeno na obrázku 21. Musí být nastaven *xhr.responseType* na *arraybuffer*. Pokud je žádost úspěšná, je vytvořen objekt typu *Blob*, kterému je nastaven type na *application/pdf*.

```
const getPdfById = id => {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', proxy + "documents/" + parseInt(id), true);
    xhr.responseType = 'arraybuffer';
    xhr.setRequestHeader("Authorization", 'bearer ' + sessionStorage.access_token);
    xhr.onload = function(e) {
        if (this.status == 200) {
            var blob=new Blob([this.response], {type:"application/pdf"});
            var link=document.getElementById("pdf-link");
            link.href=window.URL.createObjectURL(blob);
```

Obrázek 21: Zpracování PDF souboru

Na obrázku 18 lze vidět, že uživatel má možnost vyjádřit souhlas či nesouhlas s dokumentem. Musí být odeslána žádost na end-point */documents* metodou PUT.

```java
@PutMapping(value = "")
@Secured({("ROLE_USER"),("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public ResponseEntity updateApprovals(OAuth2Authentication auth, @Valid @RequestBody Approval approvals){
    String email = auth.getName();
    long userId = userService.getUserByEmail(email).get().getId();
    int approval = Integer.parseInt(approvals.getApproval());
    long doc_id = approvals.getDoc_id();

    try {
        documentService.updateSharing(userId, doc_id, approval);
        docLogger.info("User " + email + " has voted. Document id" + doc_id);
```

Obrázek 22: End-point PUT /documents

Nejdříve je zkontrolována validita dat pomocí anotace *@Valid* a následně je provedena metoda *updateSharing*, která se nachází ve třídě *DocumentService* v balíčku *service*. Zde může nastat několik scénářů.

Jsou zde brány v potaz dva stavy – *Approved* a *Approving*. Stav *Cancelled* zde není relevantní. Na obrázku 23 lze vidět část kódu ze třídy *DocumentService* v balíčku *service*. Jedná se o část kódu, která řeší proces schvalování dokumentu se stavem *Approving*. Nejdříve je zkontrolováno, zda daný uživatel má právo provést proces schvalování. Následně je zavolaná metoda *updateSharing*, kde je aktualizován atribut *approval* v tabulce *users_documents*. Pokud uživatel nesouhlasí s dokumentem, tak jsou odeslány emaily všem uživatelům, kteří jsou pověřeni schvalováním daného dokumentu se zprávou, že dokument je zablokován. Metoda *setDocumentType* nastavuje stav dokumentu na *Cancelled* na obrázku 23.

```java
if(document.getDocumentState().getId() == 2){
    if(documentRepository.getUsersDocuments(userId, documentId, sharingTypeId: 1) == null){
        return;
    }

    documentRepository.updateSharing(userId, documentId, approval);

    if(approval == 0){
        List<User> users = userService.getUsersForApprovingDocument(documentId);
        Map<String, String> map = emailService.createMessage( type: 2, user, document);

        if(document.getUser().isActive()){
            users.add(document.getUser());
        }

        setDocumentType(documentId, documentTypeId: 3);

        sendEmail(users, map.get("subject"), map.get("message"));
```

Obrázek 23: Schvalování dokumentu 1

Pokud uživatel schvaluje dokument, je nejdříve zkontrolováno, zda dokument je schválen všemi pověřenými uživateli. Pokud je tato podmínka splněna, nastaví se stav

dokumentu na *Approved* a jsou rozeslány emaily všem uživatelům, kteří byli vybráni pro schvalování se zprávou, že daný dokument byl úspěšně schválen.

```
else if(isDocumentApproved(documentId)){
  setDocumentType(documentId,  documentTypeId: 1);

  List<User> users = userService.getUsersForApprovingDocument(documentId);
  if(document.getUser().isActive()){
      users.add(document.getUser());
  }

  Map<String, String> map = emailService.createMessage( type: 5, user, document);
  sendEmail(users, map.get("subject"), map.get("message"));
```

Obrázek 24: Schvalování dokumentu 2

Za předpokladu, že je stav dokumentu *Approved*, dojde ke kontrole, zda uživatel má právo se k tomuto dokumentu vyjadřovat a pokud je potvrzeno jeho oprávnění, jsou aktualizovány databázové položky.

Další funkcionalitou, která je v aplikaci implementována je tvorba dokumentu. Uživatel musí vyplnit všechny potřebné údaje k odeslání dokumentu – titul, popis, do kdy bude dokument povolen ke schvalování, od kdy bude dokument dostupný čtenářům v aplikaci a do kdy, vybrat skupiny pro schvalování, skupiny pro čtení a vybrat PDF soubor. Na obrázku 25 lze vidět, jak vypadá žádost odeslaná na server. Velmi důležité je nastavit *enctype* na *multipart/form-data*. Tato žádost se odesílá na end-point */documents* metodou POST.

```
$.ajax({
  method: 'POST',
  enctype: 'multipart/form-data',
  url: `${proxy}documents`,
  data: data,
  processData: false,
  contentType: false,
  headers: {
    'Authorization': 'bearer ' + sessionStorage.access_token},
```

Obrázek 25: Nahrání dokumentu

O zpracování této žádosti na serveru se stará metoda *getDocumentFromClient* ve třídě *DocumentController* v balíčku *controller*.

```
if(documentService.documentDatesValidity(approvalTime, startOfReading, endOfReading)
        && documentService.checkDocumentValidity(title, desc, approvalGroup, readers)){
    if(documentService.checkFileValidity(multipartFile)){
        try {
            documentService.createDocument(readers, approvalGroup, auth.getName(), title, desc, approvalTime,
                    startOfReading, endOfReading, multipartFile);
```

Obrázek 26: End-point /documents POST

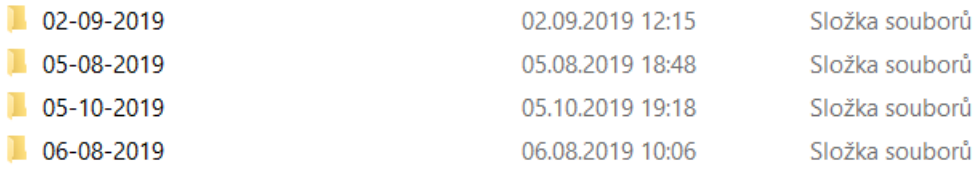
Jsou přijaty pouze dokumenty, které mají velikost maximálně 15 MB. Ostatní jsou zamítnuty. Je nejdříve zkontrolováno, zda přijatá data jsou validní. Zda soubor je validní kontroluje metoda *checkFileValidity*, která je popsána v kapitole 3.2.12.

Metoda *createDocument* je volána za předpokladu, že jsou všechna přijatá data validní. Tato metoda se nachází ve třídě *DocumentService* v balíčku *service*. V této metodě je dokumentu nastaven stav *Approving*, je zašifrován a následně uložen na disk prostřednictvím metody *write*, která je zobrazená na obrázku 27.

```
String date = LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));
createDirectory(date);
String folder = System.getProperty("user.home");
String generatedName = generateName();

String encPath = folder + "\\pdf\\" + date + "/" + generatedName + ".pdf"; // .pdf

encryptFile(file, encPath);

document.setResourcePath(Paths.get(encPath).toString());
document.setName(generatedName);

return generatedName;
```

Obrázek 27: Metoda write

Dokumenty jsou uspořádány ve složkách podle data. Metoda *createDirectory* ve třídě *DocumentService* v balíčku *service* kontroluje, zda taková složka již existuje a pokud ne, je vytvořena.



Obrázek 28: Vytvořené složky aplikací

Dokumenty jsou ukládány do uživatelského domovského adresáře. Pro dokument je generováno unikátní jméno prostřednictvím metody *generateName*. Následně je soubor zašifrován a uložen na disk prostřednictvím metody *encryptFile*. Objektu *document* je nastaveno vygenerovaného jméno a cesta k dokumentu. Metoda *encrypt* na obrázku 29 vytváří objekt typu *SecretKey* a novou instanci třídy *EncrypterDecrypter*, které se nastavuje tajný klíč a typ algoritmu. V tomto případě je používána šifra AES.

```
private void encryptFile(MultipartFile file, String name) throws NoSuchAlgorithmException, No
        SecretKey secretKey = EncrypterDecrypter.getSecretKey(key);
        EncrypterDecrypter encrypterDecrypter
                = new EncrypterDecrypter(secretKey, transformation: "AES/CBC/PKCS5Padding");
        encrypterDecrypter.encrypt(name, file);
```

Obrázek 29: Metoda encryptFile

Po tomto procesu je uložen objekt typu *Document* do databáze. Předtím vytvořený dokument uložený na disku je smazán, pokud operace uložení objektu typu *Document* do databáze není úspěšná.

```
try {
    saveDocument(document);
}catch (PersistenceException e){
    deleteDocumentFromDisk(document.getResourcePath());
    return;
}
```

Obrázek 30: Uložení dokumentu do databáze

V posledním kroku jsou rozeslány emaily uživatelům, kteří byli pověřeni schvalováním dokumentu, se zprávou, že je k dispozici nový dokument ke schválení. Podobu poslaného emailu lze vidět na obrázku 31.

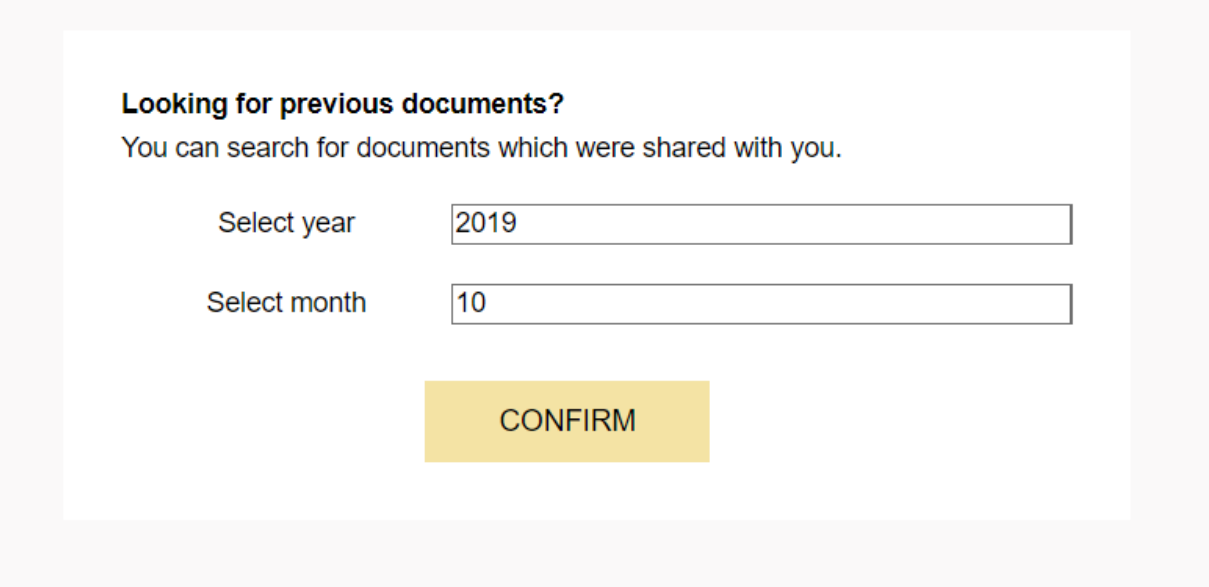
Obrázek 31: Poslaný email aplikací o novém dokumentu

Je umožněno vyhledávání dokumentů různými způsoby. Základním vyhledávacím prostředkem je vyhledávání dokumentů podle titulu. Musí být poslána žádost na end-point */documents/find?name=[hodnota]* metodou GET. Budou vráceny všechny dokumenty, které byly sdíleny s uživatelem a které odpovídají hledanému výrazu. Na obrázku 32 lze vidět JPQL dotaz, který je používán pro vyhledávání dokumentů. Nachází se v metodě *findDocumentByTitle* ve třídě *DocumentRepositoryCustomImpl* v balíčku *repository*.

```
TypedQuery<Document> q2 = entityManager.createQuery( s: "SELECT d FROM Document d " +
        "INNER JOIN d.documentsForUsers ud WHERE (d.documentState.id != 0 AND ud.sharingType = 1 AND" +
                " ud.user.id=:id AND d.title LIKE :value)" +
                " OR(d.documentState.id = 1 AND ud.sharingType.id=2 AND ud.user.id=:id" +
                " AND current_timestamp > d.activeStartTime AND " +
                " d.title LIKE :value) ORDER BY d.title"
        , Document.class);
q2.setParameter( s: "id", userId);
q2.setParameter( s: "value", o: "%"+name+"%"); //
```

Obrázek 32: JPQL dotaz pro vyhledání dokumentu podle titulu

Je umožněno vyhledávání dokumentů podle měsíce a roku. Budou vráceny dokumenty, které byly sdíleny s tímto uživatelem v zadaném období. Žádost musí být poslána na */documents/history?year=2019&month=9* metodou GET.



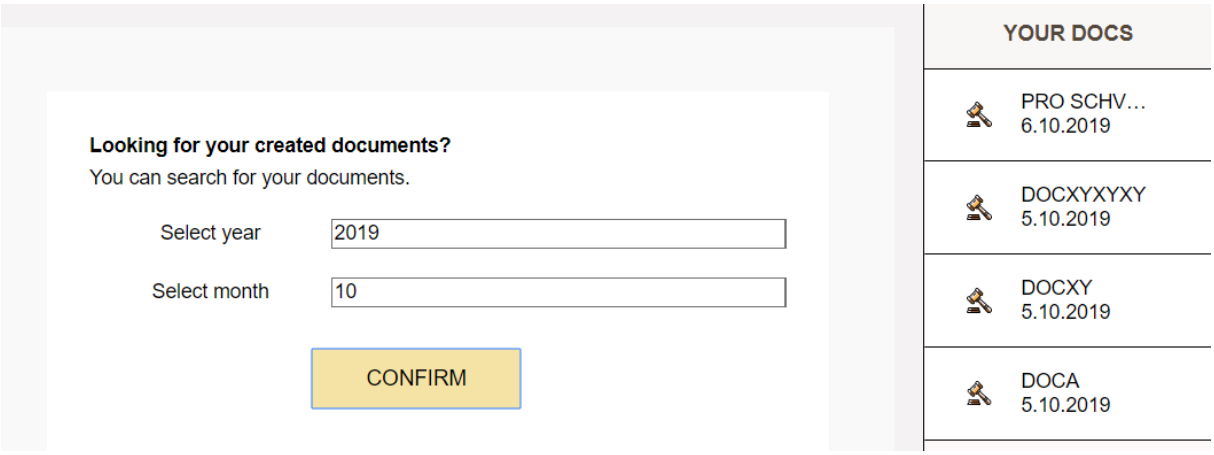
Obrázek 33: Vyhledávání podle měsíce a roku

Na obrázku 34 lze vidět JPQL dotaz, který je používán pro získání sdílených dokumentů v zadaném časovém období. Tento dotaz lze nalézt v metodě *getDocumentsByYearAndMonth* ve třídě *DocumentRepositoryCustomImpl* v balíčku *repository*.

```
TypedQuery<Document> q3 = entityManager.createQuery( s: "SELECT d FROM Document d " +
        "INNER JOIN d.documentsForUsers ud WHERE (d.documentState.id != 0 AND" +
        " ud.sharingType = 1" +
        " AND ud.user.email=:email AND ud.approval != 2 AND" +
        " FUNCTION('MONTH',d.uploadDatetime)=:value3) " +
        "AND FUNCTION('YEAR',d.uploadDatetime)=:value2" +
        " OR(d.documentState.id = 1 AND ud.sharingType.id=2" +
        " AND ud.user.email=:email" +
        " AND current_timestamp > d.activeStartTime AND ud.approval != 2" +
        " AND FUNCTION('MONTH',d.activeStartTime)=:value3) " +
        "AND FUNCTION('YEAR',d.uploadDatetime)=:value2", Document.class);
```

Obrázek 34: JPQL dotaz pro získání sdílených dokumentů v určitém období

Je umožněno vyhledávání svých vytvořených dokumentů. Musí být odeslána žádost na end-point */documents/owner* metodou GET.

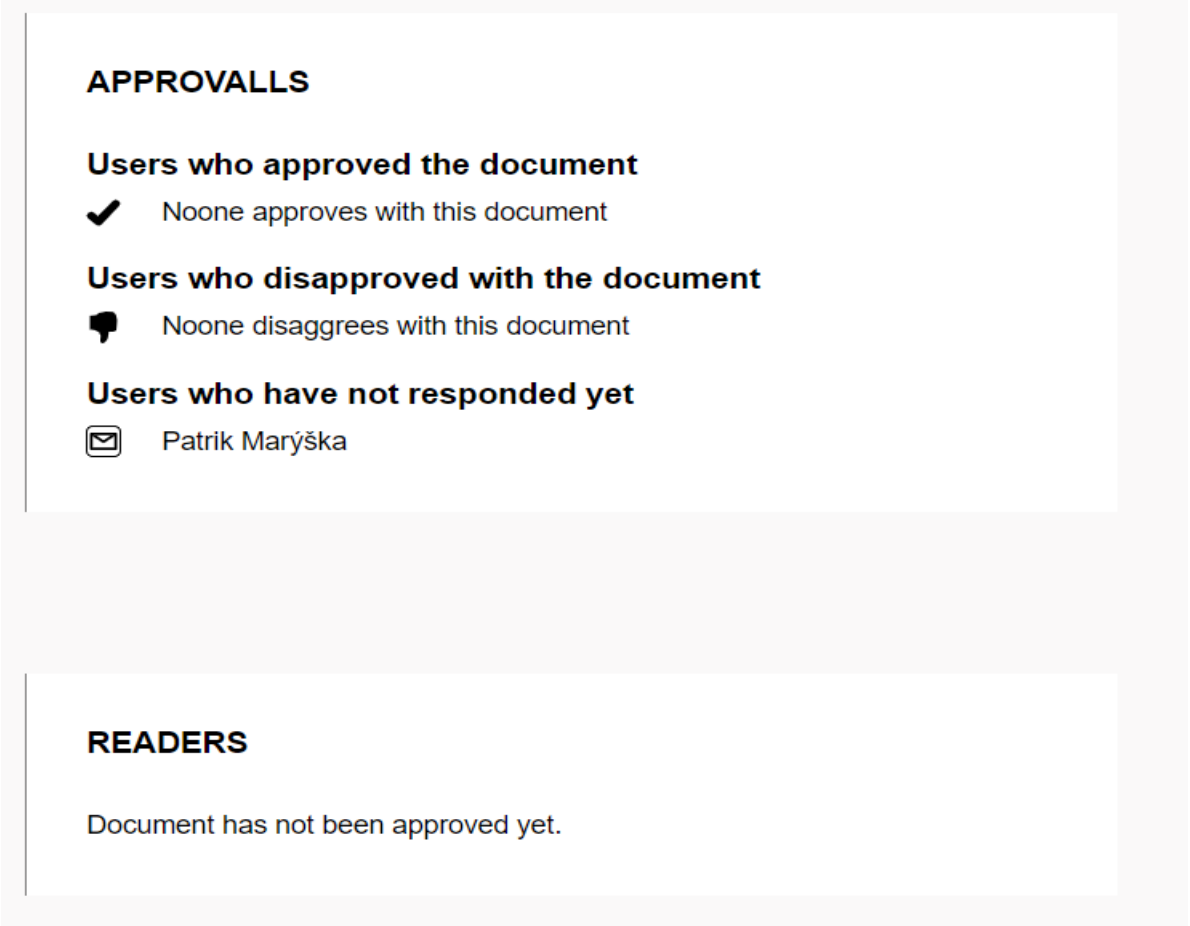


Obrázek 35: Vyhledávání vlastních dokumentů

Na obrázku 35 lze vidět vyhledané dokumenty. Kliknutím na dokument v pravém menu je spuštěna operace načtení informací o dokumentu. Uživatel může vidět výsledky schvalování a kdo dokument přečetl a kdo ne. Na obrázku 36 lze vidět JPQL dotaz pro získání potřebných dat ve třídě *DocumentRepositoryCustomImpl* v balíčku *repository*. Výsledky schvalování lze vidět na obrázku 37.

