Aplikace CertiChain – využití technologie blockchain pro podepisování dat

Bc. Jakub Maňák

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jakub Maňák**
Osobní číslo:	**A20189**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Aplikace CertiChain – využití technologie blockchain pro podepisování dat**
Téma práce anglicky:	**CertiChain Application – Using Blockchain Technology for Data Signing**

Zásady pro vypracování

1. Nastudujte problematiku spojenou s digitálními podpisy a certifikáty.
2. Popište problematiku technologie blockchain.
3. Vyberte vhodné technologie a prostředky pro implementaci vlastního řešení.
4. Implementujte vlastní webovou aplikaci CertiChain.
5. Výslednou aplikaci vhodně otestujte.
6. Vhodným způsobem reprezentujte výsledky.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ČESKÁ REPUBLIKA. Zákon o službách vytvářejících důvěru pro elektronické transakce. In: Sbírka zákonů. Praha: Tiskárna Ministerstva vnitra, 2016, ročník 16, částka 115, číslo 297. Dostupné také z: https://www.zakonyprolidi.cz/cs/2016-297 – tohle je 297/2016, adaptační zákon na nařízení eidas
2. Zákon č. 227/2000 Sb., o elektronickém podpisu. Ministerstvo vnitra České republiky [online]. 2012, 1. 10. 2012 [cit. 2021-11-19]. Dostupné z: https://www.mvcr.cz/clanek/zakon-c-227-2000-sb-o-elektronickem-podpisu.aspx – 227/2000 Sb.
3. ČESKÁ REPUBLIKA. Nařízení Evropského parlamentu a Rady (EU) č. 910/2014 ze dne 23. července 2014 o elektronické identifikaci a službách vytvářejících důvěru pro elektronické transakce na vnitřním trhu a o zrušení směrnice 1999/93/ES. In: Https://eur-lex.europa.eu. EU, 2014, ročník 2014, částka 914, číslo 914. Dostupné také z: https://eur-lex.europa.eu/legal-content/CS/ALL/?uri=celex:32014R0910

Vedoucí diplomové práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním DP souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že DP bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk DP bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování DP využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky DP využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem DP jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze DP a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.5.2022

Jakub Maňák, v. r.
……
podpis studenta

ABSTRAKT

DP se zabývá problematikou digitálních podpisů v kombinaci s použitím technologie Blockchain. V teoretické části je rozepsána problematika digitálních podpisů, certifikátů a technologie blockchain. V praktické části je nejprve navržena vlastní aplikace CertiChain na vytváření a ověřování digitálních podpisů, kdy při procesu vytváření a ověřování je využíván i systém blockchain. Praktická část se dále zabývá shrnutím možných technologií pro implementaci vlastní aplikace a pomocí vybraných technologií je poté aplikace i naimplementována. Po dokončení implementace je aplikace otestována a na závěr nasazena pro použití.

Klíčová slova: Blockchain, digitální podpis, certifikát, webová aplikace, C#, Asp.Net, Blazor, Java, Spring, MySQL, ProvenDB

ABSTRACT

The diploma thesis deals with the issue of digital signatures in combination with the use of Blockchain technology. The theoretical part describes issues of digital signatures, certificates, and blockchain technology. In the practical part, the CertiChain application is first designed for the creation and verification of digital signatures, where the blockchain system is also used when creating and verifying digital signatures. The practical part also deals with a summary of possible technologies for the implementation of their own application and with the help of selected technologies the application is then implemented. After the implementation is completed, the application is tested, and finally deployed for use.

Keywords: Blockchain, digital signature, certificate, web application, C#, Asp.Net, Blazor, Spring, MySQL, ProvenDB

V této části bych chtěl poděkovat vedoucímu mé DP, panu Ing. Petru Žáčkovi, Ph.D. za jeho ochotu, rady a veškerou pomoc při vytváření této DP.

Prohlašuji, že odevzdaná verze DP a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době je postupný trend digitalizace dokumentů a s tímto trendem jde ruku v ruce také vytváření digitálních podpisů, které lze v dnešní době využít i pro podepisování dokumentů při komunikaci se státními orgány místo svého vlastnoručního podpisu.

Další technologií, která v dnešní době získává na popularitě jsou systémy Blockchain, což je specializovaný typ decentralizované a distribuované databáze, která díky svým principům fungování může zaručit, že s vloženými daty nebude nijak manipulováno. Vložená data lze poté v blockchain ověřovat po neurčitou dobu nebo přesněji po dobu života daného blockchain systému, která se díky decentralizovaném a distribuovaném principu fungování může rovnat i době neurčité.

Vybízí se otázka, proč jsou zde tahle dvě témata uvedena a jak je lze společně využít. Digitální podpis je efektivní cestou, jak nahradit podpis vlastnoruční. Hlavním problém digitálních podpisů je doba platnosti, která nelze protáhnout na dobu neurčitou. Problém s platností podpisu by bylo možno vyřešit vložením digitálně podepsaných dat do systému blockchain. A právě to si klade za cíl tato práce – skloubit vytváření tradičních digitálních podpisů pomocí certifikátů a pomocí uložení záznamu o podepsání do systému blockchain a naimplementovat vlastní aplikaci pro vytváření digitálních podpisů a jejich ukládání do blockchain.

Práce je rozdělena do osmi kapitol. První tři kapitoly DP jsou zaměřeny na teoretickou část DP. První z nich se zabývá problematikou vytváření digitálních podpisů pomocí certifikátů. Problematika je rozebrána z hlediska platných zákonů v ČR i z hlediska technických norem. Tato kapitola čerpá ze zákonů č. 227/2000 Sb. a evropského nařízení eIDAS z technických norem pro vytváření podpisů CAdES, PAdES a XAdES. Druhá kapitola rozebírá problematiku blockchain. Je rozebrán základní princip fungování blockchain, princip přidávání záznamů do systémů, způsob distribuce systémů blockchain a ve kterých oblastech jsou využívané. Rešerše o blockchainu čerpá zejména z odborných článků dostupných v online databázích.

Třetí kapitola je zaměřena na souhrn technologií, pomocí kterých lze aplikaci naimplementovat. Ve čtvrté kapitole je navrhována vlastní aplikace, která bude umožnovat vytvářet a ověřovat digitální podpisy a záznamy o podepsání dat bude vkládat do blockchain a také je bude moci pomocí blockchain ověřovat. V páté kapitole jsou z dostupných technologií vybrány technologie pro samotnou implementaci aplikace a další technologie kritické pro fungování aplikace.

Šestá kapitola se zabývá samotným procesem tvorby aplikace. V kapitole je popsán postup od samotného vytvoření a konfigurace projektů, přes implementaci jednotlivých funkcí, které jsou uvedeny v kapitole o návrhu aplikace. V sedmé kapitole je sepsán postup testování naimplementované aplikace pomocí tvorby automatických testů a manuálního testování.

Závěrečná osmá kapitola se zabývá výslednou prezentací aplikace, kdy nejdříve popisuje postup nasazení aplikace na vlastní „server“, a její zveřejnění, aby byla dostupná z internetu. Po nasazení aplikace je popsáno samotné použití aplikace, kde je ukázáno, jak jsou naimplementovány jednotlivé funkce aplikace, které jsou definovány v analýze aplikace.

I. TEORETICKÁ ČÁST

1 DIGITÁLNÍ PODEPISOVÁNÍ

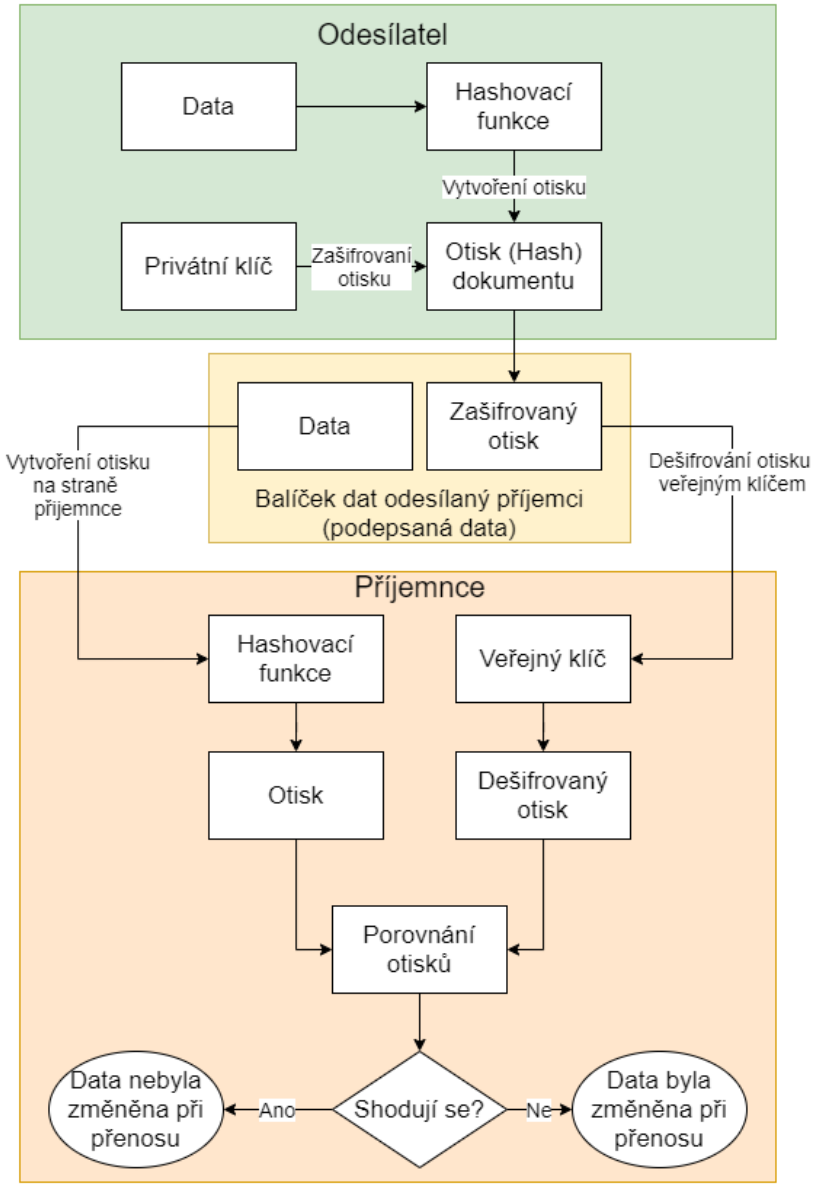
Počátky digitálního podepisování začínají v roce 1991, kdy americký institut NIST (národní institut standartu a technologií) představil algoritmus DSA (Digital Signature Algorithm). Tento algoritmus si nechal patentovat pán David W. Kravitz.

1.1 Princip DSA algoritmu

Princip DSA algoritmu je založen na klíčovém páru, který se skládá z primárního klíče, který by si vlastník měl pečlivě uschovat a dále z klíče veřejného, který předá daným subjektům (uživatelům).

Když chce uživatel vytvořit digitální podpis, nejprve si vybere dokument nebo data v libovolném formátu. Z těchto dat si vytvoří jejich otisk pomocí hashovacího algoritmu, které se vyznačují tím, že jsou jednocestné. To že je algoritmus jednocestný znamená, že je možno z původních dat získat otisk, ale není možné z otisku získat originální data. Další typickou vlastností jednocestných algoritmů je, že z dat o libovolné délce vytvoří otisk o pevně dané délce (záleží na typu hashovacího algoritmu).

Vytvořený otisk se poté zašifruje pomocí privátního klíče. Vzniklý podpis se přidá k původním datům a rozesílá se cílovým uživatelům, kteří mají veřejný klíč. Ti ověří správnost podepsaného dokumentu tak, že dešifrují zašifrovaný otisk dokumentu pomocí veřejného klíče, který dostali od majitele párů klíče. Poté si z poslaných dat vypočtou otisk dokumentu pomocí stejného hashovacího algoritmu, jako jej vytvořil podepisující uživatel a ověří jej s dešifrovaným otiskem. Pokud jsou otisky stejné, data nebyla během přenosu změněna a máme jistotu, že data byla odeslána majitelem privátního klíče.



Obr. 1 Schéma algoritmu DSA (vlastní)

1.2 Elektronické podpisy v České republice

Postupem času po zveřejnění algoritmu DSA přicházela myšlenka elektronického (digitálního) podepisování, tedy možnost nahrazení vlastnoručního podpisu digitální podobou. V roce 1999 vydala Evropská unie (EU) směrnici 1999/93/EC, díky které byly definovány různé pojmy a úrovně elektronických podpisů. Česká republika se touto směrnicí také inspirovala (i přesto, že v té době ještě nebyla součástí EU) a vytvořila zákon č. 227/2000 Sb., který umožnil používat elektronické podpisy v ČR ke komunikaci se státní správou.

1.2.1 Zákon č. 227/2000 Sb.

Jak bylo zmíněno na úvodu kapitoly 1.2, zákon o elektronickém podpisu v ČR byl poprvé představen v roce 2000 a jeho označení je zákon č. 227/2000 Sb. o elektronickém podpisu. Hlavním účelem tohoto zákona bylo umožnit používat digitální podpis v rámci elektronické komunikace, a aby digitální podpis měl stejnou úroveň jako podpis vlastnoruční. [1]

Zákon 227/2000 prošel od doby vydání několika novelizacemi. První novelizace nabyla účinnost 26. července 2004 v zákoně č. 440/2004 Sb. Tato novelizace zavedla pojem kvalifikovaného časového razítka a dále pojem elektronické značky. Druhá novelizace nabyla účinnosti 15. dubna 2010 v zákoně č. 101/2010 Sb. V zákoně přibyla povinnost pro Ministerstvo vnitra vést si a zveřejňovat seznam důvěryhodných certifikačních služeb a dále stanovuje povinnost uznávat kvalifikované certifikáty vydané v členských zemích EU. [1]

Po všech novelizacích se zákon č. 227/2000 Sb. skládá z 20 paragrafů, které definují jednotlivé pojmy, jejich pravomoci, omezení pro použití digitálních podpisů. V následující podkapitolách budou jednotlivé pojmy detailněji rozepsány.

1.2.1.1 Poskytovatel certifikačních služeb

K vytvoření digitálního podpisu je zapotřebí tzv. digitální certifikát, který je potřeba získat u nějaké autority. V zákoně č. 227/2000 Sb. tato autorita nese název Poskytovatel certifikačních služeb, obecně jsou nazývány certifikační autority.

Poskytovatelem certifikačních služeb může být i fyzická, ale typicky spíše právnická osoba, která tento status může získat od Ministerstva vnitra tím, že mu udělí akreditaci na poskytování certifikačních služeb. Poskytovatel certifikačních služeb po udělení akreditace může veřejnosti vydávat certifikáty a dále může poskytovat služby (např. aplikace) spojené digitálním podepisováním. Poskytovatel certifikačních služeb takhle zastřešuje celý proces tvorby digitálních podpisů. [1]

Dále ještě zákon č. 227/2000 Sb. hovoří o kvalifikovaném poskytovateli certifikačních služeb, který může navíc vydávat kvalifikované certifikáty. [1]

V České republice jsou ke 22.3.2022 tři kvalifikovaní poskytovatelé certifikačních služeb – První certifikační autorita, a. s., Česká pošta, s. p. a eIdentity, a. s. [7]

1.2.1.2 Digitální certifikát

V minulém odstavci bylo zmíněno, že pro vytvoření digitálního podpisu je zapotřebí mít digitální certifikát. V zákoně č 227/2000 Sb. je sepsáno, co musí daný certifikát obsahovat a také definuje různé varianty certifikátů.

Certifikát žadateli může vydat Poskytovatel certifikačních služeb, poté co žadatel prokáže u poskytovatele certifikačních služeb svou identitu (typicky pomocí občanského průkazu, a ještě jednoho dalšího průkazu, např. pas nebo řidičský průkaz). Poskytovatel vygeneruje klíčový pár (soukromý a veřejný klíč), kdy veřejný klíč je elektronicky podepsán certifikátem poskytovatele certifikačních služeb. [1]

Dále certifikát obsahuje informace o subjektu (žadateli). Jedná se o informace jako jméno a příjmení. Dále jsou v certifikátu udány informace o vydavateli certifikátu (poskytovatel certifikačních služeb), datum zahájení platnosti certifikátu a datum ukončení platnosti certifikátu, doba platnosti certifikátu (typicky 365 dní). [1]

Existuje také více typů digitálních certifikátů, které lze získat. Jedná se o komerční certifikát či kvalifikovaný certifikát. Rozdíl mezi těmito certifikáty je v úrovni autentizace žadatele u poskytovatele certifikačních služeb a od toho se také odvíjí jaký typ elektronického podpisu s daným certifikátem lze vytvořit. Kvalifikovaný certifikát v sobě navíc obsahuje identifikátor, který zajišťuje jednoznačnou identifikaci uživatele. Dále ještě zákon č. 227/2000 Sb. hovoří o systémovém certifikátu, který je určen primárně pro právnické osoby a slouží ke tvorbě elektronických značek, které jsou rozebrány v kapitole 1.2.1.4. [1]

1.2.1.3 Elektronický podpis

Elektronický podpis zaručuje, že dokument nebo data podepsala osoba, která je uvedená v digitálním certifikátu. Elektronický podpis lze vytvořit pomocí komerčního anebo kvalifikovaného certifikátu a zákon č. 227/2000 Sb. definuje dvě úrovně elektronického podpisu. Elektronický podpis lze označit jako elektronická náhrada vlastnoručního podpisu. [1]

Elektronický podpis má nižší úroveň. Elektronický podpis se vytváří pomocí komerčního certifikátu a lze používat např. pro podepisování dokumentů uvnitř společnosti, pokud se daná společnost dohodne, že ji tato úroveň podpisu dostačuje. [1]

Dále existuje zaručený elektronický podpis, který je vytvořen kvalifikovaným certifikátem. Tento podpis má vysokou právní úroveň a lze použít pro komunikaci se státní správou.