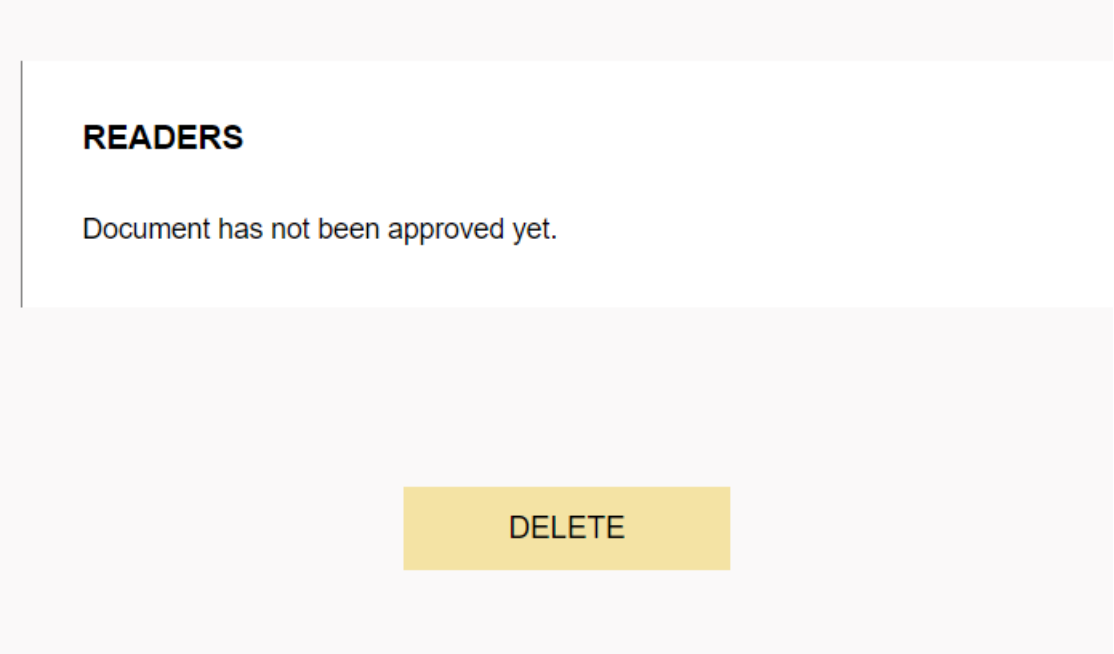
```
TypedQuery<Document> query = entityManager.createQuery( s: "SELECT d FROM Document d JOIN d.user u" +
        " WHERE u.id = :value " +
        "AND FUNCTION('YEAR',d.uploadDatetime)=:year AND FUNCTION('MONTH',d.uploadDatetime)=:month " +
        "ORDER BY LOWER(d.title)", Document.class);

query.setParameter( s: "value", id);
query.setParameter( s: "year", year);
query.setParameter( s: "month", month);
```

Obrázek 36: JPQL dotaz pro získání uživatelových dokumentů



Obrázek 37: Výsledky schvalování

Je implementováno mazání dokumentů. Před smazáním dokumentu musí být nejdříve daný dokument vyhledán. Vyhledávání funguje podobně jako v předchozích případech, kdy dochází k vyhledávání podle měsíce a roku. Operaci smazání dokumentu mají povolenou pouze uživatelé s rolí **ADMIN**. Výsledky tohoto vyhledávání nevrací sdílené dokumenty, ale všechny dokumenty vytvořené v daném časovém období. Musí být nejdříve poslána žádost na end-point */documents/all?year=[rok]&month=[měsíc]* metodou GET. Budou vráceny všechny vytvořené dokumenty v zadaném časovém období. Následně je možné kliknout na nějaký získaný dokument v pravém menu. Tato akce zobrazí informace o daném dokumentu a tlačítko Delete.



Obrázek 38: DELETE tlačítko u dokumentu

Při použití tlačítka Delete je odeslána žádosti *na /documents/[ID dokumentu]* metodou DELETE. Nejdříve je smazán dokumentu z disku a pokud je tato operace úspěšná, jsou vymazána data z databáze. Tato metoda se nachází ve třídě *DocumentService* v balíčku *service*.

```
public boolean deleteDocument(long id){
    boolean x = false;
    Document document = documentRepository.getOne(id);
    if(deleteDocumentFromDisk(document.getResourcePath())){
        documentRepository.deleteAllUsersDocumentsByDocumentId(id);
        x =  documentRepository.deleteDocument(id);
    }
    return x;
}
```

Obrázek 39: Smazání dokumentu

Důležitou třídou je třída *ScheduleService* v balíčku *service*. Tato třída slouží k plánování událostí. Celkem jsou zde vykonávány dvě události.

Každodenní opakování je zadáno pomocí *cron* výrazů v anotaci *@Scheduled(cron =)*. Na obrázku číslo 40 lze vidět první událost, která je nastavena na každodenní opakování ve 23:00 hodin. Z databáze jsou získány všechny dokumenty pomocí metody zobrazené na obrázku 41. Tato metoda vrací dokumenty, u kterých nedošlo k úspěšnému schválení v daném časovém intervalu. Těmto dokumentům je nastaven stav *Cancelled* a jsou rozeslány emaily všem uživatelům, kteří jsou pověřeni schvalováním dokumentu, se zprávou, že daný dokument

nebyl úspěšně schválen.

```
@Scheduled(cron = "0 00 23 * * ?")
public void blockPassedDocuments() {
    List<Document> passedDocuments = documentService.getAllPassedDocuments();

    if (passedDocuments.size() > 0) {
        passedDocuments.forEach(document -> {
            documentService.blockDocument(document);
            List<User> users = userService.getUsersForApprovingDocument(document.getId());

            if(document.getUser().isActive()){
                users.add(document.getUser());
            }

            Map<String, String> map = emailService.createMessage( type: 4, document.getUser(), document);
            documentService.sendEmail(users, map.get("subject"), map.get("message"));
        });
    }
}
```

Obrázek 40: Blokace expirovaných dokumentů

```
@Override
public List<Document> getAllPassedDocuments() {
    TypedQuery<Document> documents = entityManager.createQuery( s: "SELECT d FROM Document d " +
            "WHERE d.approvalEndTime < CURRENT_TIMESTAMP and d.documentState.id=2 ", Document.class);

    return documents.getResultList();
}
```

Obrázek 41: JPQL dotaz pro získání expirovaných dokumentů

Na obrázku 42 lze vidět druhou plánovou operaci. Každých třicet minut je kontrolováno, zda nepřibyly nějaké nové schválené dokumenty pomocí metody zobrazené na obrázku 43. Pokud jsou k dispozici nové dokumenty, jsou rozeslány emaily čtenářům, se zprávou, že mají v aplikaci dostupný nový dokument ke čtení.

```
@Transactional
@Scheduled(cron = "0 0/30 * * * ?")
public void sendEmailAboutReleasingNewDocumentToReaders(){
    List<UsersDocuments> releasingDocuments = documentService.getNewActiveDocuments();
    if(releasingDocuments.size() > 0){
        releasingDocuments.forEach(document -> {
            List<User> users = userService.getUsersForReadingDocument(document.getDocument().getId());

            if(document.getUser().isActive()){
                users.add(document.getUser());
            }

            Map<String, String> map = emailService.createMessage( type: 1, document.getUser(), document.getDocument());
            documentService.sendEmail(users, map.get("subject"), map.get("message"));
            document.setEmailSent(true);
            documentService.updateEmailSent(document);
        });
```

Obrázek 42: Rozeslání emailů o nových dostupných dokumentech

```
@Override
public List<UsersDocuments> getNewActiveDocuments() {
    TypedQuery<UsersDocuments> documentTypedQuery = entityManager.createQuery( s: "SELECT ud FROM Document d" +
            " JOIN d.documentsForUsers ud WHERE d.activeStartTime < current_timestamp " +
            "AND ud.emailSent=0 AND d.documentState.id=1", UsersDocuments.class);

    return documentTypedQuery.getResultList();
}
```

Obrázek 43: JPQL dotaz pro získání nových dokumentů

V balíčku *email* se nachází interface *EmailService*, který definuje základní metody pro napsání a poslání emailu. Je tu také třída *EmailServiceImpl*, která implementuje interface *EmailService*.

```
@Autowired
public JavaMailSender emailSender;

private String address = "https://docmanager.com";


@Async
@Override
public void sendSimpleMessage(String to, String subject, String text) {
    SimpleMailMessage message = new SimpleMailMessage();
    message.setTo(to);
    message.setSubject(subject);
    message.setText(text);
    emailSender.send(message);
}
```

Obrázek 44: Poslání emailu a získání objektu typu JavaMailSender

Bean typu *JavaMailSender* je vytvořen automaticky Springem. Je nutné definovat následující informace na obrázku 45 ke správné funkcionalitě tohoto beanu.

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=bcpraceemail@gmail.com
spring.mail.password=P4ssword-
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Obrázek 45: Konfigurace emailové komunikace pro Gmail SMTP Server

3.3.4 Uživatelé

Při registraci nového uživatele musí být vyplněny potřebné údaje – jméno, příjmení, email, přidělené role a heslo. Žádost musí být poslána na end-point */users* metodou POST. Je zkontrolováno, zda obdržená data jsou validní pomocí anotace *@Valid* a následně je volána metoda *createUser*. Hashování hesla je popsáno v kapitole 3.2.5.

```
public void createUser(@Valid @RequestBody UserBody userBody, OAuth2Authentication auth) {

    List<Role> roles = new ArrayList<>();
    userBody.getRoles().forEach(s -> roles.add(userService.getRoleByName(s)));

    User user = new User(userBody.getFirstName(), userBody.getSurname(), userBody.getEmail(), userBody.getPassword());
    user.setActive(true);
    user.setRoles(roles);
    try {
        userService.createUser(user);
        logger.info("USER " + userBody.getEmail() + " has been created by " + auth.getName());
    } catch (Exception e) {
        logger.error("USER " + userBody.getEmail() + " could not have been created by" + auth.getName());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "User email is already in use.");
    }
}
```

Obrázek 46: Registrace klienta

Aktualizace informací o uživateli vyžaduje poslání žádosti na end-point */users* metodou PUT. Aby mohlo dojít k aktualizaci, musí objekt *user* projít úspěšně validací, která je provedena pomocí anotace *@Valid*.

```
public void updateUser(@Valid @RequestBody UserBody user, OAuth2Authentication auth) {

    List<Role> roles = new ArrayList<>();
    user.getRoles().forEach(s -> roles.add(userService.getRoleByName(s)));

    User u = userService.findUserById(user.getId());
    u.setFirstName(user.getFirstName());
    u.setSurname(user.getSurname());
    u.setRoles(roles);
    u.setEmail(user.getEmail());

    try {
        userService.updateUser(u);
        logger.info("USER " + user.getEmail() + " has been updated by " + auth.getName());
```

Obrázek 47: Aktualizace uživatele

Před deaktivací/aktivací uživatele je nejdříve nutné vyhledat uživatele dle jména. Na obrázku 48 lze vidět, že je nejdříve zkontrolováno, zda zadaný řetězec může být jméno. Obrázek 49 ukazuje používány JPQL dotaz pro získání uživatelů dle příjmení. Obě metody se nachází ve třídě *UserRepositoryCustomImpl* v balíčku *repository*.

```java
@GetMapping("/finds")
@Secured({("ROLE_DOCUMENT_CREATOR"), "ROLE_ADMIN"})
public List<User> findByLike(@RequestParam("email") String surname, OAuth2Authentication oauth) {

    if (surname.matches( regex: "^[\\p{L}\\s.'\\-,]+$")) {
        return userService.getUsersBySurname(surname);
    } else {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Surname is not in the correct format.");
    }
}
```

Obrázek 48: Hledání uživatele podle jména

```java
@Override
public List<User> getUsersBySurname(String surname) {
    TypedQuery<User> userQuery = entityManager.createQuery( s: "SELECT u FROM User u" +
            " WHERE u.surname LIKE :surname ORDER BY u.surname", User.class);
    String like = "%" + surname + "%";
    userQuery.setParameter( s: "surname", like);

    return userQuery.getResultList();
}
```

Obrázek 49: JPQL dotaz pro získání uživatelů podle příjmení

Po obdržení dat jsou vypsány výsledky. Je možné kliknout na chtěného uživatele. Touto akcí dojde k vypsání informací o uživateli a nachází se zde dvě tlačítka – Update a Deactivate/Activate. Tlačítko Deactivate se objeví v případě, že zvolený uživatel má nastavený atribut *active* na 1. Pokud je atribut *active* nastaven na 0, objeví se tlačítko Activate. Při deaktivaci je uživatel vymazán ze všech skupin a jsou vymazána i všechna jeho spojení s dokumenty. Nebude již tedy dostávat žádné informace prostřednictvím emailů ohledně dokumentů, u kterých byl vybrán ke schvalování nebo ke čtení.

Obrázek 50: Update a Deactivate tlačítka

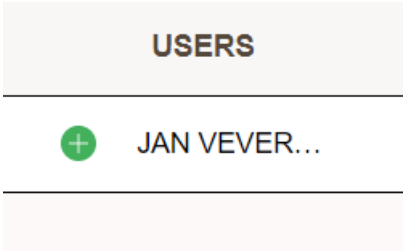
```
public void deactivateUser(long id) {
    User user = findUserById(id);
    if(user.isActive()){
        unitService.removeUserFromGroups(id);
        removeUserFromUsersDocuments(id);
    }
    user.setActive(!user.isActive());

    updateUser(user);
}
```

Obrázek 51: Deaktivace uživatele

3.3.5 Skupiny

Je umožněno vytváření vlastních skupin pro sdílení dokumentů. Celkem může mít uživatel vytvořených třicet skupin.

Tvorba skupiny zahrnuje vyhledání uživatelů dle příjmení. Budou vráceny všichni uživatelé, kteří odpovídají tomuto zadání a mají aktivní účet.



Obrázek 52: Výsledek vyhledávání

Prostřednictvím ikonky plus je daný uživatel přidán do skupiny. Skupinu je nutné pojmenovat a mít v ní alespoň jednoho uživatele. Nyní mohou být data poslána na server na end-point */groups* metodou POST.

```
@PostMapping("")
@Secured({("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public void createGroup(@Valid @RequestBody UnitBody unit, OAuth2Authentication auth){
    try {
        unitService.saveUnit(unit, auth.getName());
        logger.info("Unit " + unit.getName() + " has been created by " + auth.getName());
```

Obrázek 53: Uložení skupiny

Je umožněna aktualizace skupin. Musí být vyhledány všechny uživatelovy skupiny prostřednictvím dotazu */groups* metodou GET. Následně je možné zvolit nějakou získanou skupinu a provést chtěnou operaci. Aktualizace probíhá tak, že je odeslána žádost na end-point */groups* metodou PUT. Na serveru jsou data zkontrolována a případně je vykonána

aktualizace. Jak lze vidět na obrázku 55, je zkontrolováno, zda poslaná skupina obsahuje nějaké členy. Pokud ne, skupina je smazána. Za předpokladu, že skupina obsahuje nějaké členy, je provedena kontrola, zda daný uživatel má dostatečné oprávnění měnit tuto skupinu. Metoda na obrázku 54 se nachází v třídě *UnitService* v balíčku *service*.

```
public void updateUnit(Group group, String email) throws AuthorizationServiceException {
    User user = userService.getUserByEmail(email).get();
    if(group.getIds().size() == 0){
        deleteUnit(group.getId(), email);
        return;
    }

    if(user.getGroups().stream().anyMatch(unit -> unit.getId() == group.getId())){
        Unit unit = new Unit();
        unit.setId(group.getId());
        unit.setName(group.getName());
        unit.setUser(user);
        unit.setUsers(userService.getUsersFromIds(group.getIds()));
        unitRepository.updateUnit(unit);
    } else {
        throw new AuthorizationServiceException("You cannot update this group.");
    }
}
```

Obrázek 54: Aktualizace skupiny

Smazání skupiny je provedeno tak, že musí být poslána žádost na */groups* metodou DELETE. Je zkontrolováno, zda uživatel má právo mazat tuto skupinu předtím, než dojde k samotné operaci smazání dokumentu. Tato metoda se nachází v balíčku *UnitService* v balíčku *service*.

```
public void deleteUnit(long id, String email) throws AuthorizationServiceException {
    Unit unit = unitRepository.getOne(id);
    User user = userService.getUserByEmail(email).get();

    if(user.getGroups().contains(unit)){
        unitRepository.delete(unit);
    } else {
        throw new AuthorizationServiceException("You do not have permission to delete this group.");
    }
}
```

Obrázek 55: Smazání skupiny

3.3.6 Další funkcionalita

Při odhlášení je poslána zpráva na server na end-point */oauth/revoke-token* metodou POST, se zprávou, že došlo k odhlášení uživatele. Na obrázku 56 lze vidět způsob, jakým je řešena operace odhlášení uživatele. Tato metoda se nachází ve třídě *OauthController* v balíčku *controller*.

```
@PostMapping(value = "/oauth/revoke-token")
@ResponseStatus(HttpStatus.OK)
public void logout(HttpServletRequest request, OAuth2Authentication auth) {
    String authHeader = request.getHeader( s: "Authorization");
    if (authHeader != null) {
        String tokenValue = authHeader.replace( target: "Bearer",  replacement: "").trim();
        OAuth2AccessToken accessToken = tokenStore.readAccessToken(tokenValue);
        tokenStore.removeAccessToken(accessToken);
        logger.info("User " + auth.getName() + " has logged out.");
    }
}
```

Obrázek 56: Odhlášení

3.2 Bezpečnostní mechanismy na serveru

3.2.1 Oauth2

Autorizační server

Na obrázku číslo 57 lze vidět část kódu pro konfiguraci autorizačního serveru. Metoda se nachází ve třídě *AuthorizationServer* v balíčku *config*.

```
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients
            .inMemory() InMemoryClientDetailsServiceBuilder
            .withClient(Const.CLIENT_ID) ClientDetailsServiceBuilder<InMemoryClientDetailsServiceB
            .secret(Const.CLIENT_SECRET) ClientDetailsServiceBuilder<InMemoryClientDetailsServiceB
            .authorizedGrantTypes(GRANT_TYPE_PASSWORD, REFRESH_TOKEN) ClientDetailsService
            .scopes(SCOPE_WRITE, SCOPE_READ) ClientDetailsServiceBuilder<InMemoryClientDetailsSer
            .accessTokenValiditySeconds(900) ClientDetailsServiceBuilder<InMemoryClientDetailsSer
            .refreshTokenValiditySeconds(1800);
}
```

Obrázek 57: Autorizační server

Je nutné označit tuto třídu pomocí dvou anotací.