Právní úroveň kvalifikovaného elektronického podpisu se udává v rozmezí mezi vlastnoručním podpisem a úředně ověřeným vlastnoručním podpisem. [1]

1.2.1.4 Elektronická značka

Elektronická značka je pojem, který přišel s novelizací zákona č. 227/2000 Sb. Tato novelizace vyšla v roce 2004. Elektronická značka je obdoba elektronického podpisu s tím rozdílem, že je vytvářena systémovým certifikátem. Elektronická značka je tudíž primárně určena pro podepisování dokumentů právnickou osobou. Podobně jako u elektronického podpisu jsou k dispozici dva typy elektronických značek. [1]

První je klasická elektronická značka, která má nižší právní úroveň. Elektronickou značku lze vytvořit pomocí systémového certifikátu. Druhým typem je uznávaná elektronická značka, která je vytvořena pomocí kvalifikovaného systémového certifikátu. [1]

1.2.1.5 Časová razítka

Pojem časové razítko přišel také s první novelizací zákona č. 227/2000 Sb. Časové razítko se často přidává k elektronickým podpisům jako stvrzení, že podpis již existoval v daném čase, kdy bylo časové razítko vytvořeno. Časové razítko v sobě uchovává otisk dat, která chceme opatřit časovým razítkem a dále zaručený (velmi přesný) čas, kdy bylo časové razítko vytvořeno. Časové razítko může být vydáno poskytovatelem certifikačních služeb nebo lze získat ještě kvalifikované časové razítko, od kvalifikovaných poskytovatelů certifikačních služeb. [1]

1.2.2 Nařízení eIDAS

Čas od vydání zákona č. 227/2000 Sb. plynul a ČR se stala součástí EU. Aby bylo možno akceptovat elektronické podpisy i z ostatních zemí EU, ČR vydala novelizaci zákona č. 227/2000 Sb., která tuhle akceptaci umožnila. EU ale postupem let chtěla sjednotit formu elektronických podpisů v celé EU, a proto v roce 2016 přišla s nařízením, které se ve zkratce označuje jako nařízení eIDAS. Vzhledem k tomu, že se nejedná o směrnici, ale o nařízení, musí jej všechny státy EU používat a přizpůsobit se mu.

S příchodem nařízení eIDAS nastala v ČR na pár měsíců situace, kdy platily dva zákony o elektronickém podepisování najednou. Vláda ČR totiž nestihla připravit tzv. adaptační zákon do dne zahájení platnosti nařízení eIDAS, který měl zrušit platnost dosavadního zákona č. 227/2000 Sb. Tudíž šlo využívat obou zákonů.

Další komplikace s nařízením eIDAS se týká jednotlivých pojmů, které zákon č. 227/2000 Sb. a nařízení eIDAS definují. Zákon č. 227/2000 Sb. vycházel z evropské směrnice, kterou nařízení eIDAS aktualizuje, ale u některých pojmů nastala situace, kdy mají stejný název v obou zákonech, ale mají odlišnou definicí anebo naopak mají dva pojmy trochu odlišný název, ale stejný význam.

V následujících podkapitolách budou jednotlivé pojmy z nařízení eIDAS přirovnány k pojmům ze zákona č. 227/2000 Sb.

1.2.2.1 Kvalifikované služby vytvářející důvěru

Tento zdlouhavý pojem lze přirovnat k poskytovateli certifikačních služeb. Kvalifikované služby vytvářející důvěru musí získat povolení od dohledového orgánu, aby mohly poskytovat služby pro vydávání certifikátu a pro služby s tím spojené (např. aplikace pro podepisování souborů). V nařízení eIDAS se ale navíc poskytovatelé musí podrobovat auditům, které se konají 1x za 24 měsíců. [2]

Nově nařízení eIDAS vytvořilo jednotnou značku důvěry EU, kterou můžou kvalifikovaní poskytovatelé umísťovat ke svým produktům a službám. Tato značka má za cíl ulehčit uživatelům zjistit, zda daný poskytovatel a služba splňuje nařízení eIDAS.

Obr. 2 značka důvěry EU [31]

1.2.2.2 Digitální certifikát

Nařízení eIDAS obsahuje také termíny komerční a kvalifikovaný certifikát. Nařízení eIDAS ale klade jiné bezpečnostní požadavky na kvalifikovaný certifikát. Zde neobsahuje jednoznačný identifikátor držitele certifikátu jako v zákoně č. 227/2000 Sb., ale musí být uložen na kvalifikovaném prostředku, jako je např. čipová karta nebo USB token. [2]

1.2.2.3 Elektronický podpis

Rozdíly v elektronických podpisech u zákona č. 227/2000 Sb. a v nařízení eIDAS se liší v názvech i jejich právních úrovních. Nařízení eIDAS definuje tři úrovně elektronického podpisu. [2]

Podpis s nejnižší právní úrovní se nazývá také elektronický podpis. Zde má i velice podobnou právní úroveň a může být vytvořen pomocí komerčního certifikátu. Dalším typem je zaručený elektronický podpis, který je vytvořen kvalifikovaným certifikátem, který není uložen na kvalifikovaném prostředku typu čipová karta nebo USB token. Tento podpis má vyšší právní úroveň jako elektronický podpis, ale nižší úroveň než podpis vlastnoruční. Poslední úrovní podpisu je kvalifikovaný elektronický podpis. Ten může být vytvořen kvalifikovaným certifikátem, který je uložen na kvalifikovaném prostředku. Kvalifikovaný podpis má stejnou právní úroveň jako podpis vlastnoruční. [2]

Lze si povšimnout, že v nařízení má kvalifikovaný podpis nižší právní úroveň jako uznávaný elektronický podpis ze zákona č. 227/2000 Sb. V ČR tedy po zrušení zákona č. 227/2000 Sb. došlo k omezení možností použití elektronického podepisování.

1.2.2.4 Elektronická pečeť

Elektronickou pečeť lze přirovnat k elektronické značce ze zákona č. 227/2000 Sb. Vznikla za stejným účelem, umožnit podepisovat data právnickým osobám. Elektronické pečetě jsou ale striktnější, co se použití týče. Elektronickou pečeť může vytvořit pouze právnická osoba (elektronickou značku mohla i fyzická) a dále lze opečetit pouze data, která vznikla uvnitř právnické osoby (společnosti). Díky tomu, že právnická osoba může pečetit pouze svá data, elektronická pečeť také udává vlastníka pečetěných dat. [2]

Podobně jako u elektronického podpisu, i elektronická pečeť má tři formáty, které se liší ve své právní úrovni. Existuje elektronická pečeť, zaručená kvalifikovaná pečeť a kvalifikovaná elektronická pečeť. Zaručená kvalifikovaná pečeť je tvořena pomocí kvalifikovaného certifikátu, který není uložen na kvalifikovaném prostředku a kvalifikovaná elektronická pečeť je tvořena kvalifikovaným certifikátem, který je uložen na kvalifikovaném prostředku. [2]

1.2.2.5 Časová razítka

Nařízení eIDAS také definuje pojem časová razítka a zde jsou podmínky tvoreb razítek a jejich použití totožné jako v zákoně č. 227/2000 Sb.

1.2.3 Zákon č. 297/2016 Sb.

Zákon č 297/2016 Sb. přišel jako adaptační zákon pro nařízení eIDAS, zrušil platnost zákona č. 227/2000 Sb. a ukončil období mimořádné situace, kdy se v ČR dalo využít jak nařízení eIDAS, tak i zákona č. 227/2000 Sb.

Zákon přejmenoval poskytovatele certifikačních služeb na poskytovatele služeb vytvářející důvěru (kvalifikované poskytovatele služeb vytvářející důvěru). Dále zákon přijímá všechny typy podpisů z nařízení eIDAS a uznávaný elektronický podpis, který byl definován v zákoně č. 227/2000 Sb. změnil na elektronický podpis zaručený z nařízení eIDAS. Dále zákon ruší pojem elektronická značka a podporuje pouze elektronické pečetě z nařízení eIDAS. [3]

Zákon č. 297/2016 Sb. platí dodnes.

1.3 Technické normy digitálního podpisu

V předchozí kapitole bylo o elektronickém podpisu pojednáváno z hlediska, jak jej definují zákony. Zákony nám definují jednotlivé pojmy, úrovně použitelnosti jednotlivých podpisů, pečetí (značek). Když se v praxi vytváří nějaký podpis, musí být zaštítěn technickými normami, podle kterých se podpisy vytvářejí a ověřují. V této kapitole budou krátce rozepsány některé normy, podle kterých se podpisy vytváří.

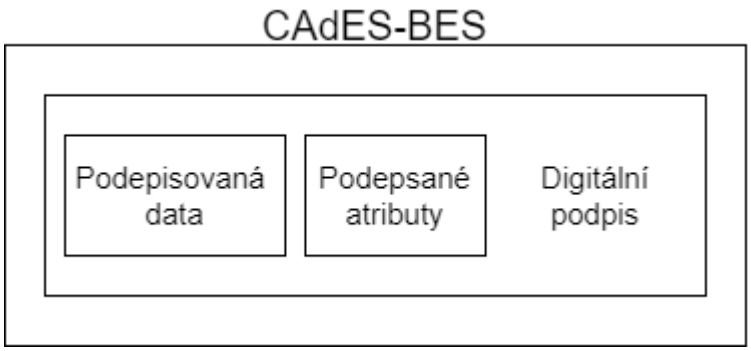
1.3.1 CAdES

CAdES má číselné označení normy jako (ETSI TS 101 733) a je založen na Cryptography Message Syntax (CMS). CMS vychází z principů infrastruktury veřejných klíčů (PKI – Public Key Infrastructure). CAdES přidává prostředky norem digitálních podpisů a lze jej aplikovat na jakákoli data. [4]

Existuje několik variant podpisu CAdES. Základní možnosti jsou CAdES-BES a CAdES-EPES, CAdES-T. Některé varianty podpisu CAdES umožňuji i tzv. dlouhodobou ověřitelnost. [4]

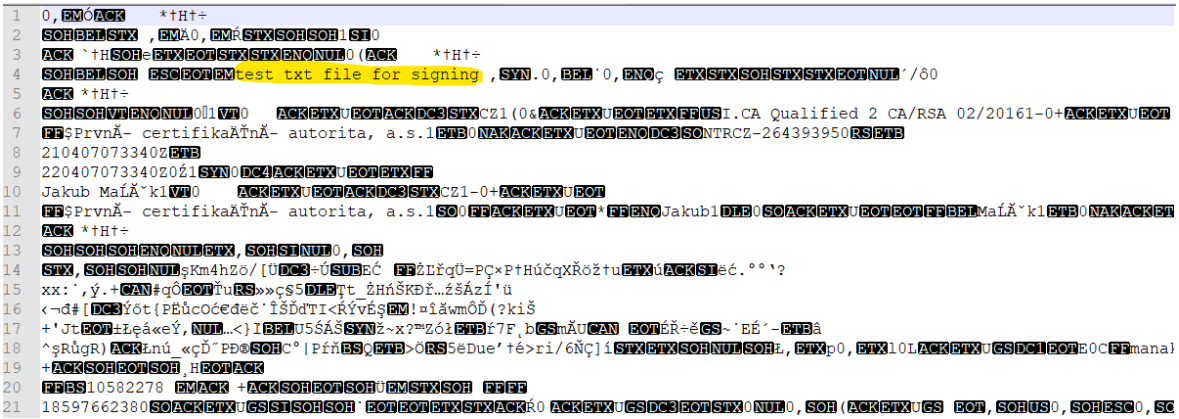
CAdES se používá po celé Evropské unii, ale jeho použití je spíše pro speciální aplikace. CAdES definuje způsob zpracování podpisu a principy jejího fungování, nediktuje ale způsoby, jak by měly být uchovány. Proto může být CAdES součásti podepisovaných dat nebo může být vytvořen i jako nový soubor vedle podepisovaných dat. Vytvořený podpis se poté zobrazuje jako binární data. [4]

CAdES-BES, který je součástí podepisovaných dat musí se skládá z více částí, kdy některé jsou povinné a některé jsou volitelné. Povinné položky podpisu jsou podepisovaná data, kolekce povinných podepisovaných atributů (Content-type dat, Hash otisk dokumentu a informace o podepisovacím certifikátu). Výsledný digitální podpis CAdES poté může mít strukturu, která je zobrazena na obr. 3. [8]



Obr. 3 Struktura podpisu CAdES-BES (vlastní)

Struktura dalších variant podpisů CAdES jsou podobné, kdy struktura většinou obsahuje nějakou položku navíc. Např. CAdES-EPES musí obsahovat v podepsaných atributech ID podepisovací politiky nebo CAdES-T obsahuje ve struktuře podpisu navíc časové razítko. [8]



Obr. 4 Ukázka podpisu CAdES (vlastní)

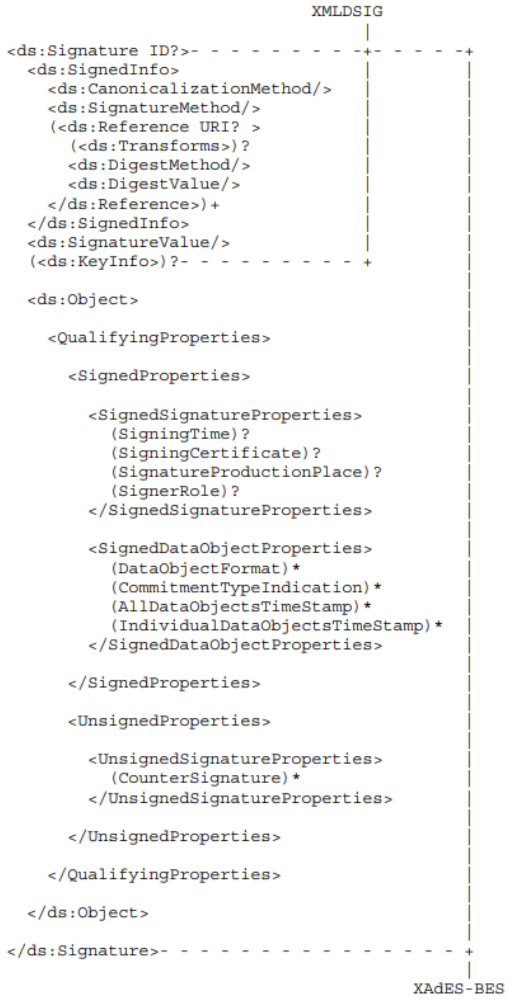
Na obr. 4 lze vidět, jak vypadá podpis CAdES-BES zobrazený v textovém prohlížeči, kdy byl podepisován textový dokument, kdy samotný digitální podpis je součástí podepisovaných dat. Podepsaná data jsou obklopena spoustou dalších údajů, a tak původní data dokumentu nemusí jít často vyčíst. Proto se v praxi podpis CAdES spíše vytváří jako samostatný soubor a umístí se vedle podepsaných dat.

1.3.2 XAdES

XAdES je další technickou normou při vytváření digitálních podpisů a její označení je ETSI TS 101 903). XAdES je postaven na startu digitálních podpisů XML-DSIG (XML struktura podpisu). Přesto že XAdES podpis fungující na standardech XML podpisů, lze s ním podepsat jakákoli data, včetně binárních, ale digitální podpis musí být vytvořen jako soubor

zvlášť. Pokud je požadováno, aby byl podpis XAdES součástí podepisovaných dat, lze tento podpis vytvořit pouze u XML dokumentů. [5]

XAdES stejně jako CAdES má více variant (např. BES, EPES, LTV – dlouhodobá ověřitelnost) Jednotlivé varianty podpisu XAdES také stavějí na základní variantě XAdES-BES a přidávají k nim další data (XAdES-T – časové razítko atd.). I XAdES se spíše používá u specializovaných aplikací. Na obr. 5 lze vidět základní XML strukturu podpisu XAdES-BES. [9]

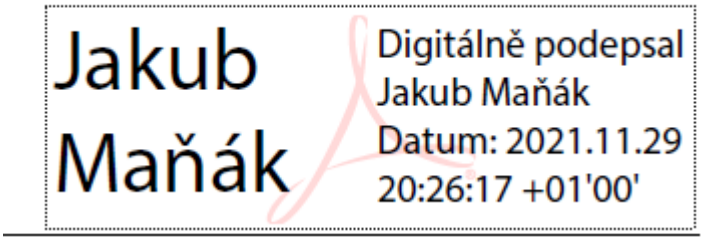


Obr. 5 XML struktura podpisu XAdES [9]

1.3.3 PAdES

PAdES je asi nejběžnější norma podpisu se kterou se může kdokoliv setkat. Označení normy PAdES je ETSI TS 102 778-1/6. PAdES se od CAdES a XAdES liší tím, že se aplikuje pouze na PDF dokumenty. Podpis PAdES je založen na podpisech z normy PKCS #7 (Public Key Cryptographic Standarts). PKCS #7 je předchůdcem CMS. [6]

Po podepsání PDF dokumentu je podpis součástí dokumentu. Norma je přímo zahrnuta v ISO PDF standartu, tudíž podepisování a ověřování lze provádět v rámci PDF prohlížečů (např. Adobe Acrobat Reader). PAdES podporuje také formuláře a umožňuje vytvořit vizuální reprezentaci podpisu, tzn. že k podpisu lze připojit libovolný obrázek. Např. lze si naskenovat svůj podpis, podepsat PDF dokument certifikátem a jako vizuální část podpisu přiložit naskenovaný obrázek našeho podpisu (není nutnost, může být i jakýkoli jiný obrázek. [6]



Obr. 6 Ukázka viditelného podpisu vytvořeného v Adobe Acrobat Reader (vlastní)

2 TECHNOLOGIE BLOCKCHAIN

Blockchainové systémy v posledních letech získávají na popularitě a stávají se velmi diskutovaným tématem. Většinu lidí, když se řekne pojem Blockchain, ihned napadnou kryptoměny jako Bitcoin nebo Ethereum, které dnes patří mezi nejznámější. Použití blockchainových systémů se nejhojněji používá u problematiky kryptoměn, ale není to pouze jediné využití těchto systémů. Výhod blockchainových systémů se začíná využívat i v průmyslu, kdy se do systému může ukládat např. historie postupu výroby určité součástky.

Co se ale vlastně skrývá pod pojmem Blockchain? Velmi zjednodušeně se dá říct, že se jedná o databázi, ale s velmi specifickými vlastnostmi a od tradičních databází se liší v určitých vlastnostech.

2.1 Rozdíly mezi tradičními databázemi a blockchainem

Hlavní rozdíly mezi blockchainem a tradičními databázemi jsou v druhu databáze a způsobu práce s daty v databázi. V následujících podkapitolách budou tyto rozdíly blíže rozebrány.

2.1.1 Rozdíly v druhu databáze

První rozdíl mezi tradičními databázemi a blockchainem je v druhu databáze. Většina tradičních databázových systémů jako např. SQL i (NoSQL) databáze jsou centralizované či decentralizované.

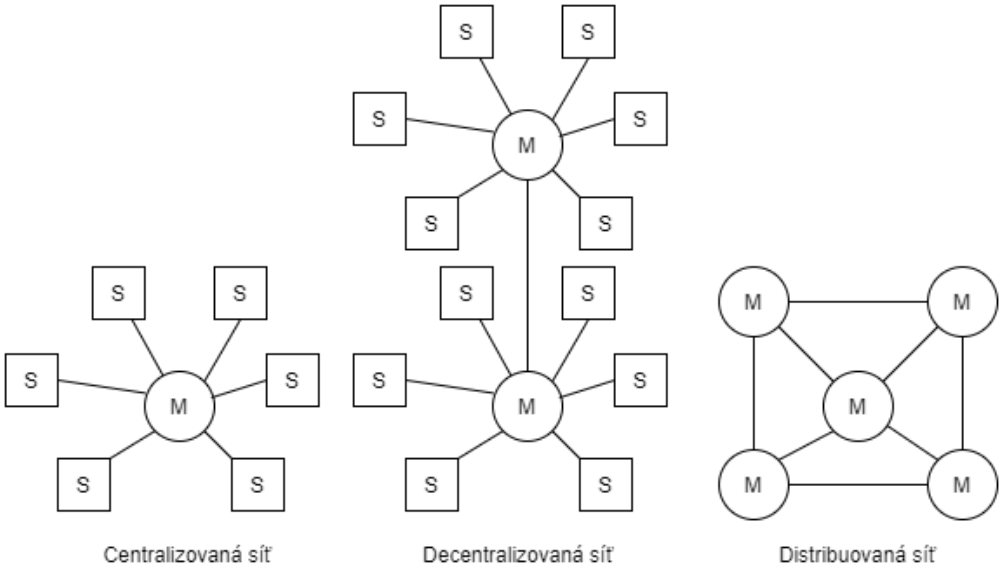
Centralizovaná síť má jedno hlavní centrum (master), a pokud master přestane fungovat (např. cíleným útokem hackerů nebo pouhou poruchou), celý databázový systém se stane nefunkční a data budou nedostupná. [10]

U decentralizovaných databází se databázová síť skládá již z více masterů, tudíž pro zničení celé sítě by bylo potřeba vyřadit z provozu všechny mastery. Pokud se podaří vyřadit z provozu např. jednoho mastera, klienti budou přepojeni na jiného mastera. [10]

Dalším druhem je distribuovaná databáze. Tato síť vychází z principu decentralizovaných sítí, kdy je navíc přidána distribuce dat mezi všemi uživateli sítě. Díky tomuhle propojení je síť naprosto soběstačná a o její chod se starají samotní uživatelé, proto zde není potřebná žádná centrální autorita. Distribuovanou síť je takřka nemožné zničit a čím více uživatelů se bude na síti podílet, tím se šance na zničení sítě menší. [10]

Na obr. 7 je graficky znázorněno, jak jednotlivé druhy sítí vypadají. Kolečka s písmenem M značí centrální místa sítě (Master) a čtverečky s písmenem S značí části, které jsou připojeny

k určitému centrálnímu bodu (Slave), např. zařízení přistupující k danému databázovému systému.



Obr. 7 Druhy databázových systémů (vlastní)

Blockchain je formou decentralizované distribuované databáze. Sít se skládá z velkého množství připojených počítačů, ale nikdo z nich není masterem. Na chodu sítě se podílejí všichni uživatelé na základě určitých pravidel, kterému se také říká konsens. [10]

2.1.2 Práce s daty

Tradiční databázové systémy nabízejí základní CRUD operace. CRUD je zkratka, která v sobě ukrývá čtyři základní operace – přidávání nových dat do databáze (Create), čtení dat z databáze (Read), aktualizování dat v databázi (Update) a mazání dat z databáze (Delete).

Systémy Blockchain umožňují pouze dvě operace, a to přidání nového záznamu do systémů (Create) a vyhledávání záznamu (Read). Všechny data, která jsou do blockchainu přidána, nelze nijak měnit ani mazat. Operaci čtení může v systému blockchain provést kdokoli. Zápis do databáze se provádí pomocí konsensu, který byl zmíněn i v minulé kapitole.

2.2 Transakce a bloky

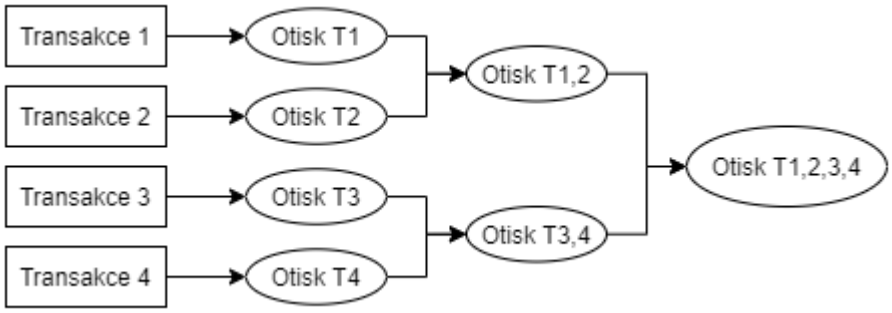
V systémech blockchain se vyskytují dva základní typy záznamů: transakce a bloky.

Transakce přidávají do databáze uživatelé. Obecně transakce může obsahovat libovolná data. Záleží na specializaci daného systému blockchain. Např. u systémů, které se zaměřují

na kryptoměny, se jedná o převod kryptoměny, u systému, které se zaměřují na odvětví průmyslu to může být aktualizace stavu přesunu nějakého výrobku. Každá transakce vždy musí splňovat nějaké základní podmínky. Tyto podmínky se opět můžou měnit na základě jednotlivých blockhainových systémů. Např. systém Bitcoin považuje za validní transakci, která obsahuje alespoň validní elektronický podpis uživatele a patrný finanční pohyb v peněžence uživatele. [11]

Transakce jsou poté obsaženy uvnitř bloků. Jeden blok může obsahovat libovolný počet transakcí. Transakce nejsou v bloku obsaženy kompletní, ale je z nich nejprve vypočítán otisk a ten se poté vkládá do bloku. Pokud blok obsahuje více transakcí, je výsledný otisk spočítán pomocí technologie Merkle Tree, česky nazývaný jako hašový strom. [12]

Merkle Tree je proces vytvoření jednoho výsledného otisku z libovolného počtu vstupních dat (zde transakcí). Nejprve se vypočtou otisky pro všechny transakce zvlášť, poté se vezmou dvojice otisků, ty se složí dohromady a vypočte se z nich nový otisk. Tímto způsobem se pokračuje do doby, dokud nezůstane pouze jeden výsledný otisk. Výsledný otisk je poté součástí bloku. Proces vypočtení otisku z více transakcí je zobrazen na obr. 8. [13]

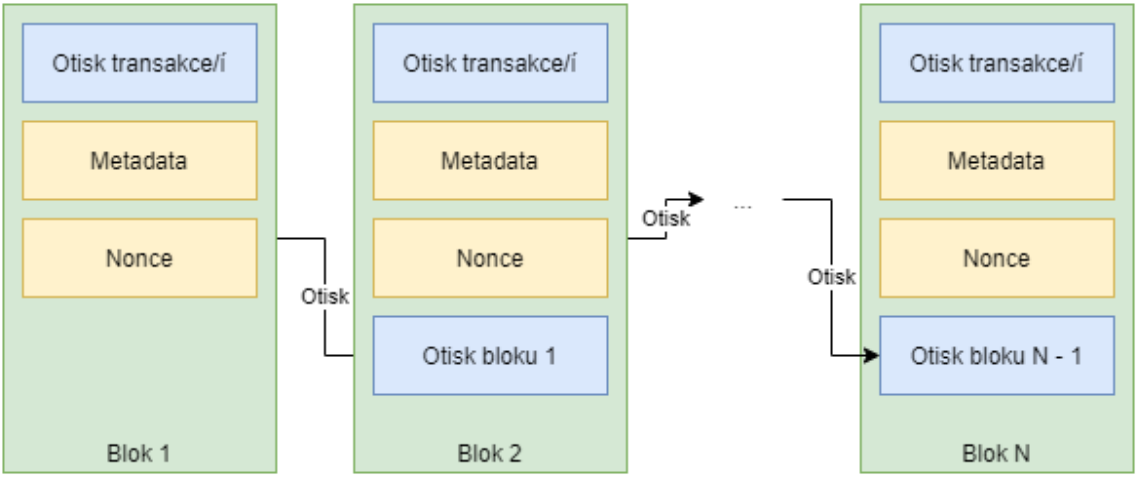


Obr. 8 Princip Merkle Tree (vlastní)

Dále blok musí obsahovat identifikátor předchozího bloku (pokud se tedy nejedná o první blok v systému). Identifikátorem předchozího bloku je konkrétně otisk kompletního bloku. Tím, že každý nový blok v systému obsahuje otisk bloku předcházejícímu dochází k řetězení bloků a dále tento krok znemožňuje měnit obsah jednotlivých bloků. Protože pokud by útočník změnil libovolný blok v řetězci, došlo by i ke změně jeho otisku a následující blok by na něj již neukazoval a došlo by k přerušení řetězu bloků. [12]

Otisk transakcí a otisk předchozího bloku jsou nejdůležitější a povinné části bloku. Dále blok může obsahovat další nepovinné položky, kterým se říká metadata. Mezi metadata lze zahrnout třeba datum vytvoření bloku a jiné. Další nepovinnou položkou, kterou lze přidat do

bloku je položka Nonce. Nonce je náhodně vygenerované velké číslo, které slouží ke snížení již tak nízké šance na podvržení bloku útočníkem, protože krom uhodnutí všech otisků v daném bloku musí útočník ještě uhodnout i hodnotu Nonce. [12]



Obr. 9 Ukázka bloků v blockchain (vlastní)

2.3 Uplatnění blockchain systémů

Nejtypičtější a nejznámější použití systémů blockchain je v dnešní době v odvětví kryptoměn, jako je např. Bitcoin, Ethereum, LiteCoin atd. Mnoho autorů článků na internetu se shoduje na velkém potencionálu použití blockchain jako budoucí měny, kdy díky principům blockchain by se dalo zabránit manipulaci s peněžními transakcemi s jednou centrální autoritou a zabránit případným úplatkům nebo různým finančním podvodům. [11]

Dále lze blockchain využít v průmyslu. Do blockchainu lze ukládat informace o historii převozu či výrobě dané součástky a tím zajistit úplnou historii vývoje dané součástky. Blockchain lze využít k ověření stavu dokumentu, zda s ním bylo od vložení do blockchain manipulováno. Využitím blockchain systémů se v dnešní době zabývají některé známé mezinárodní společnosti, jako je např. IBM. [11]

Další typy využití systémů blockchain by mohlo být pro katastr nemovitostí, kde by díky blockchain bylo možno zabránit neoprávněnému nakládání s nemovitostmi, dále neoprávněnému šíření různých uměleckých děl (filmy, písničky atd.). [11]

2.4 Využití systému blockchain v DP

Ve 2. kapitole je popsán princip fungování blockchain systémů a k čemu jsou vhodné. Myšlenka této DP je využití blockchain pro prodloužení platnosti digitálního podpisu, který je

vytvořen pomocí digitálního certifikátu na dobu neurčitou právě díky vložení informace o digitálním podpisu a podpisovém certifikátu do blockchain. Největší strastí digitálních podpisů je právě jejich platnost. V dnešní době již existují normy, jak vytvořit podpis, který může platit klidně i několik desítek let, ale i tato doba platnosti někdy skončí. Pokud by se podepsaná data (či jejich otisk) a certifikát (nebo otisk certifikátu) vložili do blockchain, bylo by možné digitální podpis považovat za platný a pravý i po vypršení doby platnosti podle norem digitálních podpisů (např. CAdES-BES, PAdES-LTA, atd.)

3 MOŽNOSTI IMPLEMENTACE APLIKACE CERTICHAIN

Aplikace CertiChain se bude skládat ze dvou částí – Front End (FE) a Back End (BE). Obě části aplikace budou implementovány samostatně a každá část může být implementována pomocí různých technologií. Tato kapitola bude pojednávat o jednotlivých technologiích, pomocí kterých by bylo možné BE i FE naimplementovat.

3.1 Možnosti implementace BE

Technologie, které by mohly být použity k implementaci BE, musí splňovat několik kritérií. Pokud by tato kritéria nesplňovala, nebylo by možno aplikaci naimplementovat podle návrhu, který je popsán ve 4. kapitole.

Prvním kritériem, aby daná technologie umožňovala vývoj serverových REST API, tedy aby po sestavení projektu a následném nasazení na server odposlouchávala http(s) požadavky na serverové adrese a určitém portu, který bude aplikaci nastaven.

Druhým kritériem je, které technologie musí podporovat, je připojení k databázi. Pro splnění všech funkcí a požadavků na aplikaci CertiChain je kritické mít připojení k persistentnímu úložišti, které uchová data i v případě výpadku serveru, a aby persistentní úložiště dokázalo efektivně fungovat i v případě většího objemu dat a více příchozích požadavků v jeden moment. Pro persistentní ukládání aplikačních dat se vybízí použití relační databáze nebo dokumentové NoSQL databáze.

Třetím kritériem je možnost zaimplementování procesu autentizace, konkrétně aby k určitým funkcích BE aplikace mohla přistoupit pouze autentizována osoba a při přístupu do neoprávněné části aplikace mít možnost uživatele určitým způsobem informovat o jeho nepovolené akci pro kterou nemá dostatečná práva.

V roce 2022 je výběr technologií pro implementaci BE systémů rozsáhlý, a i přes výše popsaná kritéria lze vybírat ze spousty programovacích jazyků a jejích BE frameworků. BE aplikace CertiChain by bylo možno implementovat např. pomocí C# .Net, Java, Python, PHP, Kotlin, C++ a jistě i v dalších programovacích jazycích, které zde nejsou uvedeny.

3.1.1 C# .NET

Pro vytváření webových API nebo i kompletních webových aplikací pomocí programovacího jazyka C# a knihoven .NET Core, dnes nazývanývh jen .NET. .NET Core dnes již nevydává nové verze a byl sjednocen společně s .NET Frameworkem pod jeden společný

název .NET, kdy první verze společného .NETu nese číslo 5. S tímhle sjednocením přišel Microsoft v listopadu roku 2020 a číslo verze 5 místo 4 bylo použito z důvodu existence verze .NET Frameworku 4. V roce 2022 je aktuální verze .NET číslo 6 a již je k dispozici preview verze .NET 7.

Pro tvorbu kompletních webových aplikací nebo pouze webových služeb (např. REST) lze použít nástroj Asp.Net. Tento nástroj nabízí sadu tříd, pomocí kterých je vývoj webových služeb dosti zjednodušený a nástroje Asp.Net udělají spoustu práce za vývojáře.

Ve vývojovém prostředí Visual Studio je dostupný typ projektu „ASP.NET Core Web API“, kdy při zakládání projektu základní šablona obsahuje ukázku implementace jednoduchého REST-ového API.