- *@Configuration* – tato anotace říká, že tato třída bude sloužit jako konfigurační.
- *@EnableAuthorizationServer* – povolení autorizačního serveru.

Tato třída musí dědit z třídy *AuthorizationServerConfigurerAdapter*. Metoda *configure(ClientDetailsServiceConfigurer clients)* specifikuje přihlašovací údaje klientské aplikace a nabízené služby. Je patrné, že tato konfigurace bude ukládat služby v paměti. Typy grantu reprezentují práva klientské aplikace na informace uživatele. V tomto případě klientská aplikace má práva číst a psát uživatelovo heslo a obnovovací token. Je definováno, že

přístupový token má životnost 900 sekund a obnovovací token má životnost 1800 sekund.

Musí být definován *AuthentizationManager* bean. Spring umožnuje tento bean používat automaticky pomocí *@Autowired* anotace.

TokenStore je objekt, ve kterém budou uloženy identifikátory, které autorizační server vytvoří. V tomto případě je použit *InMemoryTokenStore*, který ukládá identifikátory do paměti.

```
@Bean
protected InMemoryTokenStore tokenStore() { return new InMemoryTokenStore(); }
```

Obrázek 58: TokenStore u autorizačního serveru

Na obrázku číslo 59 je vytvářen nový bean, kde je nastaven objektu typu *JwtAccessTokenConverter* pár klíčů. K tomuto účelu je potřeba načíst Java KeyStore, který byl vygenerován pomocí nástroje *KeyToolExplorer*. Tím je zajištěno podepsání tokenů na straně autorizačního serveru.

```
@Bean
protected JwtAccessTokenConverter jwtTokenEnhancer() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    KeyStoreKeyFactory keyStoreKeyFactory =
            new KeyStoreKeyFactory(new ClassPathResource("test.jks"), "password".toCharArray());
    converter.setKeyPair(keyStoreKeyFactory.getKeyPair( alias: "test"));

    return converter;
}
```

Obrázek 59: Asymetrické podepsání JWT na straně autorizačního serveru

Metoda *configure(AuthroizationServerEndpointConfigurer endpoints)* slouží k nastavení funkcionality end-pointů autorizačního serveru. Těmto end-pointům jsou nastaveny objekty *tokenStore*, *jwtTokenEnhancer* a *authenticationManager*. Povolujeme pouze metodu POST pro všechny end-pointy autorizačního serveru.

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints
            .tokenStore(tokenStore()).reuseRefreshTokens(false)
            .tokenEnhancer(jwtTokenEnhancer())
            .authenticationManager(authenticationManager)
            .allowedTokenEndpointRequestMethods(HttpMethod.POST);
}
```

Obrázek 60: Konfigurace end-pointů autorizačního serveru

Resource server

Třída, která zastupuje resource server se jmenuje *ResourceServer* a nachází se v balíčku *config*. Musí být označena anotací *@Configuration* a *@EnableResourceServer* a je nutné, aby dědila z třídy *ResourceServerConfigurerAdapter*. Význam druhé anotace je, že povolujeme resource server v dané třídě.

Na obrázku 61 lze vidět proces ověřování podpisu tokenů pomocí veřejného klíče. Pokud nedojde ke správnému ověření podpisu, aplikace vrátí chybu uživateli.

```
@Bean
protected JwtAccessTokenConverter jwtTokenEnhancerr() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    Resource resource = new ClassPathResource("public.txt");
    String publicKey = null;
    try {
        publicKey = IOUtils.toString(resource.getInputStream());
    } catch (final IOException e) {
        throw new RuntimeException(e);
    }
    converter.setVerifierKey(publicKey);

    return converter;
}
```

Obrázek 61: Ověření podpisu

Je nutné přepsat metodu *configure* způsobem zobrazeným na obrázku 62. Je zde nastaven objekt *tokenStoree*, který je typu *JwtTokenStore*. Třída *JwtTokenStore* pouze čte data z tokenů, nic neukládá a používá objekt typu *JwtAccessTokenConverter* ke čtení a ověřování tokenů.

```
@Override
public void configure(ResourceServerSecurityConfigurer config) { config.tokenStore(tokenStoree()); }


@Bean
public TokenStore tokenStoree() { return new JwtTokenStore(jwtTokenEnhancerr()); }
```

Obrázek 62: JwtTokenStore a metoda configure

K povolení HTTP bezpečnosti je nutné přepsat metodu na obrázku 63.

Obrázek 63: HTTP bezpečnost

Tímto způsobem říkáme, že povolujeme všechny žádosti na end-point */oauth/token* a */oauth/valid*. Požadujeme, aby uživatel, který chce přistoupit k end-pointům */documents/***, */users/***, */groups/*** a */oauth/revoke-token* byl již autentizován. To znamená, že k přístupu k těmto end-pointům je vyžadován přístupový token. Také se nastavuje, že aplikace nepoužívá cookies a přidává se hlavička CSP.

Další třída, která souvisí s nastavením bezpečnosti resource serveru je *SecurityConfig* ve stejném balíčku. Musí být označena anotací *@Configuration*, *@EnableWebSecurity* a *@EnableGlobalMethodSecurity* a musí dědit z třídy *WebSecurityConfigurerAdapter*. Přidáním anotace *@EnableWebSecurity* říkáme, že tato třída je bezpečnostní konfigurace Springu. Anotací *@EnableGlobalSecurity(securedEnabled=true)* říkáme, že chceme povolit používání anotace *@Secured*. Tato anotace umožňuje kontrolovat, zda uživatel je oprávněn přistoupit k chráněnému zdroji.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;
```

Obrázek 64: Anotace třídy SecurityConfig

Metoda *global(AuthentizationManagerBuilder)* obsahuje informace, jakým způsobem jsou získávány informace o uživateli.

```
@Autowired
public void globalUserDetails(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
}
```

Obrázek 65: Metoda globalUserDetails

UserDetailsService interface je používáno k získání informací o uživateli. Má jednu metodu *loadUserByUsername*, která může být přepsána za účelem přizpůsobení procesu vyhledávání uživatele. Tato vlastní implementace může být nalezena ve třídě *UserDetailsService* v balíčku *service*.

```
return new User(appUser.getEmail(),
        appUser.getPassword(),
        getAuthority(appUser.getId()));
```

Obrázek 66: Vracení objektu typu User

3.2.2 HTTPS

REST server byl nakonfigurován tak, aby komunikoval pouze pomocí HTTPS, nevyužíval slabé šifry a ke komunikaci používal pouze TLS1.2.

V prvním kroku bylo nutné získat certifikát podepsaný sám sebou. Postup získání certifikátu je popsán v kapitole 2.2.1. Tento certifikát lze najít ve složce *resources/keystore/dm_fin.p12*.

Spring Boot používá ke konfiguraci soubor *application.properties*, který se nachází ve složce *resources*. V tomto souboru je použita konfigurace na obrázku číslo 67.

```
server.ssl.key-store-type=PKCS12
server.ssl.key-store=classpath:keystore/dm_fin.p12
server.ssl.key-store-password=password
server.ssl.key-alias=dm_fin
server.ssl.protocol=TLS
server.ssl.enabled-protocols=TLSv1.2
server.ssl.ciphers=DTLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS
server.port=8443
```

Obrázek 67: Nastavení HTTPS

3.2.3 Autentizace

Při přihlášení uživatele je poslána žádost zobrazená na obrázku 68.

```
POST /oauth/token HTTP/1.1
Host: 192.168.0.103:8443
Authorization: Basic bXktY2xpZW50Og==
Content-Type: application/x-www-form-urlencoded

username=test%40seznam.cz&grant_type=password&password=515EA6ir%23
```

Obrázek 68: Autentizační žádost

Je nutné použít metodu POST na end-point */oauth/token*. Musí být přítomná hlavička *Authorization: Basic [přihlašovací údaje klienta zakódované pomocí Base64]* a *Content-Type* nastaven na *application/x-www-form-urlencoded*. Pokud hlavička *Authorization: Basic [hodnota]* obsahuje správné přihlašovací údaje klienta a přihlašovací údaje uživatele jsou v pořádku, aplikace vrátí následující údaje ukázané na obrázku 69.


```
{
    "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NzEwNjY1NDYsInVzZXJfbmFtZSI6InBhdHJpa21hc
nlza2FAZ21haWwuY29tIiwiYXV0aG9yaXRpZXMiOlsiUk9MRV9ET0NVTUVOVF9DUkVBVE9SIiwiUk9MRV9VU0VSIl0s
Imp0aSI6IjA1NGVmZTU0LWVmNDUtNGEzNi05ZjkwLTJjODkzOGZjMWE3OSIsImNsaWVudF9pZCI6Im15LWNsaWVudCI
sInNjb3BlIjpbIndyaXRlIiwicmVhZCJdfQ.cpciNX1vaKSLWHecGfjznAAWRiH1BgT22ip13IENCeHytkf3dOpEGkY
5-Sh-tKi4cC-
9ROVJ4rB6NYX2R0bxeN46CxDRcKiJpnuBOVQWkOrTQ0W_5k96EG1AmYrZhK_mhSRUqynWHupoQluqHhjEppupBeg-
n2RUXCzfaV-coO8jQBmISDtgEgsdu_cXPzxRE4FCbsrabXSHtMLfUsFi1uBEc05S_vW-
m6kDVzP6jM9BQgVfQt0IDZXbHtUrfDnb4DN0eo64W978JdnzHTiwPLt0KFR1eHQsCYuyUrCA66vM7tquFdN-
Cq2GdTwkozaHetrM77MND_EC7RfX0P_h3A",
    "token_type": "bearer",
    "refresh_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJwYXRyaWttYXJ5c2thQGdtYWlsLmNvbSIsI
nNjb3BlIjpbIndyaXRlIiwicmVhZCJdLCJhdGkiOiIwNTRlZmU1NC1lZjQ1LTRhMzYtOWY5MC0yYzg5MzhmYzFhNzki
LCJleHAiOjE1NzEwNzQyODYsImF1dGhvcml0aWVzIjpbIlJPTEVfRE9DVU1FTlRfQ1JFQVRPUiIsIlJPTEVfVVNFUiJ
dLCJqdGkiOiJiN2Y5YmFiNS05ZTEwLTQ4NmItYjNlOS1kYTYxMzc5N2MyNGEiLCJjbGllbnRfaWQiOiJteS1jbGllbn
QifQ.C1QOG2trWFF1rQuYJRPVio0_SfQv_YIxDR14wJBgd7FNL80baIveBYnzB0ONuMqq-
cjIzx_LSAQLvV__F_vv1p7Ic9FOMssPiCCQd12dfpwwcTL8ZIdqIMeiyZknQE3pUijx6GC0knWxDxMSGukUfF8O4kg2
h8Gx1cYA4lA2AZKcvk1_u-6_TaYgBOSUNrBPQSakUQnkh9e4pHCo5Xn0g_8EfUWV1TtpWYBRf6-
fGWZtOVmQQ7WIvGyKt6QY4jHLWRHhvErGePPNLCjv23zEMPEX9lXWoEkqgxIx2hl0bNIgRjJwC9x08c6uVwxsOeAYcv
2uwAyElA4PpV_zuOONTw",
    "expires_in": 899,
    "scope": "write read",
    "jti": "054efe54-ef45-4a36-9f90-2c8938fc1a79"
}
```


Obrázek 69: Obdržená data po úspěšné autentizaci

Pomocí přístupového tokenu je možné přistupovat k chráněným zdrojům za předpokladu, že uživatel má dostatečné oprávnění.

Při úspěšné autentizaci uživatel obdrží i obnovovací token. Tento token slouží k žádosti o nový přístupový token, za předpokladu, že vypršela životnost přístupového tokenu.

Aplikace používá jeden důležitý end-point pro autentizaci, který slouží k obraně proti obcházení autentizačního schématu. Klientská aplikace po úspěšném loginu posílá token ke kontrole serveru. Metoda, která se stará o tento end-point je zobrazená na obrázku 70.

```
@GetMapping(value = "oauth/valid")
@ResponseStatus(HttpStatus.OK)
public String isTokenValid(HttpServletRequest request) {
    String authHeader = request.getHeader( s: "Authorization");

    if (authHeader != null) {
        String tokenValue = authHeader.replace( target: "Bearer", replacement: "").trim();
        OAuth2AccessToken accessToken = tokenStore.readAccessToken(tokenValue);

        if(accessToken != null){
            return "OK";
        }
    }
    return "NOT OK";
}
```

Obrázek 70: Kontrola tokenu

3.2.4 Autorizace

Je ověřováno několika způsoby, zda uživatel má právo přistoupit k chráněnému zdroji. Jsou využívány celkem tři role, které byly vysvětleny v kapitole 2.1. Pomocí anotace *@Secured* je stanoveno, které role mohou přistoupit k danému chráněnému zdroji.

```
@GetMapping(value = "/owner/{id}")
@Secured({("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public List<UsersApproval> getUsersForDocument(@PathVariable("id") long id, OAuth2Authentication auth){
```

Obrázek 71: Chráněný zdroj anotací @Secured

Aby toto zabezpečení bylo funkční, je nutné přidat nad třídu *SecurityConfig* v balíčku *config* anotaci *@EnableGlobalSecurity(securedEnable=true)*, jak je popsáno v kapitole 3.2.1.

Role uživatele se nachází v přístupovém tokenu. Dle rolí, které se nacházejí v JWT tokenu je rozhodnuto, zda uživatel má právo přistoupit k danému chráněnému zdroji.

Dalším způsobem autorizace je kontrola, zda uživatel má přístup k nějakému objektu. Příkladem může být metoda *hasUserAccessToDocument(long documentId, long userId)* ve třídě *DocumentRepository* v balíčku *repository*. Pokud je poslána žádost na end-point */documents/[id]* metodou GET uživatelem, který nemá mít k danému dokumentu přístup, server vrátí chybu uživateli. Podobných metod, které kontrolují, zda uživatel má přístup k objektu je v aplikaci více.

3.2.5 Heslo

Musí být vytvořen bean typu *PasswordEncoder*, který je zobrazen na obrázku číslo 72. Tento bean obsahuje dvě základní metody – *encode* a *matches*.

```
@Bean
public PasswordEncoder encoder() { return new BCryptPasswordEncoder(); }
```

Obrázek 72: Bean encoder

Registrace klienta je provedena způsobem zobrazeným na obrázku 73.

```
public void createUser(User user) {
    user.setPassword(encoder().encode(user.getPassword()));
    userRepository.createUser(user);
}
```

Obrázek 73: Vytvoření uživatele

Poskytnuté heslo musí splňovat požadavky popsané v kapitole 2.2.2. Tyto požadavky jsou zobrazeny na obrázku číslo 74.

```
@Override
public boolean isValid(String password, ConstraintValidatorContext context) {
    PasswordValidator validator = new PasswordValidator(Arrays.asList(
            new LengthRule(8, 40),
            new UppercaseCharacterRule( num: 1),
            new DigitCharacterRule( num: 1),
            new SpecialCharacterRule( num: 1),
            new LowercaseCharacterRule( num: 1),
            new WhitespaceRule()));

    RuleResult result = validator.validate(new PasswordData(password));
    if (result.isValid()) {
        return true;
    }
    context.disableDefaultConstraintViolation();
    context.buildConstraintViolationWithTemplate(
            Joiner.on(",").join(validator.getMessages(result)))
            .addConstraintViolation();
    return false;
}
```

Obrázek 74: Požadavky na heslo

Autentizace probíhá tak, že je porovnáno heslo poskytnuté uživatelem v prostém textu s hash hodnotou uloženou v databázi. Tento proces lze vidět na obrázku 75.

```
if(BCrypt.checkpw(request.getParameter( s: "password"), appUser.getPassword())){
```

Obrázek 75: Autentizace uživatele

Obrázek číslo 76 ukazuje, jak je řešena změna hesla. Je zkontrolováno, zda poskytnuté aktuální heslo souhlasí s hash hodnotou hesla v databázi a následně je porovnána hash hodnota aktuálního hesla s hash hodnotou nového hesla. Pokud nejsou stejné, dojde k operaci změny hesla.

```
public void changePassword(String name, String oldPassword, String newPassword) throws BadCredentialsException {
    User user = userRepository.getUserByEmail(name).get();
  if(encoder().matches(oldPassword, user.getPassword())){
      if(!encoder().matches(newPassword, user.getPassword())){
          userRepository.changePassword(name, encoder().encode((newPassword)));
      } else {
          throw new InternalError("Provided password is the same or too familiar with your current one. Choose different one please");
      }
   } else {
      throw new BadCredentialsException("Provided password is not correct.");
   }
```

Obrázek 76: Změna hesla

3.2.6 Chyby

Jsou vráceny vlastní reakce na chyby. K tomuto účelu je použita třída *EntityExceptionHandler* v balíčku *exception*. Tato třída reaguje na vybrané chyby globálně. K tomuto účelu je třeba přidat anotaci *@RestControllerAdvice* nad třídu a anotaci *@ExceptionHandler* nad metodu, která se stará o vracení vlastní zprávy v reakci na nějakou vyhozenou výjimku. Na obrázku číslo 77 je příklad jedné metody, která vrací vlastní zprávu na chybu typu *MethodArgumentMismatchException*.

```
@ExceptionHandler(MethodArgumentTypeMismatchException.class)
public ResponseEntity<?> handleMissingRequestBody(MethodArgumentTypeMismatchException ex, WebRequest request) {
    ErrorMessage errorObj = new ErrorMessage(new Date(),  message: "Wrong parameters occurred. Try it again please.",
            request.getDescription( b: false));
    return new ResponseEntity<>(errorObj, new HttpHeaders(), HttpStatus.BAD_REQUEST);
}
```

Obrázek 77: Globální reakce na výjimku

Je využíván ještě jeden způsob vracení výjimek uživateli zobrazený na obrázku 78.

```
if (surname.matches( regex: "^[\\p{L}\\s.'\\-,]+$")) {
    return userService.findByLikeActive(surname, oauth.getName());
} else {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Surname is not in the correct format.");
}
```

Obrázek 78: ResponseStatusException

3.2.7 CORS

Webový server IIS je nastaven, aby běžel na adrese docmanager.com na portech 80 a 443. REST server je nastaven, aby fungoval na stejné adrese, ale na portu 8443. Je tedy nutné nastavit CORS, aby mohla klientská strana komunikovat s REST serverem pomocí

technologie AJAX.