Třídy, které zpracovávají jednotlivé služby, které daná služba obsahuje, se nazývají kontroléry (Controllers). Pro vytvoření Controlleru v Asp.Net je potřeba aby daná třída Controlleru dědila od třídy ControllerBase a nad deklarací třídy se musí nacházet atribut *ApiController*.

```
[ApiController]
0 references
public class TodoListController : ControllerBase
{

}
```

Obr. 10 Ukázka vytvoření Controlleru v Asp.Net (vlastní)

Ve třídě Controlleru se poté přidávají metody, které budou obsluhovat jednotlivé požadavky podle zadané cesty v URL požadavku. Jednotlivým metodám lze nastavovat jaké metody http(s) komunikace můžou obsluhovat (např. GET, POST, PUT, DELETE). Metody, které obsluhují danou http metodu a cestu, se vytvoří pomocí atributu, který se vloží nad definici metody. Pro každou metodu http požadavku je k dispozici atribut (např. pro GET je atribut HttpGet, pro POST atribut HttpPost atd.). Atributům je potřeba v parametru předat cestu koncové adresy, kterou bude daná metoda obsluhovat.

Např. webová služba běží na URL adrese https://test.aspnet.cz. Pokud je v Controlleru metoda s atributem *HttpGet(„todos/getAll“)*, do těla metody skočí aplikace v momentě, pokud bude na server vyslán http GET požadavek s URL adresou https://test.aspnet.cz/todos/getAll. Asp.Net se poté na pozadí stará o převádění navrácených objektů z funkcí do JSONu nebo XML, taktéž se automaticky pokusí převést tělo požadavků z JSON nebo XML na objekt, který je uveden jako parametr funkce s atributem *FromBody*.

```
[ApiController]
0 references
public class TodoListController : ControllerBase
{
    [HttpGet("todos/getAll")]
    0 references
    public List<TodoModel> GetTodos()...

    [HttpPost("todos/addNew")]
    0 references
    public void AddTodo([FromBody] TodoModel todoModel)...
}
```

Obr. 11 Ukázka vytvoření metod v Controlleru (vlastní)

V Asp.Net je možné komunikovat s relačními databázemi. S databází lze komunikovat vícero způsoby. Lze využít přístupu, kdy vývojář může ručně vytvářet SQL příkazy nad připojenou databází nebo může využít knihovnu Entity Framework, pomocí které lze přistupovat k jednotlivým entitám databáze pomocí objektů způsobem Objektově Relačního Mapování (ORM). Entity Framework je primárně navržena pro databáze MSSQL, ale umí pracovat i s dalšími relačními databázemi (např. MySql, SQLite, CosmosDB, PostgreDB).

```
0 references
public class AppDbContext : DbContext
{
    0 references
    public DbSet<TodoModel> TodoDbSet { get; set; }
    0 references
    public DbSet<WeatherForecast> WeatherForecastDbSet { get; set; }
}
```

Obr. 12 Ukázka použití Entity Frameworku (vlastní)

Na Obr. 11 lze vidět ukázku, jak pomocí Entity Frameworku pracovat s databází. Třída *AppDbContext* představuje databázi a vlastnosti *TodoDbSet* a *WeatherForecastDbSet* jsou tabulky v databázi.

Asp.Net obsahuje také různé možnosti pro implementaci procesu autentizace. Pokud vývojář nechce implementovat vlastní způsob autentizace (např. pomocí JWT tokenů), může použít zabudovaný Identity Framework, který za vývojáře autentizaci vyřeší. Pokud vývojář chce autentizaci implementovat sám, k dispozici má základní třídy, od kterých musí dědit jeho vlastní implementace autentizace. Vytvořená implementace autentizace se poté nastaví v konfiguraci, která se provádí při spouštění služby.

Webová služba implementovaná pomocí .NET lze nasadit na Windows i Linux systémy, lze také použít nasazení pomocí Docker kontejneru. .Net lze použít také na ARM procesorech, tudíž lze webovou službu nasadit i na Embedded zařízení typu Rasberry Pi apod.

3.1.2 Java

Programovací jazyk Java je také hojně používán pro tvorbu webových služeb, jak REST-ových, tak i SOAP-ových. V Java jde zpracovávat http požadavky vícero způsoby. Pokud vývojář nechce používat žádné dostupné frameworky, může si službu vytvořit úplně od píky s použitím Java tříd nazývaných Servlet.

V Java je k dispozici vícero frameworků, které vývojářům ulehčují tvorbu webových služeb. Nejznámější a asi i nejpoužívanější framework, který bude i stručně popsán v této kapitole, je Java Spring. Java Spring je robustní open-source framework, který je k dispozici již od roku 2003 a i v roce 2022 je stále vyvíjen a zdokonalován.

Java Spring slouží k zjednodušení tvorby jak webových služeb, tak i webových aplikací. Java Spring obsahuje spoustu dodatečných balíčků, které vývojářům maximálně ulehčují a urychlují vývoj webových aplikací či služeb. Obsahuje balíčky pro usnadnění práce s databázemi, kdy nabízí ORM přístup k SQL databázím (např. MySql, Postgre, OrientDB aj.) i k NoSql databázím (např. MongoDB). Dále obsahuje balíčky pro usnadnění tvorby a zasílání emailových zpráv, balíček pro zakomponování autorizace do aplikací a mnoho dalších. [12]

Tvorba REST API pomocí Java Spring je velmi podobná jako v .NET. Např. pro vytvoření kontroléru, který bude obsluhovat určité http(s) požadavky stačí také vytvořit třídu, nad kterou se přidá anotace RestController. Pro obsloužení požadavku se specifickou URL cestou stačí vytvořit funkci uvnitř kontroléru a nad cílovou funkci přidat příslušnou anotaci podle cílové http metody s parametrem path, pomocí které se nastaví koncová adresa, kterou daná funkce obslouží: [12]

- GET – GetMapping(path = „/hello/world“)
- POST – PostMapping(path = „/hello/world“)
- PUT – PutMapping(path = „/hello/world“)
- DELETE – DeleteMapping(path = „/hello/world“)

Na obrázku 13 je ukázka vytvoření jednoduchého kontroléru pomocí Java Spring. Na obrázku lze vidět že postup se velice podobá ukázce kontroléru vytvořeném v .NET, který lze vidět na obrázku 11.

```
@RestController
public class HelloController {
    @GetMapping(path = "/hello/world")
    public ResponseEntity<?> getHelloWorld() {

    }

    @PostMapping(path = "hello/world")
    public ResponseEntity<?> postHelloWorld(@FromBody HelloWorldBodyObject body) {

    }
}
```

Obr. 13 Ukázka tvorby kontroléru v Java Spring (vlastní)

Na obrázku 13 lze vidět i ukázku, jak získat objekt z těla požadavku, např. při obsluze http POST požadavku. Postup je opět velmi jednoduchý a celý proces probíhá automaticky na pozadí a vývojář se nemusí o nic starat. Stačí použít anotaci *@FromBody*, a přidat parametr do funkce a Spring se automaticky pokusí převést tělo požadavku např. z JSON na daný objekt. Pokud převod z JSON na objekt selže, Spring automaticky vrátí volajícímu odpověď s http statusem 401 – BadRequest. [12]

Jak bylo zmíněno na začátku této kapitoly, Spring umožňuje komunikovat s relačními databázemi i s NoSql databázemi (MondoDb) pomocí ORM. K tomu slouží balíček Hibernate, který celou tuhle funkcionalitu zastřešuje. Pro vytvoření tabulky je nejprve potřeba vytvořit entitu. Entita se vytvoří pomocí Java třídy, na kterou se vloží anotace *@Entity*. Poté stačí vytvářet jednotlivé proměnné uvnitř třídy, což budou jednotlivé atributy výsledné tabulky. Jednotlivé proměnné lze ještě parametrizovat pomocí dalších anotací, jako např. definování primárního klíče pomocí anotace *@Id*. [12]

```
@Entity
public class ExampleEntity {
    @Id @GeneratedValue private long id;
    private String firstName;
    private String lastName;
}
```

Obr. 14 Ukázka vytvoření Entity pomocí Java a Hibernate (vlastní)

Dále je potřeba entitu zastřešit do tabulky. Toho se docílí pomocí Java rozhraní, které rozšiřuje rozhraní JpaRepository. Do JPA rozhraní se vloží mezi ostré závorky ještě datový typ entity (název třídy, která definuje entitu) a datový typ primárního klíče (např. číslo jako int, long nebo unikátní identifikátor GUID, v Javě nazýván UUID). Poté se třída reprezentující tabulku musí označit anotací *@Repository*. Rozhraní JpaRepository má již předdefinované

všechny základní CRUD operace. Pokud nejsou potřeba žádné další speciální funkce, např. nějaké speciální vyhledávání, není do rozhraní již potřeba přidávat další zdrojový kód. [12]

```
@Repository
public interface ExampleRepository extends JpaRepository<ExampleEntity, Long> {

}
```

Obr. 15 Ukázka vytvoření tabulky pomocí Java a Hibernate (vlastní)

K takto definované tabulce už je možno přistupovat v kódu a provádět nad ní CRUD operace. Ještě zbývá uvést, jak tabulku použít např. v kontroléru. Java Spring má v sobě zabudovaný IoC kontejner. Již dříve uvedené anotace jako např. *@Controller*, *@Repository* s IoC kontejnerem také pracují. U takhle anotovaných tříd při spouštění dojde k automatickému vytvoření jejich instancí a následnému vložení do IoC kontejneru. Tyto instance tříd je poté možno injektovat v jednotlivých třídách, např. v kontroléru pomocí anotace *@Autowired*.

```
@RestController
public class HelloController {

    private final ExampleRepository exampleRepository;

    @Autowired
    public HelloController(ExampleRepository exampleRepository) {
        this.exampleRepository = exampleRepository;
    }
```

Obr. 16 Ukázka použití IoC v Java Spring (vlastní)

Poslední věc pro správné fungování je nastavení připojení k databázi. Poté při spouštění Spring a Hibernate se již automaticky připojí k cílové databázi, vytvoří si databázi, postará se o vytvoření nových tabulek, či aktualizování tabulek již existujících.

Nasazení backendové aplikace vytvořené v Java Spring je také velice variabilní. Aplikace může běžet na Windows, Linux, i MacOS. Nebo lze vytvořit Docker kontejner ve kterém aplikace poběží. Menší nevýhoda použití Java Spring je v robustnosti frameworku, kdy po nasazení aplikace na server může aplikace spotřebovávat větší množství paměti a více výkonu procesoru.

3.1.3 Python

Tvorba backendových systémů pomocí Pythonu získává v posledních letech na popularitě a také Python už má frameworky pro tvorbu REST API. Dva nejznámější frameworky pro tvorbu REST API v Pythonu jsou Django a Flask. Django je starší framework, který byl představen v roku 2005, má větší komunitu i skupinu vývojářů, kteří framework vylepšují, ale Django se spíše specializuje na vývoj full stack webových aplikací než pro vývoj webových API. Flask je více „lightweight“, je vyvíjen od roku 2010 a specializuje se hlavně na vývoj REST API služeb. Pro potřeby vývoje backendu aplikace CertiChain by tedy mohl více vyhovovat framework Flask, který bude v této kapitole rozebrán. [15]

Pro obsluhu http požadavků pomocí frameworku Flask stačí přidávat do Python skriptu funkce nad které se přidá dekoratér (v angličtině decorator) *@app.route()* s textovým parametrem, který definuje koncovou adresu požadavku. V dekoratéru je poté ještě potřeba definovat typ http požadavku, přiřazením parametru methods, který přijímá seznam textových řetězců. [16]

```
@app.route('/incomes')
def get_incomes():
  return jsonify(incomes)


@app.route('/incomes', methods=['POST'])
def add_income():
  incomes.append(request.get_json())
  return '', 204
```

Obr. 17 Ukázka funkcí pro obsluhu http požadavků pomocí Flask (vlastní)

Python neumožňuje automaticky na pozadí převádět objekty do JSON nebo XML jako Java nebo C# .NET, ale pro tyto účely je dostupný balíček Marshmallow, pomocí kterého si lze převedení naimplementovat. [16]

Python také disponuje balíčky, pomocí kterých lze jednoduše realizovat autentizaci a autorizaci v aplikaci. Na autorizaci lze použít např. balíček Auth0, který vyřeší spoustu práce za vývojáře. Při použití balíčku Auth0 je potřeba pouze vytvořit metodu pro kontrolu, zda je uživatel autentizován a poté nad jednotlivé koncové body aplikace přidat dekoratér *@requires_auth*. [16]

Python disponuje knihovnami, které umožňují ORM přístup k databázím. Jedna z těchto knihoven se nazývá SQLAlchemy. Dále pro připojení k databázi je potřeba použít určitou knihovnu, která implementuje Python DBAPI. Knihovnu pro připojení je potřeba vybrat na základě použité databáze. V Pythonu je možno komunikovat s mnoha typy SQL databází (PostgreSQL, MySQL, MSSQL, SqLite) a i s řadou NoSQL databází (BerkeleyDB, Neo4j). SQLAlchemy se poté chová jako wrapper nad knihovnou implementující Python DBAPI a pomocí SQLAlchemy se volají CRUD operace nad samotnou databází. [17]

Jednotlivé tabulky uvnitř databáze se vytvářejí jako třídy, kdy jednotlivé vlastnosti uvnitř třídy jsou sloupečky uvnitř tabulky. Každá proměnná ve třídě tabulky (entity) musí být typu Column. Třída Column přebírá v konstruktoru parametry, jak má daný sloupeček v tabulce vypadat. Lze si nastavit jméno sloupce, typ sloupce a případně další parametry jako např. primární klíč. Dále si lze nastavit i jméno celé tabulky pomocí předdefinované vlastnosti *__tablename__*. [17]

```
class Product(Base):
    __tablename__ = 'products'
    id=Column(Integer, primary_key=True)
    firstName=Column('first_name', String(32))
    lastName=Column('last_name', String(32))
```

Obr. 18 Ukázka vytvoření entity pomocí SQLAlchemy (vlastní)

Pro práci se samotnou databází je potřeba vytvořit engine, který se napojí na cílovou databázi a poté ještě třídu session, kdy pomocí jejich instancí se bude manipulovat s databází. Pomocí session lze již provádět CRUD operace, kdy při přidávání či aktualizování záznamů v tabulce je potřeba po aktualizaci dat provést commit dané session a po skončení práce s danou session by se měla uzavřít. [17]

```
# připojení k databázi
engine = create_engine('postgresql://usr:pass@localhost:5432/mydb')
# vytvoření třídy pro DB sessions
Session = sessionmaker(bind=engine)
# vytvoření session
session = Session()
# přidání záznamu
session.add(entity)
# získání záznamů
session.query(User).all()
# commit session
session.commit()
# uzavření session
session.close()
```

Obr. 19 Ukázka práce s databází pomocí SQLAlchemy (vlastní)

Backendovou aplikaci lze nasadit na libovolný počítač, kde je dostupný Python, lze tedy využít počítače s Windows, Linux nebo i MacOS. Aplikaci lze také nasadit do Docker kontejneru. Při práci s Pythonem nejsou některé operace dělány automaticky jako např. v Java nebo C# .NET, výhodou Pythonu je ale jeho spotřeba výkonu, která bude jistě menší než při použití např. Java Spring.

3.1.4 PHP

Další technologií, kterou by bylo možno bez debat použít pro implementaci Backend části aplikace, je jazyk PHP. Pomocí PHP by bylo možno celý projekt vytvořit od píky bez použití frameworků. Tento postup by ale znamenal spoustu práce a spotřebovaného času navíc. PHP má spoustu frameworků pomocí kterých lze tvořit jak REST API, tak i kompletní Full Stack webové aplikace. Z dostupných frameworků lze zmínit Laravel, na který bude zaměřena i tato kapitola, dále např. frameworky Lumen, Slim, Guzzle a další.

Framework Laravel je primárně vhodný na tvorbu Full Stack webových aplikací, je možné jej použít ale i na tvorbu REST API. Výhoda použití frameworku Laravel je v nabídce poskytovaných knihoven. Laravel nabízí knihovny pro zakomponování procesu autentizace, obsahuje knihovny pro ORM přístup k databázi, IoC kontejnery, kdy použití všech těchto knihoven vývojářům ušetří hodně boiler plate kódu.