CORS je nastaveno ve třídě *CorsConfig* v balíčku *config*. Tato třída je rozšířena o interface *Filter*, který obsahuje dvě metody – *init* a *doFilter*. V metodě *doFilter* dochází k nastavení CORS politiky, jak je ukázáno na obrázku číslo 79.

```
response.setHeader( s: "Access-Control-Allow-Origin", s1: "https://docmanager.com");
response.setHeader( s: "Vary", s1: "Origin");
response.setHeader( s: "Access-Control-Allow-Methods", s1: "POST, GET , DELETE, PUT");
response.setHeader( s: "Access-Control-Max-Age", s1: "3600");
response.setHeader( s: "Access-Control-Allow-Headers", s1: "Content-type, Authorization");

if("OPTIONS".equalsIgnoreCase(request.getMethod())){
    response.setStatus(HttpServletResponse.SC_OK);
} else {
    chain.doFilter(req, res);
}
```

Obrázek 79: Nastavení CORS

3.2.8 Managment end-pointy

Spring Boot umožňuje používání actuator end-pointy. To jsou end-pointy, které umožnují monitorovat aplikaci. Spring Boot nabízí několik zabudovaných end-pointů a umožňuje přidání vlastních end-pointů. End-point */actuator/health* poskytuje základní informace o aplikaci [18]. Tyto end-pointy lze využívat při vývoji, ale při nasazení aplikace by měly být vypnuty, protože mohou prozradit interní informace o aplikaci.

Vypnutí actuator end-pointů lze dosáhnout přidáním konfiguračního řádku zobrazeného na obrázku 80 do souboru *application.properties* ve složce *resources*.

```
management.endpoints.enabled-by-default=false
```

Obrázek 80: Vypnutí actuator end-pointů

3.2.9 Logování

V souboru *logback.xml* se nachází konfigurace logování v aplikaci. Podrobnější popis logování je popsán v kapitole 2.2.10.

Na obrázku číslo 81 lze vidět vytvoření objektu typu *Logger*. Tímto způsobem získáváme *loggera* definovaného v souboru *logback.xml* se jménem *login*.

```
private static Logger logger = LoggerFactory.getLogger( name: "login");
```

Obrázek 81: Vytvoření instance třídy Logger

Zapisování do souboru je ukázáno na obrázku 82. Uživatel se úspěšně odhlásil a tato informace se zaznamenává do logu.

```
logger.info("User " + auth.getName() + " has logged out.");
```

Obrázek 82: Zápis informace do logu

Definování globálního logovacího souboru je nastaveno přidáním řádku zobrazeného na obrázku 83 do souboru *application.properties* ve složce *resources*.

```
logging.file = logfile.log
```

Obrázek 83: Globální logovací soubor

3.2.10 Šifrování na disku

Je umožněno šifrování dokumentů na disku pomocí symetrického klíče. K tomuto účelu slouží třída *EncrypterDecrypter* v balíčku *utils*. Tato třída umožňuje zašifrování dokumentu pomocí algoritmu AES a uložení zašifrovaného dokumentu na disk. Velikost klíče je 256 bitů. Tuto metodu lze vidět na obrázku číslo 84.

```
public void encrypt(String fileName, MultipartFile file) throws InvalidKeyException, IOException {
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] iv = cipher.getIV();

    try (FileOutputStream fileOut = new FileOutputStream(fileName);
         CipherOutputStream cipherOut = new CipherOutputStream(fileOut, cipher)) {
        fileOut.write(iv);

        cipherOut.write(file.getBytes()); // file.getBytes() původně
    } catch (IOException e){
        throw new IOException("Error while processing file");
    }
}
```

Obrázek 84: Zašifrování a uložení dokumentu

Tato třída umožňuje také dešifrování pomocí metody, kterou lze vidět na obrázku 85

```
public byte[] decrypt(String fileName) throws InvalidKeyException, IOException, InvalidAlgorithmParameterException {
    Path path = new File(fileName).toPath();
    try (FileInputStream fileIn = new FileInputStream(path.toFile())) {
        byte[] fileIv = new byte[16];
        fileIn.read(fileIv);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, new IvParameterSpec(fileIv));

        try (
                CipherInputStream cipherIn = new CipherInputStream(fileIn, cipher);
        ) {
            return cipherIn.readAllBytes();
        }
    } catch (InvalidKeyException e) {
        throw new InvalidKeyException("Invalid key exception");
    }
}
```

Obrázek 85: Dešifrovací metoda

3.2.11 Zakázané a povolené metody

Jsou povoleny metody GET, POST, OPTIONS, PUT a DELETE. Metoda OPTIONS musí být povolena kvůli CORS.

Metody TRACE, CONNECT a HEAD vrací výjimku *method_not_allowed*, jak lze vidět na obrázku číslo 86.

```
if(((HttpServletRequest) req).getMethod().equals("TRACE") || ((HttpServletRequest) req).getMethod().equals("CONNECT")
  ||((HttpServletRequest) req).getMethod().equals("HEAD")){
    response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, s: ((HttpServletRequest) req).getMethod() + " is not allowed.");
    return;
}
```

Obrázek 86: Zakázané metody

3.2.12 Validace inputu

Validace inputu na straně serveru je velmi kritická část bezpečnosti. Jsou validována všechna příchozí data.

Příkladem může být validace uživatelského jména ve třídě *UserBody* v balíčku *model* na obrázku číslo 87. Aby tato validace byla funkční, je nutné označit objekt, který chceme validovat pomocí anotace *@Valid*.

```
@Column(name = "first_name", length = 40, nullable = false)
@Size(min=3, max=40,  message = "Size must be 3-40 chars.")
@NotBlank(message = "Firstname cannot be blank.")
@Pattern(regexp = "^[\\p{L}\\s.'\\-,]+$", message = "First name does not seem to look like name. Try again")
private String firstName;
```

Obrázek 87: Validace jména

Dalším typem validace je použití vlastních anotací. V kapitole 3.2.5 je popisována politika hesla. Pokud potřebujeme použít tuto validaci hesla, přidáme vlastní vytvořenou anotaci nad dané pole, jak lze vidět na obrázku číslo 90.

```
@ValidPassword(message = "Password is not ok.")
private String newPassword;
```

Obrázek 88: Validace hesla pomocí vlastní anotace

V určitých metodách jsou očekávány parametry *year* a *month*. Na tyto parametry je použita následující validace pomocí regulárních výrazů.

```
if(month.matches( regex: "^(1[0-2]|[1-9])$") && year.matches( regex: "^\\d{4}$")){
```

Obrázek 89: Validace roku a měsíce

Validace na obrázku číslo 90 nepřijme v URL žádný jiný parametr, který není převoditelný na long.

```
@GetMapping(value = "/{id}")
@Secured({("ROLE_USER"), ("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public void generateReport(OAuth2Authentication auth, HttpServletResponse response, @PathVariable("id") long id) {
```

Obrázek 90: Parametr long

Na dalším obrázku číslo 91 lze vidět validaci datumů. Je kontrolováno, zda datumy poskytnuté uživatelem jsou validní. Tato metoda také kontroluje, zda poskytnuté datumy jsou validní z hlediska logického.

```
public boolean documentDatesValidity(String approvalTime, String startOfReading, String endOfReading) {
    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm");

    try {
        if(formatter.parse(approvalTime).after(formatter.parse(formatter.format(new Date()))) ||
                formatter.parse(approvalTime).equals(formatter.parse(formatter.format(new Date())))){
            if(formatter.parse(startOfReading).after(formatter.parse(formatter.format(new Date())))
                    && formatter.parse(endOfReading).after(formatter.parse(formatter.format(new Date()))
                if(formatter.parse(endOfReading).after(formatter.parse(startOfReading))){
                    return true;
                }
        }
    }catch (ParseException e){
        return false;
    }
    return false;
}
```

Obrázek 91: Validace datumu

Neméně důležitou validací je kontrola, zda poskytnutý dokument je opravdu dokument typu PDF. K tomuto účelu je použita knihovna *org.apache.tika*. Tuto validaci lze vidět na obrázku 92.

```
public boolean checkFileValidity(MultipartFile multipartFile) {
    try {
        if(!multipartFile.isEmpty())
        {
            Tika tika = new Tika();
            String detectedType = tika.detect(multipartFile.getBytes());
            if (detectedType.equals("application/pdf")) {
                return true;
            }
        }
    } catch (IOException e){
        return false;
    }
    return false;
}
```

Obrázek 92: Validace typu dokumentu

Je umožněno vyhledávání dokumentu podle titulu. Titul může obsahovat alfanumerické znaky, whitespace znaky a vybrané speciální znaky. K tomuto účelu byla použita validace pomocí regulárního výrazu, která je zobrazena na obrázku 93.

```
@GetMapping("/find")
@Secured({{"ROLE_USER"),("ROLE_DOCUMENT_CREATOR"), ("ROLE_ADMIN")})
public List<Document> findDocumentByTitle(@RequestParam("name")String name, OAuth2Authentication auth){
    if(name.trim().length() > 0 && name.length() <= 15 && name.matches( regex: "^[a-zá-žA-ZÁ-Ž0-9\\)\\(\\!\\?\\s]+$")){
```

Obrázek 93: Validace titulu

Při vyhledávání jména jsou pomocí regulárního výrazu povoleny jen znaky, které může jméno obsahovat.

3.2.13 Obrana proti útoku XSS

První formou obrany proti XSS je validace vstupních dat v aplikaci, která je rozebrána v kapitole 3.2.12. Další forma obrany je definice hlaviček, které jsou důležité v obraně proti XSS. Jedná se o hlavičky X-XSS-Protection:1; mode=block, Content-Type-Options: nosniff; a Content-Security-Policy.

Hlavičky X-XSS-Protection 1; mode=block a Content-Type-Options: nosniff jsou nastaveny frameworkem Spring Boot ve výchozím stavu [19]. Content-Security-Policy je přidáno do hlaviček způsobem ukázaným na obrázku 63.

3.2.14 Obrana proti útoku SQL injection

Prvotní obrana proti SQL injection je kontrola vstupních dat od uživatele. Druhým obranným mechanismem je používat parametrizované dotazy. Jsou používané parametrizované dotazy způsobem zobrazeným na obrázku 94.

```
Query query = entityManager.createQuery( s: "SELECT count(d.id) from Document d INNER JOIN d.documentsForUsers ud " +
        "INNER JOIN ud.sharingType dt WHERE dt.id = :val1 AND ud.document.id = :val2");
query.setParameter( s: "val1",  o: 1L);
query.setParameter( s: "val2", documentId);
```

Obrázek 94: Parametrizované dotazy

3.2.15 Obrana proti útokům hrubou silou

Třída, která je zodpovědná za obranu proti tomuto typu útoku je *LoginService* v balíčku *service*. K implementaci byla použita knihovna *com.google.guava.*

Po pěti neúspěšných pokusech o přihlášení v řadě je vrácena chyba se zprávou, že uživatel je zablokovaný. Blokace trvá 4 hodiny.

```java
public void loginFailed(String username, String key) {
    int attempts = 0;
    try {
        attempts = attemptsCache.get(key);
    } catch (ExecutionException e) {
        attempts = 0;
    }
    attempts++;
    attemptsCache.put(key, attempts);
    loginLogger.warn("Unsuccessful login:  " + username + " ip address:" + key);
}
```

Obrázek 95: Metoda loginFailed

Metody této třídy se volají ve třídě *UserDetailsService* v balíčku *service*. Při počátku autentizace se zkontroluje, zda daná IP adresa je zablokovaná. Pokud ano, autentizační proces je ukončen a je vráceno chybové hlášení uživateli. Metody *loginFailed* a *loginSuccessful* se volají v závislosti na tom, zda poskytnuté přihlašovací údaje jsou validní.

```java
String ip = getClientIP();
if (loginService.isBlocked(ip)) {
    throw new RuntimeException("You are blocked.");
}
```

Obrázek 96: Kontrola blokace IP adresy

3.3 Bezpečnostní mechanismy na frontendu

3.3.1 Oauth2

Poslání autentizační žádosti probíhá způsobem zobrazeným na obrázku 97. Následující žádost je poslána pomocí technologie AJAX na end-point *oauth/token* metodou POST.

```javascript
function authenticateUser(email, password) {
 $.ajax({
    url: "https://docmanager.com:8443/oauth/token",
    type: 'POST',
    dataType: 'json',
    headers: {"Authorization": "Basic bXktY2xpZW50Og==", "Content-Type": "application/x-www-form-urlencoded"},
    data: {
        "password": password,
        "username": email,
        "grant_type": "password"
      },
    success: function(result) {
        sessionStorage.access_token = result.access_token;
        sessionStorage.refresh_token = result.refresh_token;

        window.location.replace("index.html");
    },
```

Obrázek 97: Autentizační žádost

Tokeny jsou ukládány do sessionStorage po úspěšném obdržení. Ještě předtím, než dojde k načtení celé stránky, je poslána žádost na end-point */oauth/valid*, kde dochází ke kontrole, zda token uložený v sessionStorage je opravdu validní token. Pokud je vrácena odpověď serverem, že token není validní, uživatel je přesunut na přihlašovací stránku.

K běžné žádosti po autentizaci slouží metoda na obrázku 98. Důležité je, že přístupový token musí být posílán v hlavičce *Authorization: bearer [přístupový token]*. Posílání přístupového tokenu v URL není bezpečné.

```
method(type_, url_, data_, callback, type){
  $.ajax({
    url: proxy + url_,
    type: type_,
    contentType: "application/json",
    headers: {
      'Authorization': 'bearer ' + sessionStorage.access_token
    },
    data: data_,
    success: function(result){
        if(callback !== ""){
          callback(result);
        }
```

Obrázek 98: Běžná žádost na server

Nesmírně důležitá je i reakce na chyby. Jelikož přístupový token má životnost patnáct minut, je nutné reagovat na vypršení přístupového tokenu tak, aby to neomezovalo uživatele. Za předpokladu, že server vrátí chybu 401, znamená to, že uživateli vypršela platnost přístupového tokenu. Na tuto chybu je reagováno zavoláním metody *refreshToken*. Prostřednictvím této metody je odeslána žádost na server s použitím obnovovacího tokenu a je žádáno o nový přístupový token. Uživatel obdrží nové tokeny za předpokladu, že obnovovací token je validní. V případě, že žádost o nový token selže, bude smazáno lokální úložiště a uživatel bude přesunut na přihlašovací stránku.

```
  statusCode: {
    401: (response) => {
          refreshToken();
          setTimeout(() => {
            this.method(type_, url_, data_, callback, type);
          }, 300);
    }
  }
});
```

Obrázek 99: Zachycení chyby 401

```
export const refreshToken = () => {
  $.ajax({
        'url': proxy + "oauth/token",
        'type': 'POST',
        'content-Type': 'x-www-form-urlencoded',
        headers: {"Authorization": "Basic bXktY2xpZW50Og=="},
        data: {
          refresh_token: sessionStorage.refresh_token,
          grant_type: 'refresh_token'
        },
        'success': function (result) {
          sessionStorage.access_token = result.access_token;
          sessionStorage.refresh_token = result.refresh_token;
        },
        'error': function (XMLHttpRequest, textStatus, errorThrown) {
          sessionStorage.clear();
          logout();
          window.location.replace("login.html");
        }
```

Obrázek 100: Žádost o nový přístupový token

Při žádosti o nový přístupový token prostřednictvím obnovovacího tokenu musíme do dat vložit položky *grant_type* a *refresh_token*. Pří úspěšném dotazu dojde k uložení nových tokenů do sessionStorage. Na obrázku číslo 101 lze vidět, jaká je reakce na chybu 401. Je okamžitě odeslán nový požadavek na end-point *oauth/token*, aby mohl být získán nový přístupový token. Po úspěšném získání nového přístupového tokenu dojde k odeslání stejné žádosti, která předtím vrátila chybu 401.

Name	Stat...	Type	Initiator	Size	Time
documents	401	xhr	bundle.js...	2.0 ...	34 ...
token	200	xhr	bundle.js...	2.0 ...	242...
documents	200	xhr	bundle.js...	716 B	18 ...

Obrázek 101: Chyba 401

Dalším důležitým obranným prvkem je odhlášení. Je nutné smazat lokální úložiště a odeslat zprávu na server, že uživatel se odhlásil.

```
logout(){
  $.ajax({
    'url': proxy + "oauth/revoke-token",
    'type': 'POST',
    'headers': {
      'Authorization': 'Bearer ' + sessionStorage.access_token
    },
    contentType: "application/json",
    'success': function (result) {
      window.location.replace("login.html");
      sessionStorage.clear();
    },
    'error': function (XMLHttpRequest, textStatus, errorThrown) {
      sessionStorage.clear();
      window.location.replace("login.html");
    }
```

Obrázek 102: Odhlášení

3.3.2 XSS

Všechna příchozí data jsou kontrolována pomocí knihovny *xss-filters.js*. Tato knihovna obsahuje několik metod. Použití správných metod je závislé na tom, jak s přijatými daty pracujeme a kam je vkládáme. Na obrázcích 103 a 104 lze vidět dva příklady, jak jsou příchozí data kontrolována pomocí dvou rozdílných metod.

```
<span class="msg-title" id="msg_title">${xssFilters.inHTMLData(data[i].title)}</span>
```

Obrázek 103: Metoda inHTMLData

```
$('#roles').append(`<option value="${xssFilters.inDoubleQuotedAttr(data.rolesb[i].authority)}">$
```

Obrázek 104: Metoda inDoubleQuotedAttr

3.3.3 Další bezpečnostní prvky

Je využíváno sessionStorage k ukládání tokenů, protože sessionStorage automaticky maže svůj obsah po vypnutí prohlížeče a zavření okna. Nevýhodou je, že aplikace nemůže fungovat s více otevřenými okny.

Je využíváno vypnutí atributu *autocomplete* u inputů. Příkladem může být přihlašovací stránka. Pokud je tento atribut povolený, dochází k zobrazení předchozích vložených hodnot.

```
<input type="email" id="login-email" placeholder="Your email" required minlength="3" maxlength="40" autocomplete="off"/>
```

Obrázek 105: Vypnutí atributu autocomplete

Jsou kontrolována vstupní data pomocí regulárních výrazů. Na obrázku číslo 106 lze vidět kontrolu dat pomocí regulárního výrazu.

```
var regex = new RegExp("^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#\$%\^&\*])(?=.{8,})");

if(name.length > 0 && surname.length > 0 && roles_.length > 0){
  if(regex.test(pass1) && regex.test(pass2) && pass1 == pass2){
```

Obrázek 106: Kontrola dat pomocí regulárního výrazu

3.4 Konfigurace webového serveru IIS

S pomocí souboru CIS_Microsoft_IIS_10_Benchmark verze 1.1.1 byla provedena konfigurace webového serveru [21]. V následujících bodech budou popsány nejdůležitější části konfigurace.