Obsluha jednotlivých http požadavků probíhá v kontrolérech. Každý kontrolér, který se v aplikaci vytváří, musí dědit od základní třídy Controller, která je součástí frameworku Laravel. Aby se po přijetí daného požadavku obsluha přesunula do správné metody daného Controlleru, je nutné vytvořit přesměrování v souboru api.php, který je součástí projektu. [18]

```php
// Vytvoření Controlleru
class UserController extends Controller
{
    public function getUsers()
    {

    }

    public function addUser(User $user)
    {

    }
}

// Vytvoření Routes v souboru api.php
Route::get('/user/getAll', [AuthController::class, 'getUsers']);
Route::post('/user/add', [AuthController::class, 'addUser']);
```

Obr. 20 Ukázka vytvoření kontroléru a Routes pomocí Laravel (vlastní)

V Laravelu podobobně jako u Flasku v Pythonu se také musí serializace do JSON implementovat „manuálně“. Laravel obsahuje doplněk jménem Eloquent, který slouží na serializaci objektu do JSON a deserializaci JSON do objektu. Díky Eloquent serializace do JSON spočívá ve vytvoření třídy, která dědí od třídy JsonResource. Poté když je potřeba objekt převést do JSON, stačí na cílový objekt zavolat metodu *toJson()*. [19]

```
// Definování třídy pro serializaci JSON <-> objekt
class UserResource extends JsonResource
{
    public int $id;
    public string $firtName;
    public string $lastName;

    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'firtName' => $this->firtName,
            'lastName' => $this->lastName,
            'created_at' => $this->created_at,
            'updated_at' => $this->updated_at,
        ];
    }
}

// Serializace objektu do JSON
$user->toJson()
```

Obr. 21 Ukázka serializace objektů pomocí Eloquent (vlastní)

Laravel nabízí funckionalitu autentizace v aplikacích a zakomponovat ji do aplikace lze několika způsoby. Základní možnosti využití autorizace je využití tzv. Gates, ve kterých si lze nadefinovat pravidla autorizace a poté Gates použít v jednotlivých funkcích kontroléru. Dále Laravel nabízí dva doplňky, které vývojářům ulehčují zakomponování autentizace do jejich aplikací. Tyto doplňky se nazývají Laravel Sanctum, který podporuje OAuth autentizaci a Laravel Passport, který se zaměřuje na OAuth2 autentizaci. Přípravu autentizace mají oba doplňky velmi podobnou. Princip spočívá ve vytvoření pár tříd, které dědí od základních tříd, které jsou v doplňcích dostupné. [22, 23]

Další využití doplňku Eloquent je pro práci s databází pomocí přístupu ORM. Pro reprezentaci tabulky databáze stačí pomocí Eloquent vytvořit třídu s vlastnostmi, které budou reprezentovat jednotlivé sloupce tabulky. Poté daná třída entity musí dědit od třídy Model. Pro definování primárních klíčů či nastavení jména tabulky je možno pomocí speciálních vlastností, které se definují uvnitř třídy entity. [20]

```php
class User extends Model
{
    // sloupce tabulky
    public int $id;
    public string $firtName;
    public string $lastName;

    // název tabulky
    protected $table = 'users';
    // určení primárního klíče
    protected $primaryKey = 'id';
    // příznak na automatickou inkrementaci primárního klíče
    public $incrementing = true;
}
```

Obr. 22 Ukázka vytvoření entity pomocí Eloquent (vlastní)

Po spuštění databáze se poté Eloquent snaží najít danou tabulku v připojené databázi. Eloquent si není schopen vytvořit databázi automaticky jako např. Java Spring s Hibernate, ale je potřeba si vytvořit migraci databáze. Připojení k databázi se provádí v souboru *config/database.php*, který je dostupný v projektové šabloně Laravel projektu. Eloquent umožňuje pracovat s databázemi MariaDB, MySql, PostgreSql, SQLite a MSSQL server. [21]

Pro CRUD operace s jednotlivými tabulkami poté stačí přidat danou entitu do cílové třídy a voláním metod, které jsou definovány v rodičovské třídě Model, od které musí každá entita dědit. Pro vytvoření nového záznamu stačí vytvořit novou instanci cílové entity a zavolat metodu *save()*, pro čtení z databáze slouží metody *all()* či *where()*, pokud je třeba získat záznamy, které splňují požadovanou podmínku. Pro aktualizování záznamu stačí najít požadovaný záznam, aktualizovat hodnotu dané vlastnosti a poté použít metodu *save()*. [20]

```php
// přidání záznamu
$newUser = User;
$newUser->save();

// získání záznamů
$allUsers = User::all();
$filteredUsers = User::where('firstName', 'Joe');

// aktualizace záznamu
$user = User::find(1);
$user->firstName = 'Jacob';
$user->save();

// smazání záznamu
$user = User::find(1);
$user->delete();
```

Obr. 23 Ukázka práce s entitami pomocí Eloquent (vlastní)

3.2 Možnosti implementace FE

V minulé kapitole byly popsány možnosti implementace BE části aplikace CertiChain. V této kapitole budou rozepsány možnosti implementace části Front Endové, se kterou bude přímo interagovat uživatel aplikace.

Na výběr technologie pro implementaci FE je nutno nejprve se rozhodnout na jakou platformu aplikace cílí. Zda je potřeba pokrýt všechny platformy, tedy mobilní i desktopové a zda je potřeba cílit na všechny majoritní operační systémy, tedy Windows, Linux a MacOS pro platformy desktopové a Android a iOS pro platformy mobilní.

V dnešní době je možnost využít několik cest, kterými se lze vydat při vytváření FE aplikace. Způsobem nejpracnějším (v případě podpory všech výše zmíněných platforem) je cesta nativního vývoje. Tento přístup je sice časově nejnáročnější, ale na každé vyvíjené platformě lze využívat jejich nativních komponent a knihoven.

Další způsob implementace je možnost využití multiplatformních technologií a programovacích jazyků. Např. pomocí Java lze vytvářet aplikace pro desktopové systémy, které běží na Windows, Linux či MacOS. Pomocí Java lze vytvořit i mobilní aplikaci, ale zde je zapotřebí jiných knihoven a tvorby GUI aplikace než pro desktopovou platformu. Co se týče multiplatformních technologií, zde má .NET trochu napřed se svými frameworky, které umožňují vytvořit aplikace na všechny platformy i systémy včetně těch mobilních.

Posledním způsobem, jak aplikaci vytvořit pro všechny platformy, je vytvoření webové aplikace. Díky tvorbě webové aplikace lze cílit na všechny platformy pomocí jednoho řešení, jen je potřeba přizpůsobit a připravit vzhled aplikace pro všechny typy displejů. Problémem při vývoji webové aplikace je, že nelze využít všech vlastností daných zařízení jako při vývoji nativním či multiplatformním.

Aplikace CertiChain cílí spíše na desktopovou platformu, ale bylo by vhodné, kdyby byla dostupná na všech majoritních operačních systémech. Při vytváření FE aplikace lze tedy využít cestu nativního vývoje, multiplatformních technologií či vytvořit webovou aplikaci.

3.2.1 Nativní technologie

Nativní vývoj pro všechny desktopové systémy by znamenalo vytvořit tři samostatné aplikace, kdy každá by cílila na jeden operační sytém, tedy MacOS, Linux a Windows.

3.2.1.1 Vývoj aplikace pro MacOS

Jeden z hlavních technických problémů při vývoji nativní aplikace pro MacOS je potřeba vlastnit desktop či laptop od Apple. Existuje ještě možnost pronajmutí Apple zařízení a vzdáleně se na něj připojit. Po vyřešení problému se zařízením nabízí vývojářům Apple zdarma vývojové prostředí XCode a poměrně mladý, ale již schopný programovací jazyk Swift a grafickou knihovnu SwiftUI. Výhoda použití jazyka SwiftUI je možnost vytváření aplikace pro desktopové i mobilní zařízení pomocí jednoho projektu.

Grafická knihovna SwiftUI disponuje řadou grafických prvků, pomocí kterých lze seskládat celé GUI aplikace. Všechny mají samozřejmě nativní styl, který lze vidět např. v nastavení systému MacOS. Tvorba samotného GUI probíhá buďto pomocí designéru nebo psaním kódu. Obsluhu jednotlivých tlačítek a logiku aplikace je poté možné psát přímo so souboru s kódem dané obrazovky. Nebo lze aplikace vyvíjet pomocí architektury MVC nebo MVVM, kdy kombinace programovacího jazyka Swift a grafické knihovny SwiftUI oba tyto návrhové vzory podporuje, a oddělit logiku aplikace od samotné obrazovky.

```
HStack {
    TextField("Zadejte město", text: $cityName)
    Button("Hledat") {
        if(cityName.isEmpty) {
            showAlert.toggle()
            return
        }

        getCurrentWeather()
    }
}
.padding()
```

Obr. 24 Ukázka tvorby obrazovky pomocí Swift UI (vlastní)

Část kódu zobrazená na obrázku 24 vytvoří v GUI neviditelný kontejner (*HStack*), do kterého se můžou vkládat další prvky a kontejner bude tyto položky vkládat horizontálně vedle sebe. Uvnitř kontejneru je vloženo pole na vložení textového vstupu od uživatele (*TextField*) a tlačítko (*Button*).

Programovací jazyk Swift samozřejmě obsahuje třídy, pomocí kterých lze vytvářet http požadavky a komunikovat s BE systémy, obsahuje i třídy pro převod objektů do JSON a naopak. FE aplikace CertiChain by tedy bez problému bylo možno naimplementovat na MacOS systémech pomocí této nativní technologie.

3.2.1.2 Vývoj aplikace pro Linuxové systémy

U Linuxových systémů není dostupný žádný nativní grafický framework, který by byl čistě jen pro Linuxové systémy. Linuxové systémy mají několik grafických nadstaveb, jako např. Gnome. K dispozici jsou poté GUI knihovny, které si z těchto grafických nadstaveb získají vzhled daných prvků.

Mezi tyto knihovny patří např. knihovna GTK nebo QT. Obě knihovny lze použít pro vytváření GUI aplikací i na systémech Windows či MacOS, takže spadají mezi multiplatformní frameworky. Při nativním způsobu tvorby aplikace na Windows a MacOS, ale není jiného východiska než pro aplikaci na systému Linux zvolit jedno z těchto řešení. Knihovně QT bude věnována kapitola v sekci o možnosti využití multiplatformního frameworku pro vytvoření FE aplikace, proto v této kapitole bude stručně rozebrána knihovna GTK.

GTK je sada knihoven pro tvorbu GUI aplikací a je vyvíjena již od roku 1998 a do dnešního dne je stále aktualizována. Velikou výhodou GTK je velké množství programovacích jazyků, které lze použít pro vytváření aplikací. Pro vytvoření GUI aplikace pomocí GTK lze využít např. C++, JavaScript, Perl, Python další.

Přidávání jednotlivých prvků do aplikace lze dělat dvěma způsoby. GUI lze navrhnout čistě pomocí programovacího jazyka, kdy se ve zdrojovém kódu vytvářejí instance tříd jednotlivých prvků. Druhá možnost je popsat strukturu aplikačního okna pomocí tzv. GtkBuilder XML formátu. GTK také nabízí GUI aplikaci Glade, která slouží na interaktivní návrh grafického rozhraní. Po navržení okna je možné strukturu aplikace vyexportovat ve formátu GtkBuilder.

```
int main() {
    // načtení xml struktury okna ze souboru
    auto refBuilder = Gtk::Builder::create();
    refBuilder->add_from_file("basic.glade");
    // získání hlavního okna z návrhu
    auto pDialog = refBuilder->get_widget<Gtk::Dialog>("DialogBasic");
    // přidání okna z návrhu do aplikace
    app->add_window(*pDialog);
    // zobrazení
    pDialog->show();
}
```

Obr. 25 Ukázka vytvoření okna aplikace pomocí GTK a C++ (vlastní)

```
<interface>
  <object class="GtkDialog" id="dialog1">
    <child internal-child="content_area">
      <object class="GtkBox" id="vbox1">
        <child internal-child="action_area">
          <object class="GtkBox" id="hbuttonbox1">
            <child>
              <object class="GtkButton" id="ok_button">
                <property name="label" translatable="yes">_Ok</property>
                <property name="use-underline">True</property>
                <signal name="clicked" handler="ok_button_clicked"/>
              </object>
            </child>
          </object>
        </child>
      </object>
    </child>
  </object>
</interface>
```

Obr. 26 Ukázka definování okna aplikace pomocí GTK Builder XML syntaxe (vlastní)

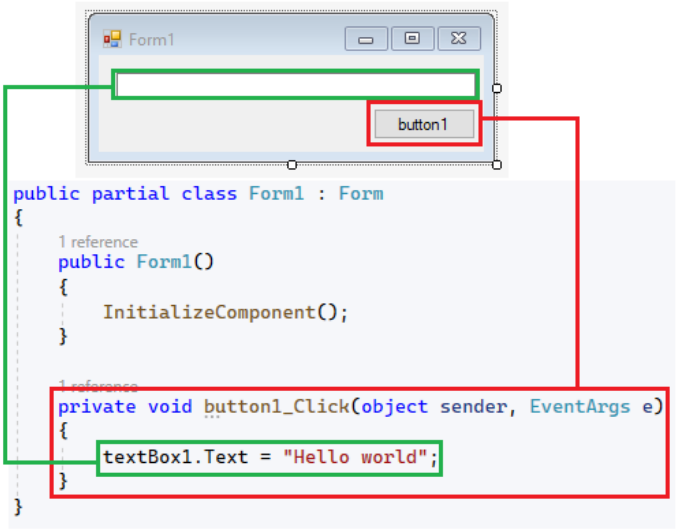
Při případném použití GTK jako implementačního nástroje pro FE aplikace CertiChain by bylo důležité zvolit správný jazyk vytváření aplikační logiky. FE aplikace CertiChain musí komunikovat s BE, proto je nezbytné, aby vybraný programovací jazyk umožňoval odesílat http požadavky a uměl provádět převádění objektů do JSON. Všechny výše zmiňované programovací jazyky podporují tyto úkony, ale proces odeslání http požadavku či deserializace JSON do objektu pomocí Pythonu bude nejspíše jednodušší na implementaci než pomocí C++.

3.2.1.3 Vývoj aplikace pro Windows

Pro implementaci aplikace, která má cílit pouze na operační systém Windows se nabízí použít C# .NET. Z grafických frameworků je možno vybírat z vícera možností, kdy lze použít Windows Forms (název je často zkracován na WinForms), WPF (Windows Presentation Foundation), UWP (Universal Windows Platform) anebo nejaktuálnější technologii pro vývoj aplikací pro Windows, framework WinUI.

WinForms je nejstarší framework od Microsoftu pomocí kterého lze vytvářet aplikace s grafickým systémem pro platformu Windows pomocí C# (vydán začátkem roku 2002). Jednotlivé prvky, ze kterých lze vytvořit grafické rozhraní aplikace (např. tlačítko, vstupní textové pole), vznikly ve WinForms jako nadstavba přímo nad prvky Windows User Interface. Vzhled prvků tedy závisí na operačním systému, na kterém je aplikace spouštěna. Např. aplikace vytvořená pomocí WinForms má styl prvků podle nového Windows 11 stylu a vypadají rozdílně, než když se spustí stejná aplikace např. na Windows 10.

Tvorba grafického rozhraní pomocí WinForms lze vytvářet buď programově, kdy se vytváří instance třídy daného prvku, nebo lze využít grafického návrháře, který zdrojový kód generuje automaticky na pozadí. Aplikační logika se poté zapisuje do obsluhy událostí jednotlivých uživatelských prvků. WinForms je vhodný na vytváření aplikací, které mají jednoduché grafické rozhraní s několika uživatelskými prvky. Vytvářet komplikované obrazovky se spoustou prvků s cílem udělat aplikaci co nejvíce dynamickou, může být pomocí WinForms zdlouhavé a komplikované, pro tento typ aplikací je vhodnější použít frameworky WPF, UWP či WinUI.



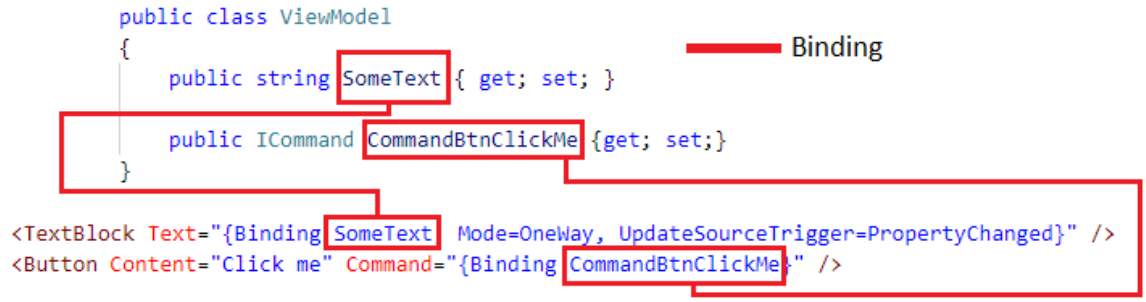
Obr. 27 Ukázka návrhu a psaní aplikační logiky pomocí WinForms (vlastní)

WPF Microsoft vydal ve 3. kvartálu roku 2006 jako alternativu pro již existující WinForms. WPF mělo za cíl vyřešit hlavní problém technologie WinForms, kterou byla nemožnost oddělit programovou logiku od tvorby grafického rozhraní. S WPF přišel nový způsob vytváření grafického rozhraní pomocí jazyka XAML, který je založen na formátu XML. Jazyk XAML obsahuje sadu klíčových prvků, kdy každý klíčový prvek je pro určitý uživatelský prvek. Grafické rozhraní aplikace lze také vytvářet pomocí návrháře podobně jako ve WinForms, ale přímé psaní XAML kódu je pro zkušené vývojáře efektivnější způsob vytváření GUI.

WPF prvky nejsou přímou nadstavbou prvků Windows User Interface, prvky mají výchozí vzhled jako prvky pro Windows 7, ale umožňuje všechny prvky libovolně stylovat. Ve výchozím nastavení tedy prvky vypadají stejně na všech systémech Windows, ale pomocí stylů lze prvky libovolně pře stylovat dle potřeb. Existují různé doplňkové knihovny, ve kterých

jsou vytvořeny styly prvků podle nového FluentUI, či MaterialDesign. Některé knihovy se styly jsou k dispozici zdarma, ale jsou i placené, které poskytují lepší podporu a použitelnost.