Directory browsing

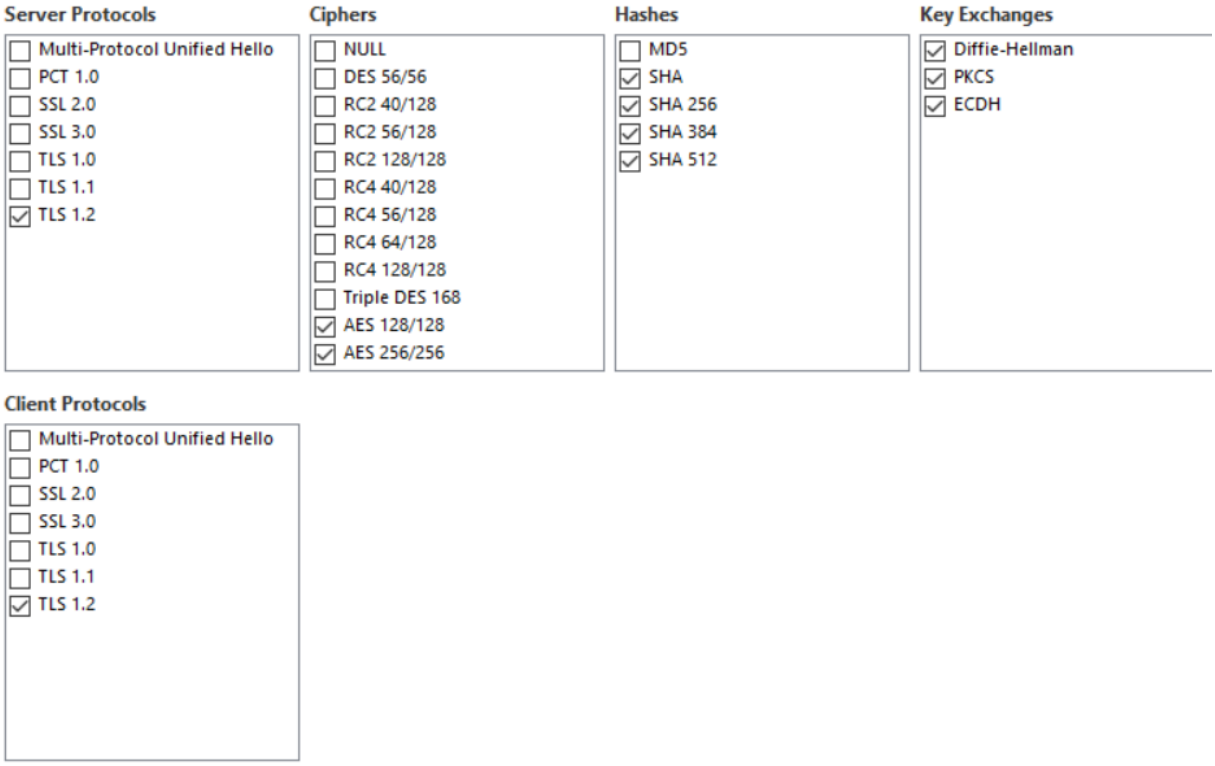
Bylo zkontrolováno, že webový server neumožňuje directory browsing. Na obrázku 107 lze vidět provedený dotaz, který vrací, zda je povoleno directory browsing.

```
PS C:\WINDOWS\system32> Get-WebConfigurationProperty -Filter system.webserver/directorybrowse -PSPath iis:\ -Name Enabled | select Value

Value
-----
False
```

Obrázek 107: Zakázané directory browsing

HTTPS

K nastavení HTTPS byl použit stejný certifikát, který je používán na REST serveru. Pro funkčnost aplikace je nutné, aby uživatel, který chce přistoupit k webovému serveru, měl certifikát webového serveru vložen mezi důvěryhodné certifikační autority na svém počítači. Byly zakázány všechny nižší protokoly než TLS1.2 a slabé šifry. K tomuto účelu byl použit nástroj *IIS Crypto 3.0*. Na obrázku 108 lze vidět některá nastavení týkající se šifrování na webovém serveru IIS.



Obrázek 108: Nastavení šifrování

Filtrování žádostí

Byl nastaven atribut *maxAllowedContentLength* na 30000000. Jedná se o maximální velikost HTTP žádosti měřené v bytech, která může být poslána klientem web serveru [21]. Atribut *maxURL* nastavuje maximální délku v bytech, které může být přijaté IIS [21]. Je používána hodnota 4096. Atribut *maxQueryString* popisuje, jak maximálně může být dotazovaný řetězec dlouhý [21]. Je nastavena hodnota na 2048. Dalším používaným atributem je *allowHightBitCharacters*, který je nastavený na hodnotu false. Při použití tohoto atributu budou zamítnuté žádosti, pokud by byly přítomny vysokobitové znaky v URL [21]. Dalším používaným atributem je *allowDoubleEscaping* nastavený na hodnotu false. Tento atribut brání proti útokům, které spoléhají na dvakrát šifrované žádosti [21].

```
PS C:\WINDOWS\system32> Get-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter "system.webServer/security/requestFiltering/requestLimits" -name "maxUrl"


ItemXPath                   : /system.webServer/security/requestFiltering/requestLimits
IsInheritedFromDefaultValue : True
IsProtected                 : False
Name                        : maxUrl
TypeName                    : System.Int64
Schema                      : Microsoft.IIs.PowerShell.Framework.ConfigurationAttributeSchema
Value                       : 4096
IsExtended                  : False
```

Obrázek 109: Hodnota maxURL

Hlavička server

Bylo nakonfigurováno, aby webový server nevracel hlavičku server. Byl vložen příkaz

zobrazený na obrázku 110 do příkazové řádky.

```
%systemroot%\system32\inetsrv\appcmd.exe set config -
section:system.webServer/security/requestFiltering /removeServerHeader:"True"
/commit:apphost
```

Obrázek 110: Konfigurace hlavičky server [21]

Bezpečnostní hlavičky

Na obrázku číslo 111 lze vidět hlavičky, které webový server používá. Hlavičky *X-XSS-Protection*, *X-Content-Type-Options* a *CSP* jsou vysvětleny v kapitole 2.2.4.

```
<httpProtocol>
    <customHeaders>
      <clear />
        <remove name="X-Frame-Options" />
        <add name="X-XSS-Protection" value="1; mode=block" />
        <add name="X-Content-Type-Options" value="nosniff" />
        <add name="Cache-Control" value="must-revalidate, no-cache, no-store, max-age=0, s-maxage=0, pre-check=0, post-check=0" />
        <add name="Pragma" value="no-cache" />
        <add name="X-Frame-Options" value="deny" />
        <add name="Content-Security-Policy" value="default-src 'none'; script-src 'self'; connect-src https://docmanager.com:8443/; img-src 'self'; style-src 'self';
            frame-ancestors https://docmanager.com/" />
        <add name="Strict-Transport-Security" value="max-age=15465600; includeSubDomains" />
    </customHeaders>
</httpProtocol>
```

Obrázek 111: Bezpečnostní hlavičky na webovém serveru IIS

Hlavička *X-Frame-Options* zakazuje vkládání webu či jeho částí do jiných webů použitím HTML tagů *frame* a *iframe*. Hodnota *deny* zakazuje jakékoli vkládání [12]. Bez použití této hlavičky by mohla být aplikace zranitelná vůči útoku Clickjacking.

Hlavička *Cache-Control* je použita ke specifikaci kešovací politiky prohlížeče v klientských žádostech i odpovědích serveru. Politiky zahrnují informace o tom, jak je zdroj kešován, kde je kešován a jaká je maximální doba před vypršením [13]. Nastavení hlavičky *Control-Cache* na obrázku 111 je doporučeno dokumentem OWASP Testing Guide v4.

Hlavička *pragma* s hodnotou *no-cache* má podobný význam jako hlavička *Cache-Control*. Je použita, protože hlavička *Cache-Control* není kompatibilní s HTTP/1.0 [14].

Strict-Transport-Security přikazuje prohlížeči, že má po stanovenou dobu komunikovat pouze přes HTTPS protokol. Hodnota *includeSubDomains* znamená, že i subdomény musí být zabezpečené a mít SSL certifikát [15].

Zakázané metody

Zakázané metody na webovém serveru jsou popsány v kapitole 2.10.

Reakce na chyby

Byl vytvořen speciální HTML soubor, který se zobrazuje, pokud dojde k chybě na webovém serveru IIS. Tato stránka se nachází ve složce *web_server/errors* na DVD.

```
<httpErrors errorMode="Custom">
    <remove statusCode="502" subStatusCode="-1" />
    <remove statusCode="501" subStatusCode="-1" />
    <remove statusCode="500" subStatusCode="-1" />
    <remove statusCode="412" subStatusCode="-1" />
    <remove statusCode="406" subStatusCode="-1" />
    <remove statusCode="405" subStatusCode="-1" />
    <remove statusCode="403" subStatusCode="-1" />
    <remove statusCode="404" subStatusCode="-1" />
    <remove statusCode="401" subStatusCode="-1" />
    <error statusCode="401" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="404" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="403" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="405" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="406" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="412" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="500" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="501" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
    <error statusCode="502" prefixLanguageFilePath="" path="/errors/error.html" responseMode="ExecuteURL" />
</httpErrors>
```

Obrázek 112: Reakce na chyby – webový server

4. Testování

Testování bylo provedeno dle popisu v kapitole 2.3. Na DVD je přiložena složka *penetrační_testy*, kde se nacházejí soubory, které obsahují výsledky některých testů. Některé výsledky nebyly přidány do bakalářské práce z důvodu jejich velké rozsáhlosti. Budou zmíněny jen testy, které jsou relevantní. Výpis všech testů lze nalézt v souboru *penetracni_testy.docx* na DVD.

4.1 – Testování sběru informací (OTG-INFO)

Otisk webového serveru (OTG-INFO-002)

Byla odeslána žádost HEAD na adresu *https://docmanager.com*. Na obrázku číslo 113 je zaznamenaná odpověď webového serveru. Důležité je, že nebyla vrácena hlavička Server. Tato hlavička může prozradit informace o verzi a typu serveru. Pro účely tohoto testu byla povolena metoda HEAD.

```
HTTP/1.1 301 Moved Permanently
Cache-Control: must-revalidate, no-cache, no-store, max-age=0, s-maxage=0, pre-check=0, post-check=0
Pragma: no-cache
Content-Length: 149
Content-Type: text/html; charset=UTF-8
Location: https:///
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: deny
Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src https://docmanager.com:8443/; img-src 'self'
; style-src 'self'; frame-ancestors https://docmanager.com/
Strict-Transport-Security: max-age=60000; includeSubDomains
Date: Thu, 19 Sep 2019 16:12:31 GMT
Connection: close
```

Obrázek 113: Test OTG-INFO-002, první žádost

V druhém kroku byla poslána neplatná žádost na webový server. Ani v tomto případě

nebyla vrácena hlavička Server.

```
docmanager.com [192.168.0.161] 80 (http) open
GET / JUNK/1.0
HTTP/1.1 400 Bad Request
Content-Type: text/html; charset=us-ascii
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 19 Sep 2019 10:09:55 GMT
Connection: close
Content-Length: 311
```

Obrázek 114: Test OTG-INFO-002, druhá žádost

Zkontrolování komentářů a metadat, které by mohly vést k úniku informací (OTG-INFO-005)

Bylo zkontrolováno, že aplikace neobsahuje komentáře v html/css a javascriptovém kódu, které by mohly vést k zneužití aplikace. Bylo otestováno, že verze HTML je validní. V posledním kroku bylo otestováno, že aplikace neobsahuje žádné meta tagy, které by mohly aplikaci poškodit. Všechny soubory s metadaty jsou nahrávány uživateli, takže je na nich, aby provedli kontrolu.

Identifikace vstupních bodů aplikace (OTG-INFO-006)

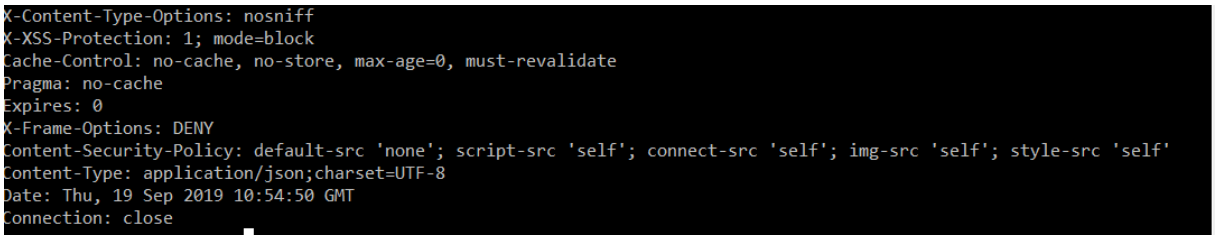
Cílem toho testu bylo identifikovat, jaké jsou používané HTTP metody a parametry. K tomuto účelu byl použit nástroj Zed Attack Proxy (ZAP). Výslednou tabulku nalezených bodů aplikace lze nalézt v souboru *penetracni_testy.docx* na DVD.

Mapování cest aplikace (OTG-INFO-007)

Cílem tohoto testu bylo identifikovat strukturu aplikace. K tomu účelu byl použit ZAP a kód aplikace, protože je používáno REST API, které nemůže být namapováno pomocí nástroje ZAP.

Otisk použitého webového frameworku (OTG-INFO-008)

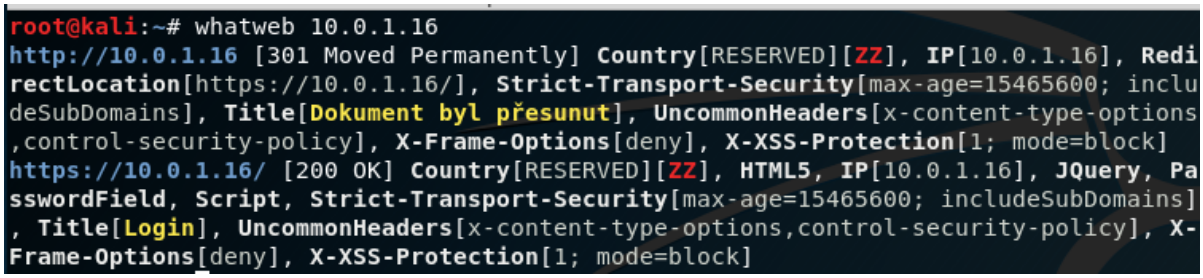
Cílem tohoto testu bylo identifikovat použité frameworky, které aplikace používá. Byla poslaná žádost HEAD na server. Obrázek číslo 115 zobrazuje odpověď, kterou poskytl server. Je patrné, že z této hlavičky nelze vyčíst, na jakém frameworku server běží. Pro účely tohoto testu byla povolena HEAD metoda na serveru.



Obrázek 115: OTG-INFO-008

Bylo ověřeno, že má server vypnuté actuator end-pointy. Byly poslány žádosti na všechny tyto end-pointy (například *actuator/health*). Tímto bylo otestováno, že aplikace má tuto funkcionalitu vypnutou.

Bylo možné identifikovat, jaké technologie jsou na frontendu používané. Po prohlídnutí kódu byl identifikován framework jQuery verze 3.4.1. Pomocí nástroje *whatweb* byl získán otisk webové aplikace, který také detekoval framework jQuery.



Obrázek 116: Otisk webové aplikace

Otisk webové aplikace (OTG-INFO-009)

Tento test byl proveden pomocí nástroje *whatweb*, který vrátil výsledek na obrázku 116. Je patrné, že aplikace nevyužívá žádnou známou aplikaci (Wordpress, phpBB, Mediawiki…). Byl prohlídnut HTML kód a byly hledány jasné identifikátory aplikací (například informace v meta elementu, které mohou identifikovat využívanou aplikaci). Aplikace nevyužívá ani žádné speciální soubory, které by mohly identifikovat používanou aplikaci.

4.2 - Testování správy konfigurace (OTG-CONFIG)

Test konfigurace aplikační platformy (OTG-CONFIG-002)

Bylo ověřeno, že aplikace vrací uživatelsky vytvořené chybové stránky. Bylo zjištěno, že dochází k logování úspěšného/neúspěšného pokusu o přihlášení, odhlášení, úspěšného/neúspěšného přístupu k dokumentu, manipulace s dokumenty, uživateli a skupinami.

Zpracování různých souborů obsahující citlivé informace (OTG-CONFIG-003)

V tomto testu bylo zkoumáno, jak aplikace pracuje s různými typy souborů. Aplikace odmítá všechny soubory, které nejsou validní PDF soubory.

Kontrola starých souborů, záloh, zda neobsahují citlivé informace (OTG-CONFIG-004)

Aplikace nemá implementované zálohování. Bylo zkontrolováno, že aplikace neobsahuje staré soubory, které by mohly obsahovat nějaké citlivé informace. Kontrolovány byly i logy, zda neobsahují nějaké citlivé informace (tokeny, hesla…). Nebyly nalezeny žádné nedostatky.

Vyhodnocení infrastruktury a administrátorská rozhraní (OTG-CONFIG-005)

Administrátorské rozhraní se zobrazuje na základě role uživatele v aplikaci, není tedy možné přistoupit k uživatelskému rozhraní pomocí přepsání URL nebo brute-forcing obsahu serveru. Toto testování úzce souvisí s testy OTG-AUTHZ-002 a OTG-AUTHZ-004, u kterých nebyly nalezeny žádné nedostatky.

Byl prohlídnut zdrojový kód a bylo zkontrolováno, že autorizační a autentizační model zajišťuje jasné oddělení funkcí mezi normálními uživateli a administrátory. Nebyly nalezeny žádné nedostatky.

Testování HTTP metod (OTG-CONFIG-006)

Cílem testu bylo zjištění, zda jsou podporované pouze metody GET a POST na webovém serveru IIS. Na obrázku 117 lze vidět, že metoda OPTIONS je zablokovaná. Stejně jako metody HEAD, TRACE, DELETE, CONNECT a PUT na webovém serveru IIS.

```
C:\Users\patri\Desktop\nc>nc docmanager.com 80
OPTIONS /HTTP/1.1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN""http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>Bad Request</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html; charset=us-ascii"></HEAD>
<BODY><h2>Bad Request - Invalid Verb</h2>
<hr><p>HTTP Error 400. The request verb is invalid.</p>
</BODY></HTML>
```

Obrázek 117: OPTIONS – webový server IIS

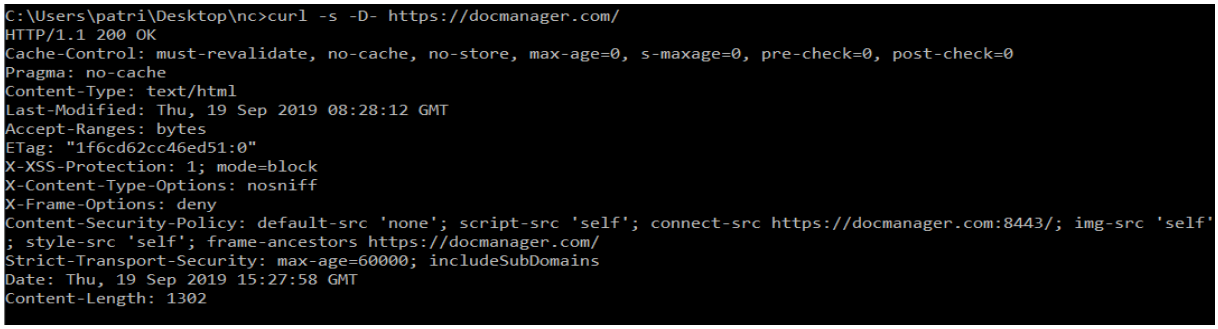
Bylo otestováno, zda poslání žádosti JEFF /HTTP/1.0 na stránce, která nutí přihlášení, vrátí 405 nebo 501 chybovou stránku.