Logika programu lze poté psát do tzv. Code Behind (kódu na pozadí). Pomocí tohoto způsobu psaní logiky ale nelze dosáhnout oddělení programové logiky od obrazovky v pravém smyslu slova. Nejlepší možnost jak od sebe oddělit programovou logiku a samotnou obrazovku, je použití návrhového vzoru MVVM (Model-View-ViewModel), kdy obrazovka spadá do části View, ViewModel se stará o programovou logiku a modelové třídy slouží pro vytvoření struktury např. tabulek. Propojení mezi View a ViewModelem je poté docíleno pomocí tzv. Bindingu, kdy se ve View nastaví tzv. DataContext, což bude instance daného ViewModelu a ve View se poté nabindují jednotlivé vlastnosti třídy do cílových grafických prvků.



Obr. 28 Ukázka WPF kódu, MVVM a Bindingu (vlastní)

S příchodem Windows 8 přišel Microsoft s novým stylem pro operační systém a s ním přišel i nový framework MDL (Microsoft Design Language), tento počin ale skončil velikým neúspěchem a dnes jej prakticky nikdo nepoužívá. Microsoft tedy tuhle cestu opustil a s Windows 10 přišel nový framework pro tvorbu aplikací s grafických rozhraním, který se jmenoval UWP. Tento framework cílí na vytváření aplikací pro desktopové zařízení s Windows 10 a 11, za dob Windows 10 Mobile i na mobilní zařízení, dále pro zařízení Xbox a HoloLens. [24]

Vývoj pomocí UWP je velmi podobný jako pomocí WPF, k tvorbě obrazovek se používá také jazyk XAML. I názvy jednotlivých prvků jsou velmi obdobné jako ty ve WPF a při programování aplikační logiky lze také využít techniky kódu na pozadí nebo architektonického vzoru MVVM. Nově Microsoft vyvíjí i nový framework WinUI, který má v roce 2022 3 verze. Sada prvků a stylů dostupných WinUI 2 je nyní výchozí styl prvků pro platformu

UWP a nynější verze WinUI obsahuje prvky a styly, které jsou používány ve Windows 11. Díky WinUI 3 se můžou vyvíjet aplikace s nejaktuálnějšími styly i pro Windows 10.

3.2.2 Multiplatformní technologie

Další možnost, jak implementovat FE aplikace CertiChain, je použití multiplatformních frameworků. Díky použití tohoto řešení lze vytvořit pouze jeden projekt, který poběží na všech platformách. Multiplatformní aplikace lze vytvořit např. pomocí programovacích jazyků C# .NET, Java, C++ a jejich frameworků.

3.2.2.1 Java

Programovací jazyk Java je multiplatformní a lze používat na všech hlavních desktopových operačních systémech, na které aplikace CertiChain primárně cílí. Na spouštění aplikací vytvořených pomocí programovacího jazyka Java si stačí nainstalovat Java Runtime Environment (JRE). Za dobu existence programovacího jazyka Java je k dispozici několik grafických frameworků, jako např. AWT, Swing nebo JavaFX. Nejmladší z těchto frameworků, který má nahradit starší GUI frameworky je poslední zmíněný framework JavaFX. Pomocí JavaFX lze dokonce vytvářet aplikace i na mobilní platformu.

JavaFX obsahuje širokou škálu dostupných prvků pomocí kterých lze vytvářet aplikace s grafickým rozhraním. Pro tvorbu jednotlivých obrazovek se používá speciálně vytvořený jazyk FXML, který je založen na XML. K dispozici je interaktivní aplikace SceneBuilder pro vytváření obrazovek, kdy obrazovky vývojář vytváří přetahováním jednotlivých prvků do aplikace. [25]

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
  xmlns:fx="http://javafx.com/fxml" fx:controller="hellofx.FXMLDocumentController">
  <children>
      <Button layoutX="126" layoutY="90" text="Click Me!"
      onAction="#handleButtonAction" fx:id="button" />
      <Label layoutX="126" layoutY="120" minHeight="16"
      minWidth="69" fx:id="label" />
  </children>
</AnchorPane>
```

Obr. 29 Ukázka jazyka FXML (vlastní)

Aplikační logika se poté vytváří do tzv. Controllerů (kontrolérů). Kontroléry jsou klasické Java třídy, které se nastaví jako zdrojový kontrolér v FXML souboru pomocí atributu *fx:controller* (lze vidět na obrázku 46). Na jednotlivé prvky, se kterými je potřeba pracovat v kontroléru je možno získat referenci na prvek vytvořením proměnné s cílovým typem, kdy

jméno proměnné se musí shodovat s id daného prvku v FXML souboru. Dále je potřeba před proměnnou vložit anotaci *@FXML*, který se pokusí najít daný prvek FXML souboru. [25]

```
public class FXMLDocumentController {
    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }
}
```

Obr. 30 Ukázka kontroléru pro aplikační logiku obrazovky v JavaFX (vlastní)

3.2.2.2 C# .NET

C# má k dispozici frameworky pomocí kterých lze vytvářet multiplatformní aplikace pomocí jednoho projektového řešení. K dispozici jsou frameworky třetích stran, např. UnoPlatform nebo Avalonia. Pod záštitou Microsoftu vzniká framework .NET MAUI, který je v poměrně rané fázi vývoje a Microsoft na něm usilovně pracuje. V budoucnu by .NET MAUI mohl být velice dobrý nástroj na tvorbu multiplatformních aplikací pomocí C# .NET. MAUI ale nepodporuje platformu Linux, z tohoto důvodu by nebylo možné jej použít pro implementaci FE aplikace CertiChain. Avalonia a PlatformUno umožňují vytvořit aplikace na všechny primárně cílené platformy.

Framework PlatformUNO je založen na frameworku UWP. V základu je tedy vývoj aplikace takřka totožný s vývojem UWP aplikace. Pro návrh obrazovek se využívá XAML jazyk a aplikační logika může být implementována v kódu na pozadí, nebo lze využít návrhového vzoru MVVM a DataBindingu. Pro vývoj aplikací pomocí UnoPlatform je potřeba si stáhnout projektové šablony řešení (solution) do Visual Studia či JetBrains.Rider. Vytvořené řešení (solution) se skládá z více projektů (projects), kdy každá část cílí na danou platformu. V hlavní části se implementuje logika aplikace a zbylé projekty, pro dané platformy se na tuto část odkazují. Projekt na Windows je poté klasická WinUI aplikace pro platformu Linux se používá nástroj Skia a pro Android, iOS a MacOS se využívá nástroj Xamarin. [26]

Avalonia je dalším frameworkem, který lze využít pro multiplatformní vytváření aplikací. V Avalonia se pro vývoj obrazovek také využívá jazyk XAML, ale s názvy prvků z frameworku WPF. Při vývoji aplikační logiky lze využít techniky kód na pozadí návrhový vzor MVVM. Aplikace lze poté sestavit pro operační systémy Windows, Linux i macOS a lze

cílit i na mobilní zařízení s Android a iOS. Po sestavení aplikace vypadá aplikace stejně na všech platformách, kdy výchozí styl uživatelských prvků je na základě UWP stylů. [27]

3.2.2.3 C++

Pomocí programovacího jazyka C++ lze také vytvářet multiplatformní aplikace. Na tvorbu okenních aplikací je vhodný framework Qt, pomocí kterého lze vytvářet grafické aplikace na desktopové zařízení i na zařízení mobilní. Na desktopové zařízení je možno využít dvou přístupů na tvorbu aplikací, které Qt umožňuje.

První přístup je použití Qt Widgets, které umožňují vytvářet aplikace s grafickým rozhraním pro Windows, Linux i macOS. Aplikaci lze vytvářet na kterékoli platformě a pro použití na cílové platformě je potřeba projekt na ní sestavit, tedy aby aplikace běžela na Windows, je potřeba projekt sestavit na systému Windows. Aplikace bude mít poté vždy vzhled podle výchozího stylu daného operačního systému. Vzhled jednotlivých obrazovek se vytváří pomocí interaktivního návrháře, který je součástí vývojového prostředí Qt. Programová logika se poté vytváří uvnitř C++ tříd, kdy ke každé obrazovce je vždy vytvořena jedna třída. [28]

Druhá možnost, jak vytvořit multiplatformní aplikaci pomocí Qt, je využití Qt Quick. Qt Quick je sada technologií, která umožňuje vytvářet aplikace s dynamickým grafickým rozhraním a pomocí Qt Quick lze vyvíjet aplikace jak na desktopové i mobilní zařízení. Qt Quick obsahuje speciální jazyk QML, který slouží pro definování vzhledu uživatelského rozhraní, kdy jednotlivé obrazovky se vytvářejí do souborů s příponou qml. Aplikační logiku lze poté psát přímo do qml souboru pomocí jazyka JavaScript, nebo lze využít i jazyka C++, kdy lze vytvořit C++ třídu, která se poté naimportuje do qml souboru a vytvoří se její instance, a poté s ní je možno dále pracovat v qml souboru. Tento princip vývoje aplikace lze přirovnat k návrhovému vzoru MVVM, kdy třídu C++ je možno přirovnat ke ViewModelu, kdy v této třídě bude implementována aplikační logika dané obrazovky a qml soubor by bylo View, ve kterém se vytvoří instance C++ třídy a u jednotlivých akcí (např. stisknutí tlačítka) se zavolá příslušná funkce z C++ třídy. [29]

3.2.3 Technologie pro vytváření webových aplikací

V kapitole 3.1 Možnosti implementace BE byly rozebrány jednotlivé technologie jen pro vytvoření BE části aplikace. Programovací jazyky C# .Net a PHP Laravel nabízejí nástroje pro vývoj klientských částí webových aplikací a lze tak vytvořit celou aplikaci pomocí

jednoho projektového řešení. Tento způsob řešení má obrovskou výhodu, protože lze vytvořit celou aplikaci pomocí jedné technologie.

3.2.3.1 C# .NET

Při použití jazyka C# .Net a Asp.Net jako nástroj pro vytvoření BE části vyvíjené aplikace je vhodné pro FE využít např. framework Blazor, který cílí na vytváření interaktivních klientských webových aplikací. Blazor je pro .net vývojáře vhodný použít, protože místo programování logiky webové aplikace pomocí Javascriptu lze použít C#. Další výhoda je, že lze vytvořit sdílenou část celé aplikace, kterou je možno použít na FE i BE aplikace, a tím zjednodušit vytvářený kód. Např. komunikace mezi FE a BE, kdy se pomocí http(s) komunikace posílají data v JSON. Třídy, které slouží pro převod JSON na objekt a naopak, je vhodné umístit do sdílené části celé aplikace a tento objekt může poté využít FE i BE část. Blazor také obsahuje sadu komponent a umožňuje také vytvořit své vlastní komponenty.

Asp .Net nabízí užitečné knihovny, které usnadňují implementaci webových aplikací, např. knihovna pro usnadnění oddělení obrazovek aplikace jen pro přihlášené uživatele či pro uživatele s určitou rolí atd. Asp.Net a Blazor nabízí velmi propracované šablony projektů, kdy je možno si vygenerovat projekt, ve kterém je již předpřipravena autentizace v aplikaci.

3.2.3.2 PHP Laravel

Další z dostupných technologií z kapitoly 4.1, který umožňuje vytvořit Full Stack webovou aplikaci pomocí jednoho projektového souboru a jedné technologie je framework Laravel, který využívá jazyk PHP.

Laravel využívá šablonový engine Blade. Blade lze používat při vytváření dané stránky v html kódu. Pomocí něj lze vytvářet různé sekce, které se zobrazí např. pokud je splněná určitá podmínka. Dále pomocí Blade lze efektivně zobrazovat data získané z BE. Blade také obsahuje šablony pro vytváření sekcí stránek, které si může zobrazit jen přihlášený uživatel. Výhodou Blade je také možnost vkládat do html kódu klasický PHP kód.

3.2.3.3 Další frameworky pro implementaci FE webových aplikací

Pro vytváření uživatelské části webové aplikace je dnes dostupná spousta frameworků, která je založena na TypeScriptu či JavaScritpu. Mezi nejznámější a nejpoužívanější frameworky patří např. Angular, React nebo Vue.js. Pokud vývojář nechce používat žádný framework, ničemu nebrání vytvořit aplikaci jen pomocí html a Javascriptu. V dnešní době se velká

většina vývojářů spíše uchyluje cestou používání frameworků, které vývoj webové aplikace usnadňují a zároveň urychlují.

Angular je platforma pro vyvíjení webových aplikací, která je založena na jazyku TypeScript. Angular obsahuje knihovny pro vytváření škálovatelných aplikací, knihovny pro vytváření směrování v aplikací, tvorbu formulářů, knihovny pro komunikaci mezi FE a BE, obsahuje nástroje pro vytváření animací, nástroje pro vytvoření progresivních webových aplikací. [30]

React a Vue.js jsou frameworky, které se soustředí na usnadnění vývoje interaktivních a dynamických webových aplikací. Oba frameworky jsou založeny na Javascriptu a pro vytváření webových UI využívají tvorby pomocí komponent.

II. PRAKTICKÁ ČÁST

4 NÁVRH APLIKACE CERTICHAIN

Hlavním cílem aplikace CertiChain je spojit použití digitálního podepisování pomocí certifikátů s technologií průmyslového blockchainu a pokusit se tak vyřešit problém omezených platností digitálních podpisů.

Jedním z řešení, jak vyřešit platnost digitálního podpisu je použití průmyslového blockchainu. Princip blockchain systémů je navržen tak, aby šlo soubory ověřit po dobu, dokud daná blockchain databáze běží alespoň na jednom uzlu. Průmyslový blockchain lze tedy při prodloužení platností digitálního podpisu tím, že podepsaná data (nebo jejich otisk) se vloží společně s otiskem certifikátu do systému blockchain. Takto podepsaná data lze poté ověřovat klasickým způsobem jako se data ověřují při digitálním podepisování a po vypršení platnosti podpisu lze dat ověřovat pomocí blockchain systému. Pokud bude na základě ověřovaných dat bude nalezen záznam v blockchainu, lze mezi smluvenou skupinou lidí tento podpis stále považovat za platný.

Tato kapitola bude pojednávat o základních funkcích aplikace, prvním návrhu datových entit a o wireframech základních obrazovek, které by aplikace měla obsahovat.

4.1 Základní funkcionality aplikace CertiChain

Funkcionální a nefunkcionální požadavky se zaměřují na to, co by měl výsledný systém nebo aplikace měla umět. Základní funkcionální požadavky pro aplikaci CertiChain jsou:

- Přístup pouze pro autentizované uživatele
- Uživatel může vytvářet podepisovací profily
- Uživatel může digitálně podepisovat data pomocí svého certifikátu
- Uživatel může podepsaná data uložit do systému Blockchain
- Uživatel může ověřit digitální podpis tradičním způsobem ověření digitálního podpisu
- Uživatel může ověřit podepsaná data v systému blockchain
- Uživatel si může prohlížet záznamy uložené v systému blockchain

4.1.1 Přístup pouze pro autentizované uživatele

Při návrhu jednotlivých funkcionalit aplikace bylo navrhnuto několik prvků, které jsou vázány čistě na jednoho uživatele (např. podepisovací profily, záznamy v blockchain) a tyto údaje je potřeba vyhledat v databázi, aby je měl daný uživatel k dispozici.

Proto bylo při návrhu přikloněno k využití uživatelských účtů s tím, že všechny funkce aplikace budou dostupné jen pro registrované uživatele. Zakomponováním uživatelských účtů lze poté na každého uživatele jednoduše vázat jednotlivé entity, které se v aplikaci budou vyskytovat a budou vázány na dané uživatele. Použití uživatelských účtů může být výhodou také v budoucnosti, pokud by měly být služby aplikace CertiChain např. zpoplatněny.

Při registraci uživatel zadá svou emailovou adresu a heslo, kterým se do aplikace bude přihlašovat. Uživateli bude umožněno heslo si resetovat, pokud jej zapomene.

Přihlášený uživatel si bude moci potvrdit svou emailovou adresu, změnit heslo, případně svůj účet smazat (deaktivovat). Při smazání účtů dojde ke smazání jeho dat z databáze.

4.1.2 Uživatel může vytvářet podepisovací profily

Aby při vytváření digitálních podpisů nemusel uživatel stále dokola vyplňovat všechna povinná nastavení pro vytvoření digitálního podpisu, byl vytvořen pojem podepisovací profil.

Podepisovací profil je entita, ve které jsou uloženy jednotlivé položky, které jsou potřeba k vytvoření jednotlivých typů digitálních podpisů.

4.1.3 Uživatel může digitálně podepisovat data pomocí svého certifikátu

Vytváření digitálních podpisů je hlavní důvod vývoje aplikace CertiChain. Uživatel nebude moci podepsat přímo vložená data (např. text napsaný v textovém poli), ale bude moci pouze vytvářet digitální podpisy k souborům.

Aby uživatel mohl podepisovat co největší škálu různých typů dat, aplikace bude umožňovat vytvářet digitální podpisy pomoci více druhů technických norem digitálních podpisů. Pro PDF dokumenty bude primárně sloužit podpis PAdES, pro XML dokumenty podpis XAdES. Při použití externího podpisu může uživatel podepsat libovolná data a ještě bude k dispozici podpis CAdES, pomocí kterého lze také podepsat libovolná data.

Při vytváření podpisu uživatel vždy vybírá soubor, který se bude digitálně podepisovat, poté si vybere certifikát, který může být komerční nebo kvalifikovaný. Po podepsání dat si je bude moci uložit zpět do počítače.

4.1.4 Uživatel může podepsaná data uložit do systému Blockchain

Přidanou hodnotou aplikace CertiChain bude při vytváření digitálního podpisu možnost uložení podepsaných dat do systému blockchain. Při ukládání záznamu do systému blockchain

se bude nahrávat podepsaný soubor (případně externí podpis a data ke kterým byl vytvořen) a otisk podepisovacího certifikátu.

4.1.5 Uživatel může ověřit digitální podpis tradičním způsobem ověření digitálního podpisu

Uživatel si bude moci ověřit jakákoli podepsaná data klasickým způsobem, jak to umožňují normy CAdES, XAdES a PAdES. Pro ověření podpisu uživatel vybere podepsaná data, případně data, ke kterým byl vytvořen externí podpis a samotný externí podpis. Po ověření podpisu bude uživateli zobrazen výsledek ověření s informacemi o digitálních podpisech, které se v dokumentech nacházejí.

Při ověřování digitálních podpisů není nutné ověřovat pouze podpisy, které byly vytvořeny v aplikaci CertiChain. CertiChain bude moci ověřit digitální podpisy vytvořené v libovolných aplikacích, které vytvářejí podpisy pomocí norem XAdES, CAdES a PAdES.

4.1.6 Uživatel může ověřit podepsaná data v systému blockchain

Uživatel může podepsaná data ověřit, zda jsou dostupná v systému blockchain. Pro ověření dostupnosti podepsaných dat v systému je potřeba vybrat podepsaná data (případně externí podpis a podepisovaná data) a navíc ještě musí zvolit i certifikát, kterým byla data podepsána. Na základě těchto dat se poté vyhledá záznam v blockchain. Uživatel nebude muset nutně ověřovat jen svá podepsaná data v systému blockchain, ale může ověřovat i podepsaná data ostatních uživatelů aplikace CertiChain, pokud bude mít k dispozici všechna potřebná data.

4.1.7 Uživatel si může prohlížet záznamy uložené v systému blockchain

Pokud uživatel při vytváření digitálního podpisu bude vytvářet záznam také v systému blockchain, id vytvořeného záznamu se bude ukládat do databáze aplikace. Uživatel si tak bude moci prohlížet historii záznamů, které do systému blockchain přidal.

4.2 Návrh datových entit

Z popisu základních funkcionalit aplikace lze vyčíst datové entity a vztahy mezi nimi. Základní entity, které se budou vyskytovat v aplikaci, jsou entity uživatele, podepisovací profil obecný, podepisovací profil specializovaný (celkově tři entity – entita pro podpis PAdES,

CAdES a XAdES), dále entita s informacemi o podepisujícím a entita pro informace o záznamu v sytému Blockchain.

4.2.1 Entita uživatele (User)

Hlavní entitou, na kterou budou vázány i další entity, bude entita uživatel. Entita „uživatel“ bude obsahovat následující atributy:

- ID
- Emailovou adresu
- Otisk hesla
- Příznak, zda má uživatel potvrzenou emailovou adresu
- Příznak, zda je účet aktivní nebo „smazán“

4.2.2 Entita obecného podepisovacího profilu (ProfileGeneral)