Obrázek 118: OTG-CONFIG-006, druhá část

Na REST serveru bylo otestováno, že jsou umožněny pouze metody GET, PUT, DELETE, POST a OPTIONS. Metody DELETE, PUT a OPTIONS jsou na REST serveru nutností. Metoda OPTIONS musí být použita, protože je používán CORS standard. Při použití metod CONNECT, HEAD a TRACE server vrátil chybu *method_not_allowed*. Metody DELETE a PUT mohou být využívány pouze rolemi **ADMIN** a **DOCUMENT_CREATOR**. Mazání uživatelů a dokumentů pomocí metody DELETE je povoleno pouze uživatelům s rolí **ADMIN**.

Testování HSTS (OTG-CONFIG-007)

Následujícím testem bylo prokázáno, že je správně používána hlavička Strict-Transport-Security-Policy.



Obrázek 119: OTG-CONFIG-007

4.3 – Testování správy identit (OTG-IDENT)

Testování definice rolí (OTG-IDENT-001)

Bylo ověřeno, že definované role vyhovují principu minimálních privilegií.

Testování registrace uživatele (OTG-IDENT-002)

Bylo testováno, zda každý uživatel má možnost se sám zaregistrovat. Byla tedy poslaná žádost o registraci běžným klientem na end-point */users* metodou POST. Tato žádost byla zamítnuta. Bylo ověřeno, že registrace nových uživatelů může být provedena jen

administrátorem.

Dále bylo otestováno, že stejná osoba se nemůže vícekrát zaregistrovat. Byla tedy poslána žádost na end-point */users* metodou POST se stejnými údaji dvakrát. Server tuto žádost zamítl a vrátil chybu. V aplikaci funguje jako unikátní klíč email, takže pokud by byl změněný v předchozí žádosti pouze email, žádost by byla přijata.

Bylo otestováno, že uživatelé nemohou měnit své role. Byly úspěšně vytvořeny nové role pro uživatele, ale pouze pokud žádost o změnu rolí poslal administrátor. Jiné žádosti byly zamítnuty serverem. Identita je ověřována administrátorem.

Testování nastavování uživatelského účtu (OTG-IDENT-003)

Bylo zkoumáno, zda je v aplikaci nějaký mechanismus prověřování žádostí, které slouží k vytvoření nového účtu nebo k nastavování rolí. Byl nalezen autorizační mechanismus, který vyžaduje, aby tyto operace mohl provést pouze administrátor. Ostatní žádosti byly zamítnuty serverem.

Bylo otestováno, že je umožněno administrátorům vytvářet nové klienty i administrátory. Bylo zkontrolováno, že administrátoři a uživatelé nemohou nastavovat jiným účtům větší privilegia než jejich vlastní. Bylo zjištěno, že administrátor si může sám sobě vzít oprávnění. Pokud je uživatel deaktivován/zbaven privilegií, soubory nejsou mazány, ale uživatel již k souborům nemá přístup a emaily odeslané aplikací mu jíž nadále nedochází. Nebyly nalezeny nedostatky.

Testování výčtu účtu a uhodnutelného přihlašovacího jména (OTG-IDENT-004 a OTG-IDENT-005)

Cílem tohoto testu bylo ověřit, zda aplikace vrací stejné odpovědi pro každou klientovu žádost, která produkuje neúspěšný pokus o přihlášení.

Byla poslána žádost se správným přihlašovacím jménem a heslem, byla zaznamenána odpověď a délka odpovědi.

V druhém kroku byla poslána žádost s validním přihlašovacím jménem a špatným heslem a byla zaznamenána odpověď generována aplikací. Server vrátil odpověď *Bad credenentials*.

V třetím kroku byla poslána žádost se špatným přihlašovacím jménem a validním heslem a byla zaznamenána odpověď serveru. Opět byla vrácena odpověď *Bad credentials*.

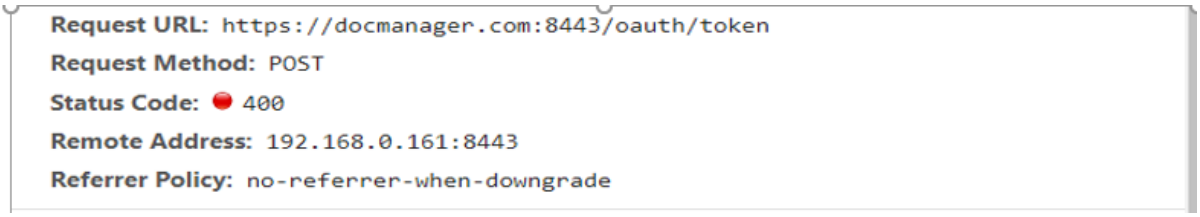
Bylo tedy tímto testem ověřeno, že aplikace vrací pro různé kombinace přihlášení generické výsledky. Aplikace by měla vracet stejnou chybu a stejnou délku na různé nesprávné

žádosti. Nebyly nalezeny žádné nedostatky.

4.4– Testování autentizace (OTG-AUTHN)

Testování, zda přihlašovací údaje jsou přenášeny přes šifrovaný kanál (OTH-AUTHN-001)

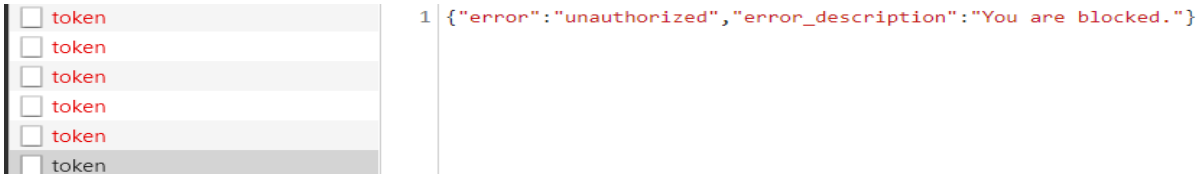
Na obrázku 120 lze vidět, že přihlašovací údaje jsou posílány šifrovaně.



Obrázek 120: OTG-AUTHN-001

Testování slabého zamykacího mechanismu (OTG-AUTHN-003)

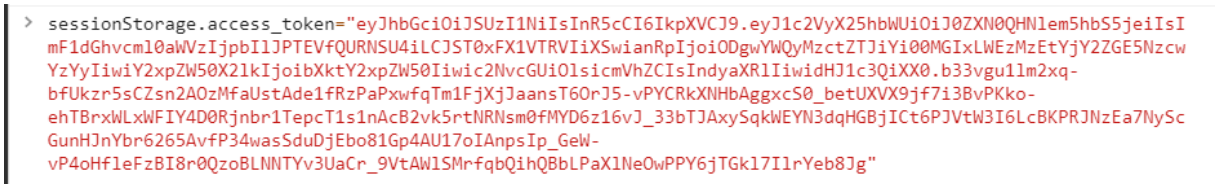
Byly provedeny tři neúspěšné pokusy o přihlášení následované jedním úspěšným pokusem. Následovalo pět neúspěšných pokusů o přihlášení a jeden pokus se správnými údaji. V tomto bodě byla vrácena chyba *„You are blocked.“*. Další pokus o přihlášení byl proveden o pět minut později. Následovaly pokusy po deseti minutách, po jedné hodině, po dvou hodinách a po třech hodinách. Blokování bylo stále funkční i po uběhnutí těchto časových intervalů. Po uplynutí čtyř hodin bylo povoleno se opět přihlásit. Bylo zjištěno, že odblokování se děje po čtyřech hodinách automaticky aplikací. Nebyly nalezeny nedostatky.



Obrázek 121: OTG-AUTHN-003

Testování obcházení autentizačního schématu (OTG-AUTHN-004)

Bylo otestováno, že není možné přímo přejít na stránku *docmanager.com/index.html* bez autentizace. Bylo zkontrolováno, že bez validního přístupového tokenu se nelze dostat na tuto stránku. Na obrázku číslo 122 lze vidět uložení falešného přístupového tokenu do sessionStorage pomocí konzole na přihlašovací stránce.



Obrázek 122: OTG-AUTHN-004

S falešným přístupovým tokenem na obrázku 122 uloženým v sessionStorage bylo otestováno, že aplikace neumožní přejít na stránku *docmanager.com/index.html*, která vyžaduje autentizaci. Před načtením dokumentu je poslána žádost na server, zda je tento token validní, proto tímto způsobem nelze obejít autentizační schéma. Bylo otestováno, že nelze obejít autentizační schéma pomocí útoku SQL injection prostřednictvím testu OTG-INPVAL-005. Nebyly nalezeny žádné nedostatky.

Testování zranitelné cache prohlížeče (OTG-AUTHN-006)

Byla úspěšně otestováno, že není možné načíst zpětně informace o přihlášení a jiných podobných citlivých datech (například email po loginu, hesla). Testem bylo vložení senzitivních informací (email, heslo) do aplikace a následovalo odhlášení. Po odhlášení bylo zkontrolováno, že přes tlačítko zpět se nelze dostat k předchozím stránkám. Nebyl umožněn přístup na danou stránku přes tlačítko zpět. Bylo zkontrolováno, že se do cache prohlížeče neukládají žádná citlivá data. Nebyly nalezeny žádné nedostatky.

Testování politiky slabého hesla (OTG-AUTHN-007)

Bylo otestováno, že je vyžadováno minimálně 8 znaků dlouhé heslo, jeden speciální znak, velké a malé písmeno a číslice. Bylo zjištěno, že aplikace umožňuje provést operaci změny hesla v libovolnou dobu. Není implementována nucená změna hesla v daných časových intervalech. Podařilo se úspěšně nastavit předchozí změněné heslo při třetí žádosti o změnu heslu. Není tedy udržována historie osmi hesel. Bylo potvrzeno, že při tvorbě nového hesla nesmí být heslo totožné.

Testování slabého mechanismus změny hesla nebo resetovacích funkcionalit (OTG-AUTHN-009)

Bylo otestováno, že uživatel bez role admin nemůže změnit heslo ostatním uživatelům. Bylo zachyceno JWT, dekódováno a byla změněna položka *username*, kam bylo vloženo uživatelské jméno uživatele, kterému bylo cílem změnit heslo. JWT byl opět podepsán a poslán na server. Server vrátil chybu *INVALID_TOKEN*. Funkcionalita změny hesla

nepožaduje žádnou speciální otázku od uživatele, ale požaduje od uživatele jeho staré heslo. Heslo musí dodržovat politiku popsanou v testu OTG-AUTHN-007. Bylo potvrzeno, že funkcionalita změny hesla není zranitelná vůči útoku CSRF, protože nejsou používány cookies. Nebyly nalezeny nedostatky.

4.5 – Testování autorizace (OTG-AUTHZ)

Testování Directory traversal (OTG-AUTHZ-001)

Po úspěšném přihlášení stránka funguje jako Single-Page Application, nedochází tedy k přepínání stránek jako je ukázáno na obrázku 123.

```
GET http://docmanager.com/show.asp?view=index.html HTTP/1.1
Host: docmanager.com
```

Obrázek 123: Potencionálně nebezpečné přepínání stránek

Byl testován příkaz na obrázku 124. kde byla vrácena chyba 403 webovým serverem. Nebyly nalezeny žádné nedostatky.

```
GET https://docmanager.com/scripts/..%5c../Windows/System32/cmd.exe?/c+dir+c:\ HTTP/1.1
```

Obrázek 124: OTG-AUTHZ-001

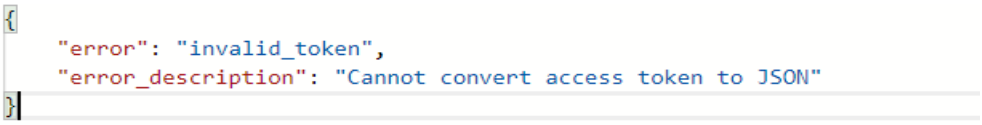
Testování obcházení autorizačního schématu (OTG-AUTHZ-002)

Byly postupně otestovány všechny end-pointy poskytnuté serverem. Bylo otestováno, že neautentizovaný uživatel nemůže přistoupit k chráněnému end-pointu. Byla odeslána žádost bez hlavičky Bearer. Všechny chráněné end-pointy vrátily odpověď zobrazenou na obrázku 125.


```
{
    "error": "unauthorized",
    "error_description": "An Authentication object was not found in the SecurityContext"
}
```


Obrázek 125: OTG-AUTHZ-002, první část

Bylo ověřeno, že při použití náhodného řetězce místo validního přístupového tokenu dochází k vrácení chybového hlášení zobrazeného na obrázku 126.

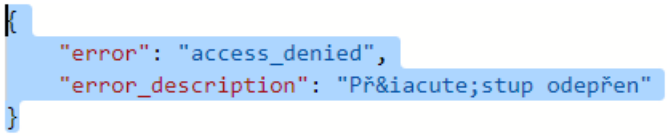


Obrázek 126: OTG-AUTHZ-002, druhá část

Bylo otestováno, že nelze přistoupit k chráněným end-pointům pomocí přístupového tokenu, kterému vypršela životnost. Server vrátil u všech chráněných end-pointů následující odpověď se statusem 401.

Obrázek 127: OTG-AUTHZ-002, třetí část

Bylo otestováno, že uživatel s nedostatečnou rolí, nemůže přistoupit k chráněným end-pointům, ke kterým by neměl mít přístup. Všechny tyto end-pointy vrátily chybu na obrázku 128.



Obrázek 128: OTG-AUTHZ-002, čtvrtá část

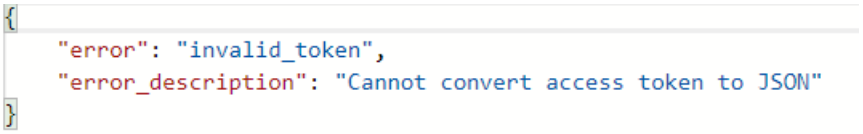
Bylo ověřeno, že právě odhlášený uživatel nemá přístup k části aplikace, kde je vyžadována autentizace. Nebyly nalezeny žádné nedostatky.

Testování eskalace privilegií (OTG-AUTHZ-003)

Cílem testu byla změna hodnot *username* a *authorities* v JWT tokenu. Veřejný i privátní klíč jsou v případě této aplikace neveřejné. Privátní klíč je používán autorizačním serverem, veřejný klíč je používán resource serverem. Aby mohlo dojít ke změně informací v JWT tokenu, musel by útočník odchytit JWT, dekódovat, změnit informace v JWT, podepsat pomocí stejného privátního klíče a poslat na server. Tato situace není možná za předpokladu, že nedojde k odcizení privátního klíče.

Bylo otestováno, že není možné použít JWT se změněnými údaji podepsaným jiným

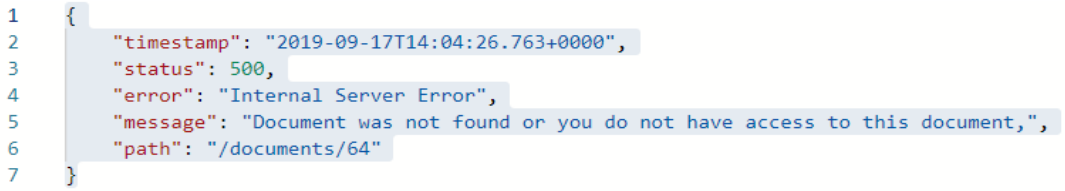
privátním klíčem. Byl zachycen a dekódován původní JWT token poslaný serverem. Byla změněna položka *authorities* v JWT, kde místo **ROLE_USER** bylo vloženo **ROLE_ADMIN**. Zbytek JWT tokenu zůstal nedotčený. Tento token byl podepsaný pomocí stejného algoritmu, ale jiným vygenerovaným privátním klíčem. Po přidání tohoto JWT k žádosti, server vrátil následující odpověď se statusem 401. Nebyly nalezeny žádné nedostatky.



Obrázek 129: OTG-AUTHZ-003

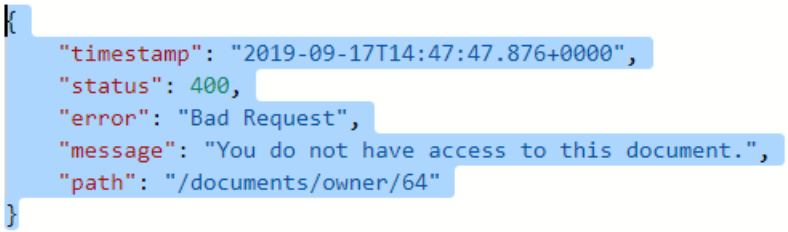
Testování přímých nezabezpečených referencí na objekty (OTG-AUTHZ-004)

Bylo ověřeno, že v bodě */documents/[id dokumentu]* není umožněn přístup k dokumentu uživateli s nedostatečným oprávněním. Testování bylo provedeno tak, že uživatel, který by neměl mít přístup k danému dokumentu, se pokusil získat neoprávněně přístup k dokumentu. Byla poslána žádost na */documents/64* uživatelem, který by neměl mít přístup k danému dokumentu. Výsledná odpověď serveru je zaznamenána na obrázku 130.



Obrázek 130: OTG-AUTHZ-004, první část

Pomocí end-pointu *documents/owner/[id dokument]* bylo otestováno, že uživatel, který má roli **DOCUMENT_CREATOR**, ale daný dokument nevytvořil, nemá přístup k danému dokumentu.



Obrázek 131: OTG-AUTHZ-004, druhá část

V aplikaci se nachází další podobné end-pointy, které byly podobným způsobem také ověřeny. Jedná se o end-pointy */documents/[id dokumentu]* metodou DELETE *a /users/[id uživatele]* metodou DELETE. Nebyly nalezeny žádné nedostatky.

4.6 – Testování řízení relace (OTG-SESS)

Tento soubor testů nebyl prováděn, protože je využíván Oauth2.

4.7 - Testování validace vstupu (OTG-INPVAL)

Testování Reflected XSS (OTG-INPVAL-001) a Stored XSS (OTG-INPVAL-002)

K testování XSS byl použit nástroj OWASP ZAP, pomocí kterého byly zachyceny všechny žádosti, které aplikace posílala. Tyto zachycené žádosti byly použity k fuzz útokům. Řetězce, které byly použity k provedení testu, lze nalézt v dokumentu *xss_payload.txt* na DVD. Tímto způsobem došlo i k otestování všech inputů v aplikaci.

Výsledek testu poukázal, že aplikace je vůči XSS resistentní. Bylo otestováno, že jsou správně zaměňované nebezpečné znaky za HTML entity.

Testování manipulace s HTTP protokolem (OTG-INPVAL-003)

Bylo otestováno, že webový server povoluje pouze metody GET a POST. Byly tedy jednotlivě poslány žádosti OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE a CONNECT. Byly vráceny chyby webovým serverem na všechny poslané metody kromě GET a POST.

Stejný test byl proveden na REST serveru. Bylo otestováno, že metody HEAD, TRACE, CONNECT jsou zakázány. Metody OPTIONS, GET, POST, PUT, DELETE jsou povoleny. REST server potřebuje k vyřízení předběžných žádostí mít povolenou metodu OPTIONS. Metody PUT a DELETE jsou nutné ke správné funkcionalitě aplikace. Nebyly nalezeny žádné nedostatky.

Testování znečištění HTTP parametrů (OTG-INPVAL-004)

Byly poslány celkem tři HTTP žádosti na každý z testovaných end-pointů. Nejdříve byla poslána žádost se standartními parametry a jejich hodnotami (například *documents/history?year=2019&month=9*). Odchycená žádost byla modifikována tak, že se přidal pozměněný parametr následujícím způsobem (*documents/history?year=TMP1&month=TMP2*). Následně se odeslala třetí žádost, která kombinovala oba předchozí testy (*documents/history?year=2019&month=8&year=TMP1&month=TMP2*). Byly porovnány jednotlivé

odpovědi. Pokud by odpověď z třetího žádosti byla odlišná od odpovědi z první žádosti a odpověď z třetí žádosti by byla také odlišná od odpovědi z druhé žádosti, mohla by být aplikace zneužita. Tímto způsobem byly otestovány všechny existující HTTP parametry v aplikaci. Na obrázku číslo 132 je zobrazena odpověď z druhé a třetí žádosti. Nebyly nalezeny žádné nedostatky.

{"timestamp":"2019-09-23T10:20:08.317+0000","status":400,"error":"Bad Request","message":
"Year or month are not in the correct format.","path":"/documents/history"}

Obrázek 132: OTG-INPVAL-005

Testování SQL injection (OTG-INPVAL-005)

Bylo otestováno, že není možné se přihlásit bez znalosti hesla, případně bez znalosti přihlašovacího jména a hesla.

Na end-pointu */oauth/token* byla zkoušena následující data.

- password=" or '1'='1'&username=test@seznam.cz&grant_type=password
- password=1' or '1' = '1&username=1' or '1' = '1&grant_type=password

K přihlašovací stránce byly provedeny ještě dodatečné testy pomocí nástroje OWASP ZAP. Zbylé testy byly provedeny pomocí nástroje sqlmap s příkazem zobrazeným na obrázku 133.

```
sqlmap.py -u "http://192.168.0.161:8080/documents/all?year=2019&month=8" --
headers="Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJ0ZXN0QHNlem5hbS5
jeiIsImF1dGhvcml0aWVzIjpbIlJPTEVfQURNSU4iLCJST0xFX1VTRVIiXSwianRpIjoiZm
Y4YWQ3ZWEtMzc0OS00MGU3LTg0NzAtODI5ZTg4ZmFhMDY3IiwiY2xpZW50X2lkIj
oibXktY2xpZW50Iiwic2NvcGUiOlsicmVhZCIsIndyaXRlIiwidHJ1c3QiXX0.W_ME05ssQ
qhpt6GpTv_FjECfNICXscpIMlY3uiRklNOtVbA3bcTgh71PsIWJoJCeSJYbLy0TOjM9rKm
LHiAMKoND0XzOsIeB0IL5fzHFEZ-riQVMOfT7bF4g6m-V-Ok-z8D7BgXwlq-og-
BKeKbkXHnutMsJrrgS5dE3Xh-
Jh_9WGOsmRQzdeQmolbd2L_aXjmrWyeDWNhTY7IqKBBbnAVrfbmGdZuMv93zRQeh
TtfxKm5ik2mAkzD1b5CdMq_SIVOQfDd1DSonqMcXMmUpDfLyORQzorxWGy9bpjr81
ergcYJHD8koR3iplBVbHtzG8NVu5YXgI3Kwkn292YUHmvg" --dbs --dbms=MySQL --
level=5 --risk=3 --user-agent=SQLMAP --flush-session -t scan_trace6.txt
```

Obrázek 133: sqlmap příkaz

Tímto způsobem byly otestovány všechny existující parametry v aplikaci. Testování neprozradilo žádné nedostatky vůči útoku SQL injection. Výsledky lze nalézt ve složce *sql_injection* na DVD. Nachází se zde celkem tři soubory. Soubor *out.txt* obsahuje kratší popis, jaké všechny testy byly prováděny a celkové výsledky testování každého end-pointu. Soubor *oauth_token.csv* obsahuje další poslané útočné řetězce na endpoint */oauth/token*. Soubor *trace.txt* obsahuje odeslané žádosti s útočnými řetězci. Do souboru *trace.txt* se nepodařilo nahrát všechny odeslané žádosti, protože testy některých end-pointů poslaly velké množství žádosti a výsledné soubory dosahovaly velikosti několika GB. Byly tedy přidány do závěrečného dokumentu pouze žádosti testů, které měly přijatelnou velikost (v MB).

scan_trace1.txt	24.09.2019 12:59	Textový dokument	22 702 kB
scan_trace3.txt	24.09.2019 14:04	Textový dokument	2 486 796 kB
scan_trace5.txt	24.09.2019 13:43	Textový dokument	1 257 140 kB
scan_trace6.txt	24.09.2019 14:16	Textový dokument	2 486 503 kB
scan_trace7.txt	24.09.2019 13:56	Textový dokument	26 571 kB
scan_trace8.txt	24.09.2019 20:04	Textový dokument	49 121 kB
scan_trace9.txt	24.09.2019 12:23	Textový dokument	70 997 kB
scan_trace10.txt	24.09.2019 13:06	Textový dokument	75 436 kB
scan_trace11.txt	24.09.2019 14:23	Textový dokument	0 kB
scan_trace12.txt	25.09.2019 7:03	Textový dokument	6 736 214 kB
scan_trace13.txt	25.09.2019 7:05	Textový dokument	6 741 028 kB
scan_trace17.txt	24.09.2019 15:39	Textový dokument	24 386 kB
scan_trace18.txt	24.09.2019 21:04	Textový dokument	37 197 kB

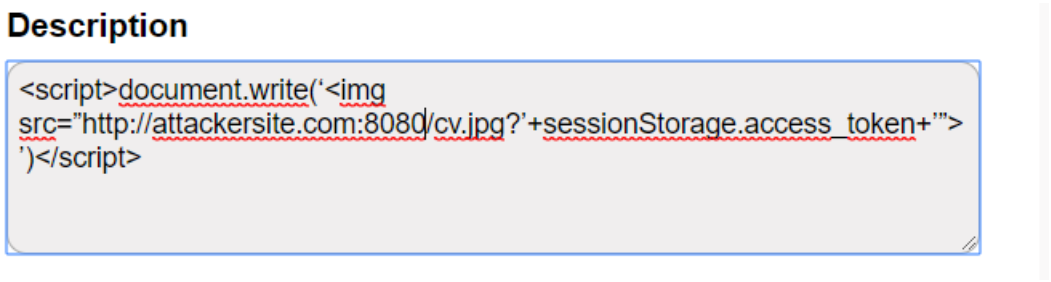
Obrázek 134: Velikosti výsledných souborů s žádostmi

Testování ORM injection (OTG-INPVAL-007)

Byl prozkoumán kód, zda dochází k používání parametrizovaných dotazů a správné kontrole příchozích dat. Je používán whitelist, kterým je definováno, jaké znaky se mohou vyskytnout v příchozích datech. Nejsou povoleny znaky, které by mohly vést k tomuto typu útoku.

Testování skrytých zranitelností (OTG-INPVAL-015)

Byl nahrán dokument s popisem, který by měl poškodit uživatele. Byl použit k testu script na obrázku 135 při tvorbě dokumentu v aplikaci. K funkčnosti tohoto scriptu bylo zapotřebí vytvořit novou aplikaci, která bude schopná reagovat na dané URL a získat z něho hodnotu přístupového tokenu.



Obrázek 135: Útočný script

Cílem tohoto útoku je zmocnit se uživatelova přístupového tokenu. Jakmile by útočník obdržel tímto způsobem přístupový token, mohl by přistupovat k chráněným end-pointům pomocí tohoto tokenu jakožto uživatel, kterému byl ukraden token. Po uložení dokumentu s nebezpečným popisem, se přihlásil uživatel do aplikace, se kterým byl dokument sdílen a tento dokument byl otevřen. Tento script ale nebyl vykonán díky obraně proti XSS. Podobné útoky lze provést i pomocí SQL injection. Aplikace je ale resistentní vůči tomuto útoku. Nebyly nalezeny žádné nedostatky.

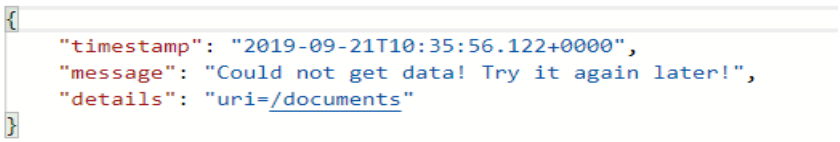
Testování HTTP Splitting/Smuggling (OTG-INPVAL-016)

Není umožněno žádné přesměrování v rámci aplikace. Nikde nezávisí cílové URL na uživatelské. Tento typ útoků není možný.

4.8 – Testování, jak aplikace pracuje s chybami (OTG-ERR)

Testování chyb (OTG-ERR-001)

Byly poslány žádosti na všechny end-pointy za účelem vrácení chyb 404, 400, 405, 408 a 501 serverem. Ve všech případech nebyla vrácena chybová hlášení, která by mohla útočníkovi něco prozradit o aplikaci. Bylo testováno, jaká chyba je vrácena v případě, když se server nemůže spojit s databází. Vrácenou chybu lze vidět na obrázku číslo 136. Server nevrátil informace o používaném frameworku nebo typu RDBMS.



Obrázek 136: OTG-ERR-001

Stejným způsobem byl otestován i webový server. V kapitole 3.4 lze vidět nastavenou vlastní HTML stránku v reakci na chyby. Nebyly nalezeny žádné nedostatky.

Testování Stack Traces (OTG-ERR-002)

Byly poslány žádosti na všechny end-pointy s neplatnými vstupy. Byl poslán vstup, který obsahuje speciální znaky, prázdné vstupy a příliš dlouhé řetězce. Server nikde neposlal odpověď obsahující Stack Traces. Nebyly nalezeny žádné nedostatky.

4.9 - Testování slabé kryptografie (OTG-CRYPST)

Testováních slabých SSL/TLS šifer, nedostatečná ochrana transportní vrstvy (OTG-CRYPST-001)

Bylo zkontrolováno, že přihlašovací údaje jsou přenášeny šifrovaně (test OTG-AUTHN-001).

Bylo otestováno kryptografické nastavení webového serveru IIS i REST serveru. K testování byl použit nástroj *testssl.sh*. Výsledek celého testu lze nalézt v souboru *ssl_test.txt*. Bylo prokázáno, že jsou zakázané protokoly nižší než TLS1.2, jsou povolené pouze silné šifry (tyto šifry lze nalézt v souboru *ssl_test.txt*), je používáno řazení šifer (viz *ssl_test.txt*). Velikost klíče certifikátu je RSA 2048 bitů a je používán silný podepisovací algoritmus SHA256 s RSA.

Test poukázal na potencionální nedostatek, který souvisí s používáním certifikátu samo sebou podepsaného. Použití tohoto certifikátu není pro aplikaci bezpečnostní hrozbou, protože je validní.

Testování, zda aplikace posílá citlivá data pouze přes nešifrované kanály (OTG-CRYPST-003)

Bylo otestováno, že není používán pro přenášení přihlašovacích údajů mechanismus Basic Authentization. Bylo prokázáno, že údaje přenášené z autentizačních formulářů se neodesílají přes HTTP. Bylo ověřeno, že přístupový token je přenášen v hlavičce Authorization: Bearer a vždy pomocí HTTPS.

4.10 - Testování business logiky (OTG-BUSLOGIC)

Testování validace logických business dat (OTG-BUSLOGIC-01)

Bylo otestováno, že pouze validní hodnoty jsou akceptovány aplikací. Byly ověřeno, že všechny proměnné odmítají logicky nevalidní data. Nebyly nalezeny žádné nedostatky.

Testování integrity dat (OTG-BUSLOGIC-003)

Musely být identifkovány části aplikace, které se starají o práci s daty. Bylo testováno, jak aplikace reaguje, pokud dojde k vložení, aktualizaci nebo smazání dat v každé identifikované komponentě uživatelem, který by neměl mít k tomuto úkonu právo. Nebyly nalezeny žádné nedostatky.

Testování časování procesu (OTG-BUSLOGIC-004)

Nebyly nalezeny žádné předvídatelné operace v této aplikaci.

Testování, zda funkce mohou být používané nad limit (OTG-BUSLOGIC-005)

Není nijak limitováno, kolikrát může daná funkce být vykonávána v čase.

Testování obcházení pracovního toku (OTG-BUSLOGIC-006)

Byl otestován chod transakce ukládání dokumentu. Byl vytvořen dokument, ale nebylo umožněno uložení informací o dokumentu do databáze. Tato situace vyvolala správně vrácení předchozích reakcí. Podobné situace byly otestovány i pro skupiny, uživatele a další operace s dokumenty. Nebyly nalezeny žádné nedostatky.

Testování obranných mechanismů proti nesprávnému použití aplikace (OTG-BUSLOGIC-007)

Byly testovány obranné mechanismy proti nesprávnému použití aplikace. Bylo otestováno, že vstup, který obsahoval zakázané znaky nebo parametry navíc, byl zamítnut serverem. Bylo otestováno, že aplikace vrací chybu, pokud dojde k záměně GET na POST a pokud je vložen znak ‘ místo validního ID dokumentu. Byl také duplikován parametr žádosti */users/finds?email=Maryška&email=Janš.* V tomto případě nebyla vrácena chyba, ale nebyl vrácen ani žádný výsledek. Bylo ověřeno, že dochází ke správnému zamknutí IP adresy po pěti neúspěšných pokusech o přihlášení viz test OTG-AUTHN-003**.** Nebyly nalezeny žádné nedostatky.

Testování nahrání neočekávaných souborů (OTG-BUSLOGIC-008)

Testem bylo nahrání nepodporovaného typu souboru a ujistit se, že není aplikací akceptovaný. Byl nahrán soubor typu EXE, který není podporován aplikací. Byla vrácena chyba 422 se zprávou *„File is not valid .pdf file“.* Nebyly nalezeny žádné nedostatky.

Testování nahrání souboru škodlivého souboru (OTG-BUSLOGIC-009)

Byla poslána žádost se souborem, který měl platnou koncovku (PDF), ale nejednalo se o PDF soubor. Byl odeslán soubor SVG s koncovkou PDF. Tato žádost byla odmítnuta serverem.

{"timestamp":"2019-09-20T11:44:15.055+0000","status":422,"error":"Unprocessable Entity","message":"File is not valid .pdf file.","path":"/documents"}

Obrázek 137: OTG-BUSLOGIC-009

Byl poslán soubor PDF, který obsahoval škodlivý javascriptový kód. V tomto bodě nedošlo k rozpoznání škodlivého PDF souboru a došlo k uložení na disk.

4.11 - Testování klientské strany (OTG-CLIENT)

Testování DOM based XSS (OTG-CLIENT-001)

Byla provedena kontrola kódu a nebyly nalezeny žádné možné útoky pomocí útoku DOM based XSS.

Testování javascript zranitelnosti (OTG-CLIENT-002)

Nebyly objeveny žádné takové zranitelnosti.

Testování HTML injection (OTG-CLIENT-003)

Bylo ověřeno, že není uživateli umožněno žádným způsobem zasahovat do HTML kódu aplikace.

Testování URL přesměrování na klientské straně (OTG-CLIENT-004)

Není umožněno přesměrování v žádné části aplikace. Nebyly nalezeny žádné nedostatky.

Testování CSS injection (OTG-CLIENT-005)

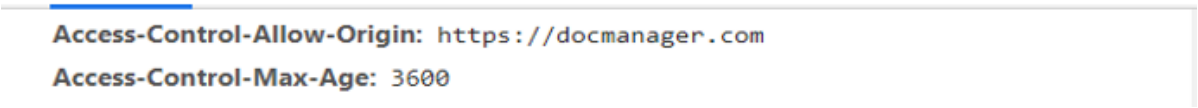
Bylo ověřeno, že není uživateli umožněno nijak zasahovat do CSS stylů.

Testování manipulace se zdroji na klientské straně (OTG-CLIENT-006)

Nebyly nalezeny žádné body, kde by mohlo dojít k tomuto typu útoku.

Testování CORS (OTG-CLIENT-007)

Bylo testováno, jakým způsobem je používáno CORS. Bylo zkoumáno, zda hlavička Access-Control-Allow-Origin není nastavena na hodntou *. Nastavení je v pořádku, protože pouze povolená adresa je *https://docmanager.com*.



Obrázek 138: OTG-CLIENT-007

Testování Clickjacking (OTG-CLIENT-009)

Bylo testováno, zda by aplikace mohla být nahrána na jiný web pomocí HTML elementu *iframe*. Byl vytvořen nový webový server s HTML kódem zobrazeným na obrázku 139 umístěným v body elementu HTML stránky. Bylo otestováno, že se testovaná stránka neobjeví v elementu *iframe* na nově vytvořené stránce. Stránka se neobjevila. Nebyly nalezeny žádné nedostatky.

```
<div class="container">
    <div class="content" id="main_content">
        <p>Website vulnereable to clickjacking</p>
        <iframe src="https://docmanager.com" width="500" height="500"></iframe>
</div>
```

Obrázek 139: OTG-CLIENT-009

Testování lokálního úložiště (OTG-CLIENT-012)

Na obrázku 140 lze vidět, že je používáno lokální úložiště k ukládání obnovovacího a přístupového tokenu.

Key	Value
access_token	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2...
refresh_token	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2...

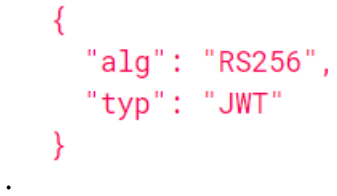
Obrázek 140: Obsah sessionStorage

Obě dvě proměnné jsou z hlediska bezpečnosti velmi kritické. Ukradení těchto údajů z úložiště může nastat pomocí útoku XSS, proti kterému je aplikace resistentní dle předchozích testů nebo tím, že se útočník dostane k těmto hodnotám tak, že se uživatel neodhlásí. Bylo otestováno, že po uzavření okna či browseru nejsou hodnoty tokenů uchovávány v úložišti. Nebyly nalezeny žádné nedostatky.

Dodatečné testy zaměřené na REST server byly testovány s pomocí OWASP REST Security Cheat Sheet [9]. Byly provedeny jen testy, které nebyly v dřívějších testech již prováděny.

JWT (REST-JWT)

Bylo otestováno, že JWT používá silný algoritmus a to RS256. Bylo ověřeno, že je používán podpis na JWT. Bylo ověřeno, že server odmítne zfalšovaný JWT token. Nebyly nalezeny žádné nedostatky.



Obrázek 141: Použitý algoritmus u tokenů

Validace content-type žádosti (REST-CONTENT_TYPE)

Bylo otestováno, že jsou vraceny na neočekávané žádosti (nepodporovaný, chybějící content-type hlavička) 406 nebo 415 chyby. Nebyly nalezeny žádné nedostatky.

.

Audit logy (REST-LOGS)

Je podporováno logování událostí definovaných v kapitole 2.2.10. Bylo zjištěno, že není podporováno logování neúspěšných validací přístupového tokenu.

Citlivé informace v HTTP žádostech (REST-SENSITIVE_DATA_HTTP)

Bylo ověřeno, že aplikace se nikde se nevyskytují hesla, přístupové tokeny a jiná citlivá data v URL. Nebyly nalezeny žádné nedostatky.

Vlastní testy

Životnost přístupového a obnovovacího tokenu (MY_Lifetime_AT_RT)

Bylo otestováno, že uživatel nemůže přistoupit k chráněným zdrojům po uplynutí životnosti daného přístupového tokenu. Bylo ověřeno, že není možné obnovit přístupový token po uplynutí životnosti obnovovacího tokenu. Nebyly nalezeny žádné nedostatky.

Odhlášení (MY _logout)

Bylo otestováno, že po kliknutí na tlačítko, které způsobí odhlášení uživatele, dojde ke

smazání lokálního úložiště a je poslána zpráva na server. Nebyly nalezeny žádné nedostatky.

4.12 Výsledky testů

Test OTG-AUTHN-007 prokázal nedostatky. Síla hesla je naprosto dostatečná, ale celková politika hesla není optimální. Měla by být implementována změna hesla v nějakých časových intervalech. Příkladem může být nucená změna hesla každé tři měsíce, čímž by se síla politiky hesla zvýšila. V aplikaci je sice změna hesla implementována, ale nucená není. Je kontrolováno při změně hesla, zda uživatelovo nově chtěné heslo není totožné s aktuálním heslem, ale není udržována historie hesel, takže si uživatel může po poslání několika žádostí za sebou opět nastavit své staré heslo.

Test OTG-INFO-008 odhalil, jaké technologie jsou na frontendu používané. Po prohlídnutí kódu bylo identifikováno, že se jedná o framework jQuery verze 3.4.1. Tento nedostatek je neřešitelný, protože soubor obsahující jQuery je veřejně přístupný.

Test OTG-BUSLOGIC-009 prokázal nedostatky při nahrávání souborů. První část testu proběhla úspěšně. Byl odeslán soubor typu SVG s koncovkou PDF, který byl odmítnut serverem. Druhá část testu se týkala nahrání souboru typu PDF, který obsahoval škodlivý javascriptový kód. Server tento soubor přijal a uložil na disk. Tento problém není chybou aplikace, ale spíše chybou antivirového programu, který nebyl schopen rozeznat škodlivý soubor typu PDF. Řešením tohoto problému by bylo použití lepšího antivirového programu.

Test, který také prokázal nedostatky je REST-LOGS. Aplikace nemá implementovanou funkcionalitu logování neúspěšných validací přístupového tokenu.

Všechny ostatní provedené testy neprokázaly žádné jiné nedostatky aplikace.

5. Závěr

Byla úspěšně implementována aplikace pro řízení vnitropodnikové dokumentace dle popisu v kapitole 1.1. Byly úspěšně implementovány různé bezpečnostní mechanismy na serveru i frontendu. Důležitou součástí procesu zabezpečení aplikace byla bezpečná konfigurace webového serveru IIS.

Framework Spring Boot se prokázal jako velmi dobrý nástroj při implementaci bezpečnostních mechanismů. Spring Boot má některé bezpečnostní mechanismy přednastavené, například základní bezpečnostní hlavičky. Pomocí anotací lze velmi snadno nastavovat různé bezpečnostní mechanismy a kontrolovat validitu příchozích dat. Velmi snadná je přeměna dat ve formátu JSON na Java objekt. Tato transformace je prováděna frameworkem automaticky.

Dle provedených penetračních testů se jeví, že je aplikace dobře zabezpečená. Dle dokumentu OWASP Top 10 z roku 2017 je nejvíce kritickým bezpečnostním rizikem útok injection [6]. Aplikace je vůči tomuto typu útoku dle provedených testů resistentní. Druhým největším rizikem je prolomení autentizačního schématu. Provedené testy z kapitoly OTG-AUTHN neodhalily téměř žádné nedostatky. Velmi častým bezpečnostním rizikem je obejití autorizačního schématu. Provedené testy z kategorie OTG-AUTHZ také neodhalily žádné nedostatky.

Penetrační testy prokázaly, že aplikace obsahuje také nějaké bezpečnostní nedostatky. Pomocí testů byly nalezeny nedostatky jako uložení souboru typu PDF obsahujícího škodlivý kód na disk, chybějící logování neúspěšné validace tokenů a slabá politika hesla. První dva zmíněné nedostatky jsou velmi snadno řešitelné. Nedostatek týkající se hesla je z hlediska časové implementace nejnáročnější. Musela by být navržena nová databázová struktura, která by řešila historii hesel a musela by být implementována funkcionalita nucené změny hesla v nějakých časových intervalech.

Autor je přesvědčen, že aplikace je použitelná v praxi za předpokladu, že by byla implementována paginace pro vyhledávané dokumenty a uživatele. Podpora více typů souborů by mohla být velmi užitečná. Bylo by rozumné použít silnější politiku hesla a nasadit vhodnější antivirový program.

6. Seznam použité literatury

[1] Number of global e-commerce transactions 2015 | Statista. • *Statista - The Statistics Portal for Market Data, Market Research and Market Studies* [online]. Copyright © Statista 2019 [cit. 11.09.2019]. Dostupné z: https://www.statista.com/statistics/369333/number-ecommerce-transactions-worldwide/

[2] Transport Layer Protection · OWASP Cheat Sheet Series. *Introduction · OWASP Cheat Sheet Series* [online] [cit. 05.10.2019]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

[3] SULLIVAN, Bryan a Vincent LIU. *Web application security: a beginner's guide*. 2012. New York: McGraw-Hill, c2012. ISBN 978-0-07-177616-5.

[4] X-XSS-Protection - HTTP | MDN. [online]. Copyright © 2005 [cit. 03.10.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

[5] Content Security Policy (CSP) - HTTP | MDN. [online]. Copyright © 2005 [cit. 03.10.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[6] Category:OWASP Top Ten Project - OWASP. [online] [cit. 05.10.2019]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[7] Cross-Origin Resource Sharing (CORS) - HTTP | MDN. [online]. Copyright © 2005 [cit. 03.10.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[8] OWASP Testing Project - OWASP. [online] [cit. 05.10.2019]. Dostupné z: https://www.owasp.org/index.php/OWASP_Testing_Project

[9] REST Security · OWASP Cheat Sheet Series. *Introduction · OWASP Cheat Sheet Series* [online] [cit. 06.10.2019]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

[10] JSON Web Token Introduction - jwt.io. *JSON Web Tokens - jwt.io* [online] [cit. 07.10.2019]. Dostupné z: https://jwt.io/introduction/

[11] X-Content-Type-Options :: informace, nastavení | SecurityHeaders.cz. *Security Headers | Informace, návody a nastavení HTTP bezpečnostních hlaviček* [online]. Copyright © 2018 Web security s.r.o. [cit. 04.10.2019]. Dostupné z: https://securityheaders.cz/x-content-type-options

[12] X-Frame-Options :: informace, nastavení | SecurityHeaders.cz. *Security Headers | Informace, návody a nastavení HTTP bezpečnostních hlaviček* [online]. Copyright © 2018 Web security s.r.o. [cit. 07.10.2019]. Dostupné z: https://securityheaders.cz/x-frame-options

[13] Cache-Control – HTTP | MDN. [online]. Copyright © 2005 [cit. 07.10.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

[14] Pragma – HTTP | MDN. [online]. Copyright © 2005 [cit. 07.10.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Pragma

[15] HTTP Strict Transport Security :: informace, nastavení | SecurityHeaders.cz. *Security Headers | Informace, návody a nastavení HTTP bezpečnostních hlaviček* [online]. Copyright © 2018 Web security s.r.o. [cit. 07.10.2019]. Dostupné z: https://securityheaders.cz/hsts

[16] Hardt, D. (2012). *RFC 6749 - The OAuth 2.0 Authorization Framework*. [online] Tools.ietf.org [cit. 07.10.2019]. Dostupné z: https://tools.ietf.org/html/rfc6749 [cit. 09.10.2019].

[17] BCryptPasswordEncoder (Spring Security 4.2.12.RELEASE API) [online] [cit. 10.10.2019]. Dostupné z: https://docs.spring.io/spring-security/site/docs/4.2.12.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html

[18] 53. Endpoints [online] [cit. 10.10.2019]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html

[19] 21. Security HTTP Response Headers [online] [cit. 10.10.2019]. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.0.x/reference/html/headers.html

[20] Spring Boot Reference Guide [online] [cit 11.10.2019]. Copyright © 2012 [cit. 02.10.2019]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[21] CIS Microsoft IIS 10 Benchmark [online]. Center for Internet Security, 2019 [cit. 2019-18-10]. Dostupné z: https://learn.cisecurity.org/benchmarks?fbclid=IwAR0cf7uyvZDC0nTh6HzvPlBVY_Vkt3fGN0Cz_GYD2PECmXL69IqZmfOTwNk

[22] ImperialViolet – Lucky Thirteen attack on TLS CBC. *ImperialViolet – ImperialViolet.org* [online] [cit. 11.10.2019]. Dostupné z: https://www.imperialviolet.org/2013/02/04/luckythirteen.html

[23] SAML2 vs JWT: Understanding OAuth2 - Robert Broeckelmann - Medium. *Medium – Get smarter about what matters to you.* [online] [cit. 30.10.2019]. Dostupné z: https://medium.com/@robert.broeckelmann/saml2-vs-jwt-understanding-oauth2-4abde9e7ec8b

7. Přílohy

I. Obsah DVD

- *bp.pdf* – text bakalářské práce
- složka *penetrační_testy* obsahuje výsledky penetračních testů. Nachází se zde složka *sql_injection*, která obsahuje testy týkající se testu OTG-INPVAL-005. Soubor *xss_payload.txt* obsahuje řetězce použité k testování testu OTG-INPVAL-001 a OTG-INPVAL-002. Soubor *ssl-test.txt* obsahuje výsledky testů OTG-CRYPST-001 a OTG-CRYPST-002. Soubor *penetrační_testy.docx* obsahuje popis provedených testů a nerelevantních testů.
- složka *backend* obsahuje zdrojový kód REST serveru (Intellij IDEA project)
- složka *pre_final* obsahuje zdrojový kód scriptu *bundle.js*
- složka *web_server* obsahuje všechny soubory používané na webovém serveru IIS a tedy i na frontendu (HTML, CSS, JS, web.config, obrázky)

II. Seznam obrázků

Obrázek 1: UML use case diagram 2
Obrázek 2: Relační model databáze 4
Obrázek 3: Používané role, stavy a položky v tabulce sharing_type 4
Obrázek 4 Architektura aplikace 6
Obrázek 5: Certifikát serveru 7
Obrázek 6: Uložené heslo v databázi 8
Obrázek 7: Password Credentials Flow [23] 9
Obrázek 8: Zakázané metody na webovém serveru IIS 12
Obrázek 9: Hlavičky REST serveru 13
Obrázek 10: Používané balíčky na REST serveru 16
Obrázek 11: Frontendová složka 17
Obrázek 12: Script bundle.js 18
Obrázek 13: Testování validity tokenu 19
Obrázek 14: Dekódování tokenu 19
Obrázek 15: JPQL dotaz pro získání všech sdílených dokumentů s uživatelem 20
Obrázek 16: Metoda compare 20

Obrázek 17: Použití komparátoru 20
Obrázek 18: Načtený dokument k odsouhlasení 21
Obrázek 19: Generování PDF dokumentu 21
Obrázek 20: Poslání PDF dokumentu 22
Obrázek 21: Zpracování PDF souboru 22
Obrázek 22: End-point PUT /documents 23
Obrázek 23: Schvalování dokumentu 1 23
Obrázek 24: Schvalování dokumentu 2 24
Obrázek 25: Nahrání dokumentu 24
Obrázek 26: End-point /documents POST 24
Obrázek 27: Metoda write 25
Obrázek 28: Vytvořené složky aplikací 25
Obrázek 29: Metoda encryptFile 26
Obrázek 30: Uložení dokumentu do databáze 26
Obrázek 31: Poslaný email aplikací o novém dokumentu 26
Obrázek 32: JPQL dotaz pro vyhledání dokumentu podle titulu 27
Obrázek 33: Vyhledávání podle měsíce a roku 27
Obrázek 34: JPQL dotaz pro získání sdílených dokumentů v určitém období 28
Obrázek 35: Vyhledávání vlastních dokumentů 28
Obrázek 36: JPQL dotaz pro získání uživatelových dokumentů 28
Obrázek 37: Výsledky schvalování 29
Obrázek 38: DELETE tlačítko u dokumentu 30
Obrázek 39: Smazání dokumentu 30
Obrázek 40: Blokace expirovaných dokumentů 31
Obrázek 41: JPQL dotaz pro získání expirovaných dokumentů 31
Obrázek 42: Rozeslání emailů o nových dostupných dokumentech 31
Obrázek 43: JPQL dotaz pro získání nových dokumentů 32
Obrázek 44: Poslání emailu a získání objektu typu JavaMailSender 32
Obrázek 45: Konfigurace emailové komunikace pro Gmail SMTP Server 32
Obrázek 46: Registrace klienta 33
Obrázek 47: Aktualizace uživatele 33
Obrázek 48: Hledání uživatele podle jména 34
Obrázek 49: JPQL dotaz pro získání uživatelů podle příjmení 34
Obrázek 50: Update a Deactivate tlačítka 34

Obrázek 51: Deaktivace uživatele 35
Obrázek 52: Výsledek vyhledávání 35
Obrázek 53: Uložení skupiny 35
Obrázek 54: Aktualizace skupiny 36
Obrázek 55: Smazání skupiny 36
Obrázek 56: Odhlášení 37
Obrázek 57: Autorizační server 37
Obrázek 58: TokenStore u autorizačního serveru 38
Obrázek 59: Asymetrické podepsání JWT na straně autorizačního serveru 38
Obrázek 60: Konfigurace end-pointů autorizačního serveru 38
Obrázek 61: Ověření podpisu 39
Obrázek 62: JwtTokenStore a metoda configure 39
Obrázek 63: HTTP bezpečnost 40
Obrázek 64: Anotace třídy SecurityConfig 40
Obrázek 65: Metoda globalUserDetails 40
Obrázek 66: Vracení objektu typu User 41
Obrázek 67: Nastavení HTTPS 41
Obrázek 68: Autentizační žádost 41
Obrázek 69: Obdržená data po úspěšné autentizaci 42
Obrázek 70: Kontrola tokenu 43
Obrázek 71: Chráněný zdroj anotací @Secured 43
Obrázek 72: Bean encoder 44
Obrázek 73: Vytvoření uživatele 44
Obrázek 74: Požadavky na heslo 44
Obrázek 75: Autentizace uživatele 44
Obrázek 76: Změna hesla 45
Obrázek 77: Globální reakce na výjimku 45
Obrázek 78: ResponseStatusException 45
Obrázek 79: Nastavení CORS 46
Obrázek 80: Vypnutí actuator end-pointů 46
Obrázek 81: Vytvoření instance třídy Logger 46
Obrázek 82: Zápis informace do logu 47
Obrázek 83: Globální logovací soubor 47
Obrázek 84: Zašifrování a uložení dokumentu 47

Obrázek 85: Dešifrovací metoda 47
Obrázek 86: Zakázané metody 48
Obrázek 87: Validace jména 48
Obrázek 88: Validace hesla pomocí vlastní anotace 48
Obrázek 89: Validace roku a měsíce 48
Obrázek 90: Parametr long 49
Obrázek 91: Validace datumu 49
Obrázek 92: Validace typu dokumentu 49
Obrázek 93: Validace titulu 50
Obrázek 94: Parametrizované dotazy 50
Obrázek 95: Metoda loginFailed 51
Obrázek 96: Kontrola blokace IP adresy 51
Obrázek 97: Autentizační žádost 51
Obrázek 98: Běžná žádost na server 52
Obrázek 99: Zachycení chyby 401 52
Obrázek 100: Žádost o nový přístupový token 53
Obrázek 101: Chyba 401 53
Obrázek 102: Odhlášení 54
Obrázek 103: Metoda inHTMLData 54
Obrázek 104: Metoda inDoubleQuotedAttr 54
Obrázek 105: Vypnutí atributu autocomplete 54
Obrázek 106: Kontrola dat pomocí regulárního výrazu 55
Obrázek 107: Zakázané directory browsing 55
Obrázek 108: Nastavení šifrování 56
Obrázek 109: Hodnota maxURL 56
Obrázek 110: Konfigurace hlavičky server [21] 57
Obrázek 111: Bezpečnostní hlavičky na webovém serveru IIS 57
Obrázek 112: Reakce na chyby – webový server 58
Obrázek 113: Test OTG-INFO-002, první žádost 58
Obrázek 114: Test OTG-INFO-002, druhá žádost 59
Obrázek 115: OTG-INFO-008 60
Obrázek 116: Otisk webové aplikace 60
Obrázek 117: OPTIONS – webový server IIS 61
Obrázek 118: OTG-CONFIG-006, druhá část 62

Obrázek 119: OTG-CONFIG-007 62
Obrázek 120: OTG-AUTHN-001 64
Obrázek 121: OTG-AUTHN-003 64
Obrázek 122: OTG-AUTHN-004 65
Obrázek 123: Potencionálně nebezpečné přepínání stránek 66
Obrázek 124: OTG-AUTHZ-001 66
Obrázek 125: OTG-AUTHZ-002, první část 66
Obrázek 126: OTG-AUTHZ-002, druhá část 67
Obrázek 127: OTG-AUTHZ-002, třetí část 67
Obrázek 128: OTG-AUTHZ-002, čtvrtá část 67
Obrázek 129: OTG-AUTHZ-003 68
Obrázek 130: OTG-AUTHZ-004, první část 68
Obrázek 131: OTG-AUTHZ-004, druhá část 68
Obrázek 132: OTG-INPVAL-005 70
Obrázek 133: sqlmap příkaz 70
Obrázek 134: Velikosti výsledných souborů s žádostmi 71
Obrázek 135: Útočný script 72
Obrázek 136: OTG-ERR-001 72
Obrázek 137: OTG-BUSLOGIC-009 75
Obrázek 138: OTG-CLIENT-007 76
Obrázek 139: OTG-CLIENT-009 76
Obrázek 140: Obsah sessionStorage 76
Obrázek 141: Použitý algoritmus u tokenů 77