Další entita bude potřeba pro uložení podepisovacích profilů. Problémem při ukládání profilů je podpora více druhů podpisů. PAdES, XAdES i CAdES mají několik společných nastavení, ale některé jsou rozdílné. Proto by bylo nevhodné, kdyby profily byly uloženy jen v jedné mohutné entitě, kde by většina atributů entit nebyla využita. Tento přístup by také porušil normální formy databází.

Proto budou podepisovací profily rozděleny na více entit. Základní entitou bude entita „Obecný profil“, která bude uchovávat pouze základní společné atributy jednotlivých profilů. Obecný profil bude obsahovat následující atributy:

- Id
- Jméno profilu
- Hashovací algoritmus použitý při vytváření podpisu
- Id uživatele, pod kterého podepisovací profil spadá. Vztah mezi entitami bude 1:N (jeden uživatel může mít n profilů)
- Typ podepisovacího profilu (XAdES, CAdES, PAdES)
- Id na entitu, která obsahuje další nastavení potřebná pro vytvoření digitálního podpisu (vztah mezi entitami 1:1)
- Id na entitu o informacích o podepisujícím, která je nepovinná (vztah mezi entitami 1:1)

4.2.3 Entita Profil CAdES (ProfileCades)

Pro uložení specifických parametrů profilu CAdES bude sloužit entita „Profil CAdES“, která se bude skládat z následujících atributů:

- Id profilu CAdES
- Typ podpisu CAdES (B, T)
- Příznak, zda bude podpis CAdES součástí podepisovaných dat nebo bude digitální podpis vytvořen jako soubor zvlášť

4.2.4 Entita Profil XAdES (ProfileXades)

Pro uložení specifických parametrů profilu XAdES bude vytvořena entita „Profil XAdES“, která bude obsahovat následující atributy:

- Id profilu XAdES
- Typ podpisu XAdES (B, T)
- Příznak, zda bude podpis XAdES součástí podepisovaných dat nebo bude digitální podpis vytvořen jako nový soubor

4.2.5 Entita Profil PAdES (ProfilePAdES)

Posledním podporovaným podepisovacím profilem v aplikaci bude podpis PAdES, jehož specifické parametry budou uloženy v entitě „Profil PAdES“. Tato entita bude nejobsáhlejší entitou ze skupiny entit pro nastavení podepisovacích profilů z důvodu, že podpisu PAdES lze nastavit i parametry viditelného podpisu. Entita Profil PAdES bude obsahovat následující parametry:

- Id profilu PAdES
- Typ podpisu PAdES
- Příznak, zda se jedná o viditelný podpis PAdES
- Typ viditelného podpisu (zdroj viditelného podpisu – obrázek, text nebo text a obrázek)
- Obrázek podpisu (data ve formátu pole bytů), který se použije ve viditelném podpisu, pokud má typ viditelného podpisu hodnotu obrázek nebo text a obrázek
- Horizontální umístění viditelného podpisu v PDF dokumentu
- Vertikální umístění viditelného umístění podpisu v PDF dokumentu
- Šířka viditelného podpisu

- Výška viditelného podpisu
- Stránka dokumentu, kde bude viditelný podpis umístěn
- Text, který bude umístěn ve viditelném podpisu, pokud je zvolen typ viditelného podpisu obrázek nebo text a obrázek
- Barva pozadí, kterou bude mít viditelný podpis
- Jméno podepisujícího
- Místo podepsání
- Důvod podepsání

4.2.6 Entita Informace o podepisujícím (SignerInfo)

Poslední entita, která ještě souvisí s podepisovacím profilem, je „Informace o podepisujícím“. Jedná se o dodatečné informace, které lze přidat k digitálnímu podpisu, jejich přítomnost však není povinná. Entita „Informace o podepisujícím obsahuje následující atributy:

- Stát
- Kraj
- Poštovní směrovací číslo
- Město
- Role podepisujícího (např autor, vlastník apod.)
- Příznak, že při podepisování dokumentu se potvrzuje původ dokumentu
- Příznak, který při podepisování dokumentu se potvrzuje vytvoření dokumentu

4.2.7 Entita záznamu v blockchain (BlockchainRecord)

Poslední entitou, která se bude vyskytovat v aplikaci CertiChain, bude entita „Záznam v Blockchain“. Ta bude uchovávat základní informace o záznamu v systému blockchain, který tam daný uživatel přidal. Entita se bude skládat z následujících atributů:

- Id záznamu
- Id uživatele, který vytvořil záznam
- Datum vytvoření záznamu
- Otisk záznamu
- Id záznamu v systému Blockchain

Na obrázku 31 lze vidět vytvořené schéma entit s jednotlivými vztahy.

Obr. 31 Schéma datových entit aplikace (vlastní)

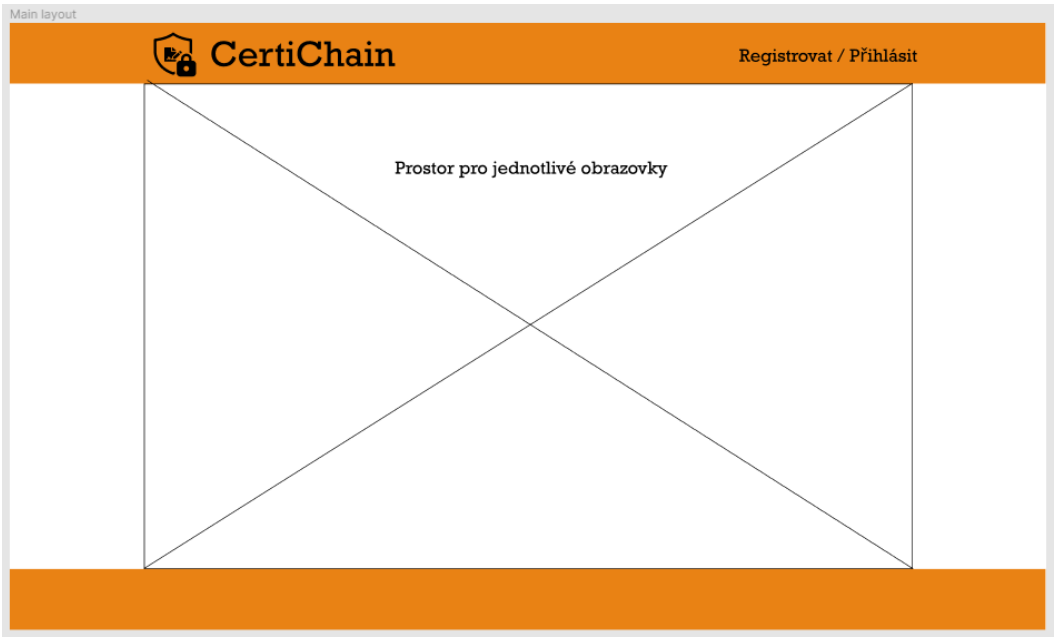
4.3 Návrh obrazovek aplikace

Z popsaných funkcionalit aplikace v kapitole 4.2 lze navrhnout i seznam obrazovek aplikace, pomocí kterých bude možno jednotlivé funkcionality aplikace vykonávat. Návrh obrazovek aplikace bude zpracován pomocí nástroje Figma, který je k dispozici zdarma, a který slouží na vytváření náhledu obrazovek aplikací. Figma obsahuje spoustu designerských pomůcek a dá se pomoci tohoto nástroje připravit vzhled celých aplikací včetně designu.

4.3.1 Základní layout aplikace

Základní layout aplikace bude obsahovat záhlaví, ve kterém bude jméno aplikace s vytvořeným logem, které bude umístěno v levé části záhlaví. V pravé části záhlaví poté bude navigační menu, které bude měnit na základě podmínky, zda je uživatel přihlášen. Pokud v aplikaci nebude přihlášen žádný uživatel, v pravé části záhlaví budou tlačítka pro přihlášení, případně registraci do aplikace. Pokud uživatel bude přihlášen, bude v pravé části záhlaví bude mít své uživatelské jméno (emailová adresa) a po rozkliknutí se mu zobrazí výběrové menu pro navigaci po aplikaci anebo se bude moci odhlásit z aplikace.

Většinu prostoru okna aplikace bude obsahovat prostor pro jednotlivé obrazovky aplikace. Vespod bude prostor pro patičku aplikace, která bude sloužit pro zobrazení informací o autorovi aplikace, copyright, případně kontakt na autora aplikace.



Obr. 32 Základní layout aplikace (vlastní)

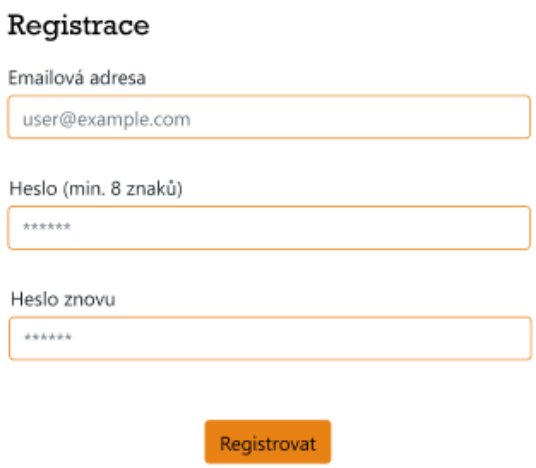
4.3.2 Uvítací obrazovka

Aplikace bude mít uvítací obrazovku, která se zobrazí jako první, když uživatel zapne aplikaci a není registrován. Na uvítací obrazovce budou popsány informace o aplikaci a jakou přidanou hodnotu přináší používání aplikace, a dále informace, jak aplikaci začít používat.

Obr. 33 Návrh uvítací obrazovky (vlastní)

4.3.3 Obrazovka pro registraci nového uživatele do aplikace

Pro registraci nového uživatele do aplikace bude vytvořena speciální obrazovka, která se bude skládat z polí pro zadání povinných údajů k registraci. Mezi potřebné údaje patří emailová adresa a heslo. Pro ujištění správnosti zadaného hesla, bude v obrazovce ještě pole pro zopakování hesla.



Obr. 34 Návrh obrazovky Registrace (vlastní)

4.3.4 Obrazovka pro přihlášení uživatele

Obrazovka pro přihlášení uživatele bude obsahovat pole pro zadání emailové adresy a hesla. Pro zahájení procesu přihlášení bude dostupné tlačítko „Přihlásit“. Pokud uživatel své heslo zapomene, bude si jej moci z resetovat pomocí tlačítka „Zapomněl jsem heslo“.

Obr. 35 Návrh obrazovky Přihlášení (vlastní)

4.3.5 Domovská obrazovka přihlášeného uživatele

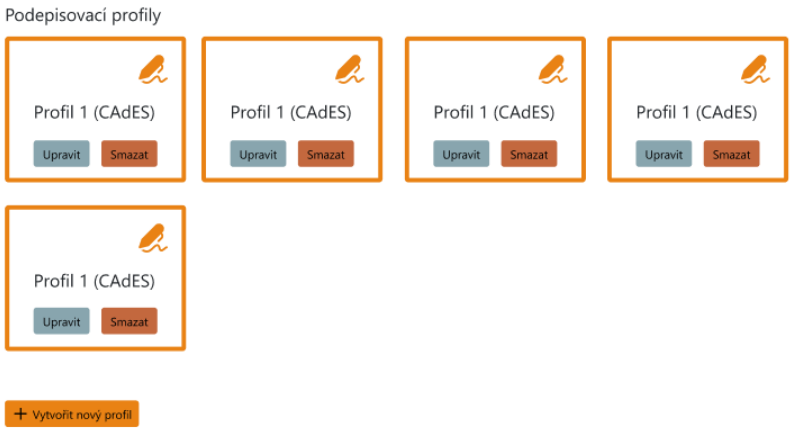
Domovská stránka uživatele bude „rozcestník“ pro uživatele k jednotlivým funkcionalitám, které aplikace bude obsahovat. Z domovské obrazovky se bude moci přesunout na stránky pro:

- Vytváření digitálního podpisu
- Ověřování digitálního podpisu
- Správu podepisovacích profilů
- Prohlížení záznamů v systému Blockchain
- Správu uživatelského účtu

Obr. 36 Návrh domovské obrazovky pro přihlášeného uživatele (vlastní)

4.3.6 Obrazovka pro správu podepisovacích profilů

Na obrazovce pro správu podepisovacích profilů budou k dispozici všechny podepisovací profily, které si přihlášený uživatel vytvořil. Na této stránce bude moci přejít na úpravu vybraného profilu nebo bude moci vybraný profil smazat. V obrazovce pro správu podepisovacích profilů bude také moci zahájit proces vytvoření nového podepisovacího profilu.



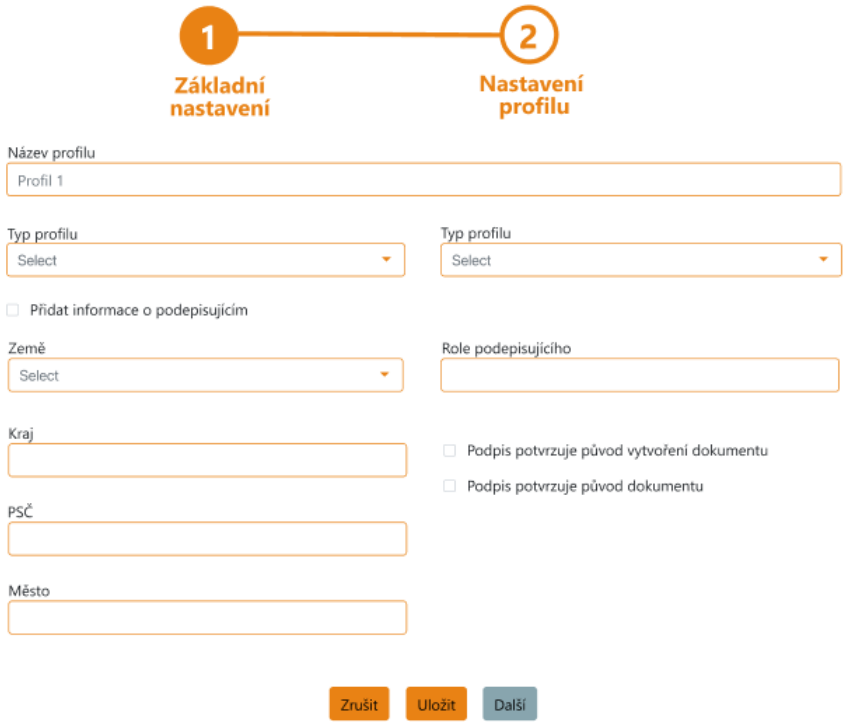
Obr. 37 Návrh obrazovky pro správu podepisovacích profilů (vlastní)

4.3.7 Obrazovka pro editaci tvorbu podepisovacího profilu

Proces vytvoření a editace existujícího profilu lze spojit do jedné obrazovky, protože vzhled obou stránek je totožný. Jediný rozdíl mezi editací existujícího profilu a vytvořením nového profilu je načtení existujících dat z databáze při editaci profilu.

Proces editace / vytvoření profilu je obsáhlý a na základě možnosti výběru z více typů podpisů i proměnlivý. Proto bude proces editace / tvorby profilu rozdělen na dvě části, mezi kterými se bude uživatel moci přepínat. První část se bude týkat základního nastavení profilu a část druhá nastaveními daného podepisovacího profilu. Mezi jednotlivými částmi obrazovky bude možno přepínat pomocí tlačítek „Předchozí“ a „Další“. Změny při vytváření bude možno uložit pomocí tlačítka „Uložit“ nebo bude možno proces zrušit pomocí tlačítka „Zrušit“.

V části pro základní nastavení profilu budou pole pro zadání názvu profilu, typ hashovacího algoritmu, typ podepisovacího algoritmu. Dále zde bude volitelné pole pro možnost zobrazení polí pro přidání informací o podepisujícím. Informace o podepisujícím bude obsahovat pole pro Výběr země, zadání kraje, PSČ, města, roli podepisujícího a zaškrtávací pole pro potvrzení o původu nebo vytvoření dokumentu. Část Základní nastavení tedy pokryje datové entity Obecný profil a Informace o podepisujícím.



Obr. 38 Návrh základního nastavení editace / vytvoření profilu (vlastní)

Druhá část obrazovky pro editaci / vytvoření nového profilu bude záviset na výběru typu podpisu (CAdES, XAdES, PAdES) z části Základní nastavení.

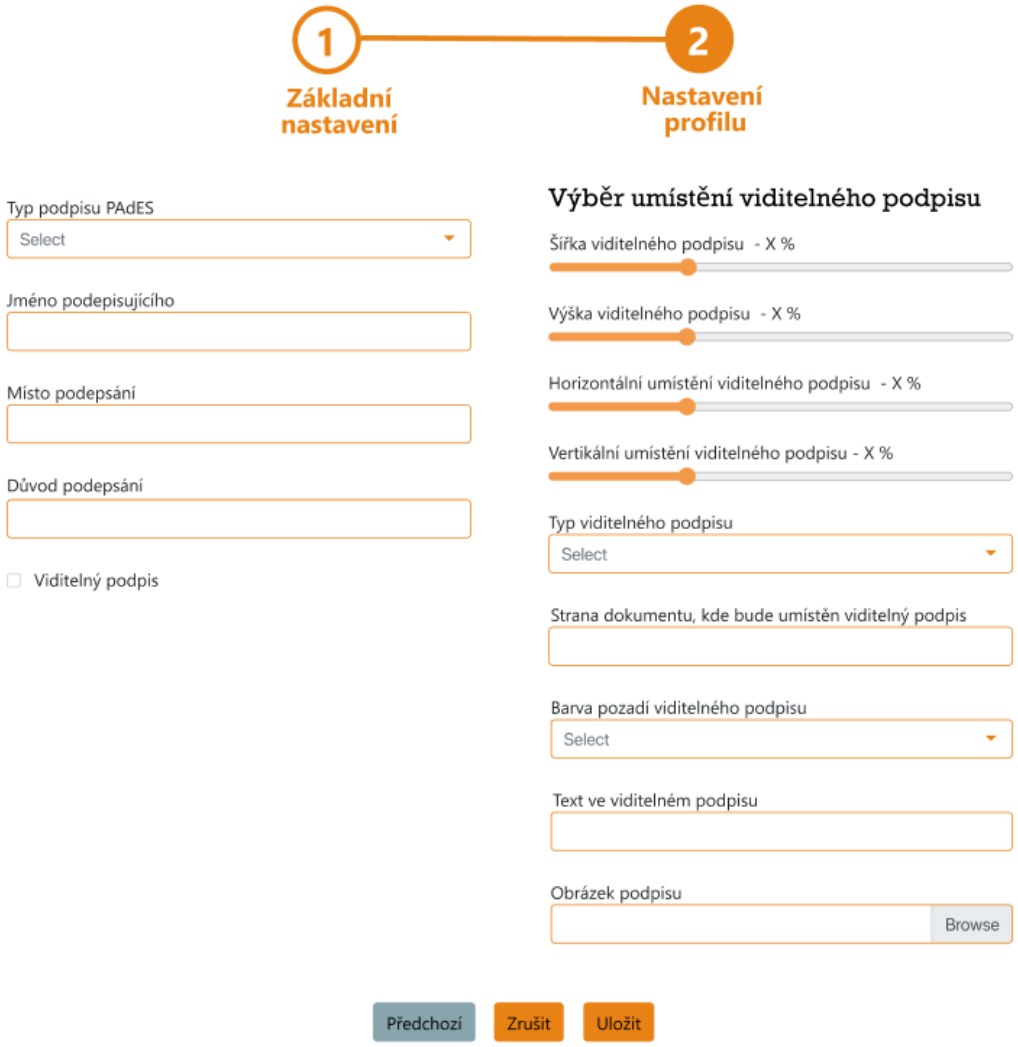
Podpis CAdES a XAdES mají obdobné nastavení, které se týká nastavení typu profilu a umístění podpisu (uvnitř podepisovaných dat nebo externí podpis). V nastavení tedy bude pole pro výběr typu podpisu CAdES / XAdES a výběr umístění podpisu.

Obr. 39 Návrh části obrazovky nastavení profilu CAdES (vlastní)

Obr. 40 Návrh části obrazovky profilu XAdES (vlastní)

Nastavení parametrů podpisu PAdES má nejvíce parametrů, z důvodu možnosti vložení viditelného objektu do PDF dokumentu, který znázorňuje digitální podpis. Pro nastavení základních parametrů podpisu PAdES budou potřeba pole pro výběr typu podpisu PAdES, dále textové pole pro zadání jména podepisujícího, místa podepsání a důvodu podepsání. Tato pole však budou nepovinná. Dále zde bude pole pro možnost vložení viditelného podpisu.

Při zvolení volby použití viditelného podpisu se uživateli zobrazí dodatečné pole pro parametrizaci vzhledu viditelného podpisu – tedy možnosti pro nastavení velikosti podpisu, umístění podpisu, volba strany dokumentu, na kterou se podpis umístí a nastavení vzhledu viditelného podpisu. Viditelný podpis se může skládat z textu, obrázku nebo z textu i obrázku. Pro nastavení těchto parametrů bude k dispozici výběrové pole pro typ viditelného podpisu, textové pole pro zadání textu, a pole pro výběr obrázkového souboru podpisu. Tato pole se budou dynamicky zobrazovat na základě vybraného typu viditelného podpisu.

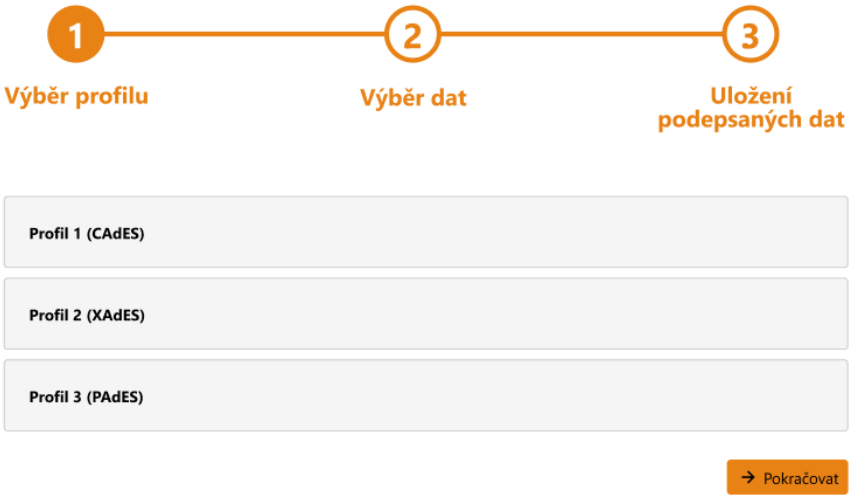


Obr. 41 Návrh části obrazovky pro parametrizaci podpisu PAdES (vlastní)

4.3.8 Obrazovka pro vytvoření digitálního podpisu

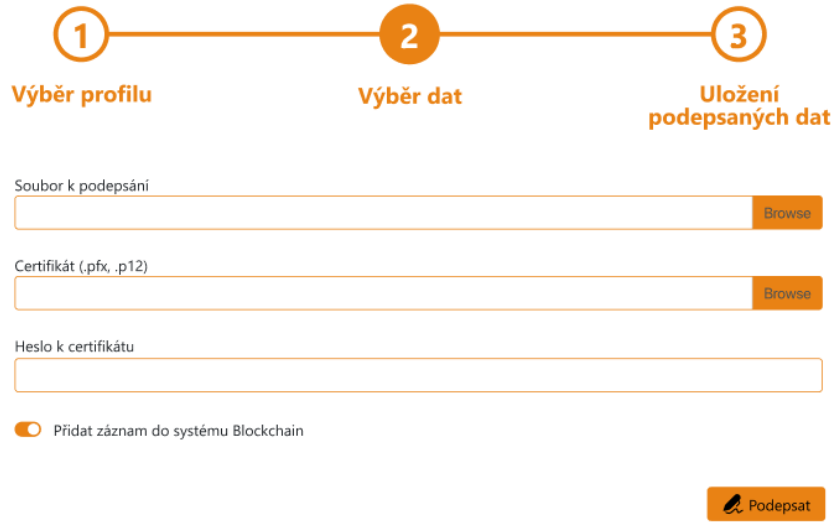
Proces vytvoření digitálního podpisu lze rozdělit na tři samostatné kroky – výběr podepisovacího profilu, výběr dat potřebných pro vytvoření digitálního podpisu a uložení podepsaných dat, případně externího podpisu. Podle těchto kroků bude také rozdělena obrazovka pro vytvoření digitálního podpisu.

První část obrazovky pro vytvoření digitálního podpisu bude volba podepisovacího profilu. Budou zde zobrazeny všechny podepisovací profily uživatele v seznamu, kdy uživatel si bude moci jeden z profilů vybrat. Po výběru profilu bude umožněno uživateli přejít na další krok pomocí tlačítka „Pokračovat“. Pokud by uživatel neměl dostupný žádný podepisovací profil, bude na tuhle situaci upozorněn a bude moci přejít na obrazovku pro vytvoření nového podepisovacího profilu.



Obr. 42 Návrh části pro výběr podepisovacího profilu při vytváření podpisu (vlastní)

Druhá část obrazovky bude zaměřená na výběr dat potřebných pro vytvoření digitálního podpisu, tedy data k podepsání a certifikát, pomocí kterého bude podpis vytvořen. Pro oba vstupy bude použito pole pro výběr souboru a textové pole pro zadání hesla k certifikátu. V obrazovce bude ještě k dispozici zaškrtávací pole pro aktivaci / deaktivaci možnosti vytvořit záznam v systému blockchain při úspěšném podepsání dat.



Obr. 43 Návrh části pro výběr dat při vytváření podpisu (vlastní)

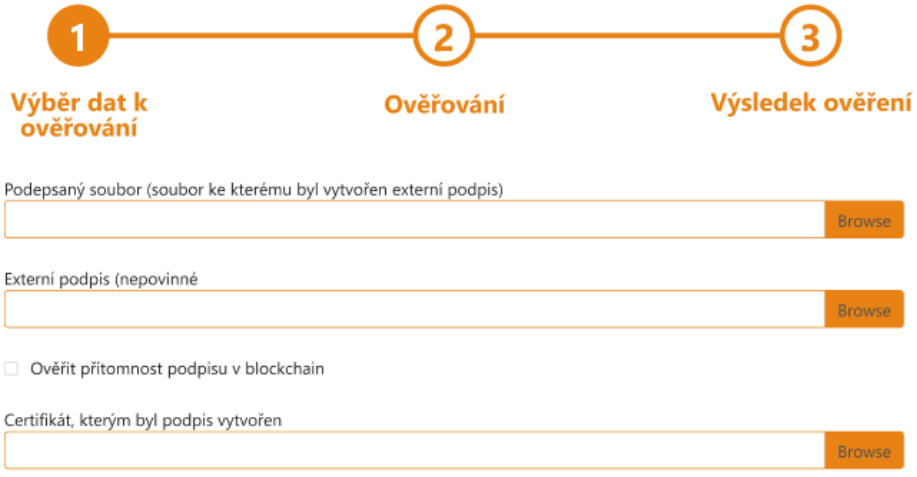
V poslední části obrazovky bude zobrazen výsledek procesu vytvoření digitálního podpisu. Pokud se digitální podpis úspěšně vytvoří, bude k dispozici tlačítko pro jeho stažení. Pokud bude zvolena možnost vytvoření záznamu v systému blockchain, bude v této části obrazovky také výsledek operace vložení do systému blockchain.

Obr. 44 Návrh části obrazovky pro zobrazení výsledku procesu podepisování (vlastní)

4.3.9 Obrazovka pro ověření digitálního podpisu

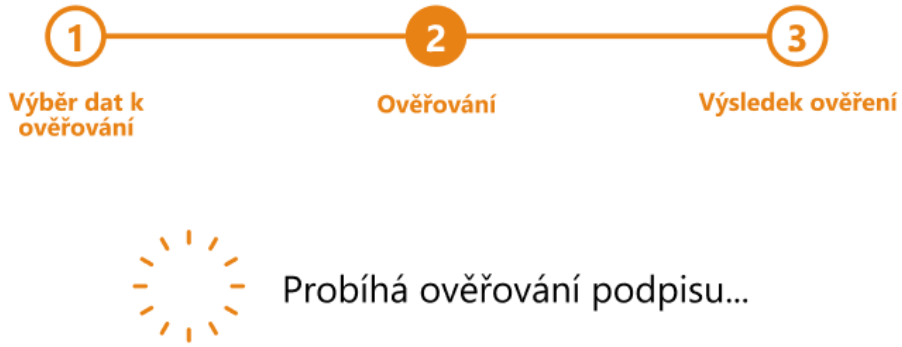
Proces ověření digitálního podpisu se skládá ze dvou hlavních částí. První částí je výběr dat, která se budou ověřovat, a druhá část je zobrazení výsledku ověření. Ověření digitálního podpisu je akce, která potenciálně může trvat delší dobu, proto je důležité uživateli zobrazit, že ověřování stále probíhá, a aby nepřecházel do jiné obrazovky aplikace. Proto se bude obrazovka pro ověření digitálního podpisu skládat ze tří částí.

První část obrazovky bude sloužit pro výběr dat k ověření. Pro výběr dat bude k dispozici pole pro výběr podepsaného souboru (případně pro výběr dat, ke kterým byl vytvořen externí podpis) a pole pro výběr souboru externího podpisu. Pokud bude uživatel chtít vyhledat záznam podpisu v systému blockchain, bude si tuto možnost moci zvolit pomocí zaškrtávacího pole, dále bude muset vybrat certifikát, pomocí kterého byl digitální podpis vytvořen.



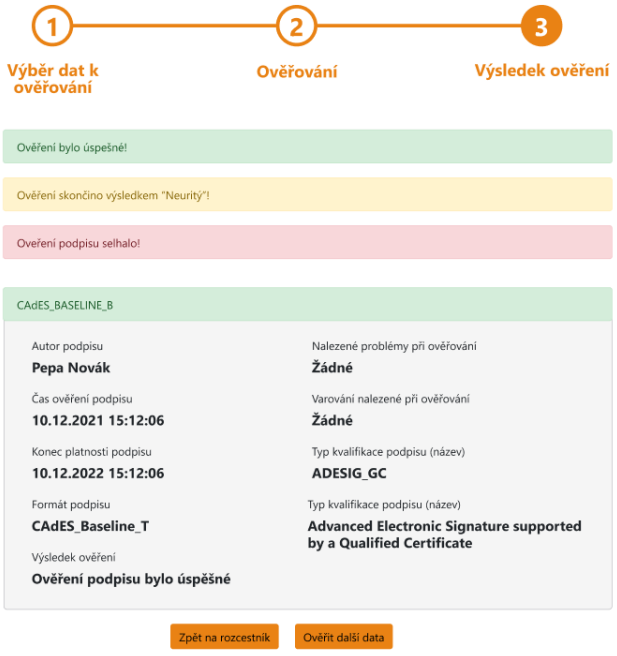
Obr. 45 Návrh části obrazovky pro výběr dat k ověření (vlastní)

Druhá část obrazovky bude zobrazena po dobu průběhu ověřování digitálního podpisu a bude obsahovat animaci načítacího kolečka a text s informací, že proces ověření podpisu stále probíhá.



Obr. 46 Návrh části obrazovky pro vyčkání na dokončení ověření (vlastní)

Poslední část obrazovky bude zobrazovat výsledek ověření digitálních podpisů. Dokument může obsahovat vícero digitálních podpisů, proto budou ve výsledku o ověření zobrazeny výsledky ověření všech podpisů. Na úvodu bude dostupná informace o celkovém výsledku ověření – zda byly všechny podpisy úspěšně ověřeny, nebo zda nastaly při ověřování nějaké problémy. Pokud si uživatel při ověřování vybere možnost kontroly záznamu v systému blockchain, obrazovka bude také obsahovat informaci o dostupnosti záznamu v blockchain.

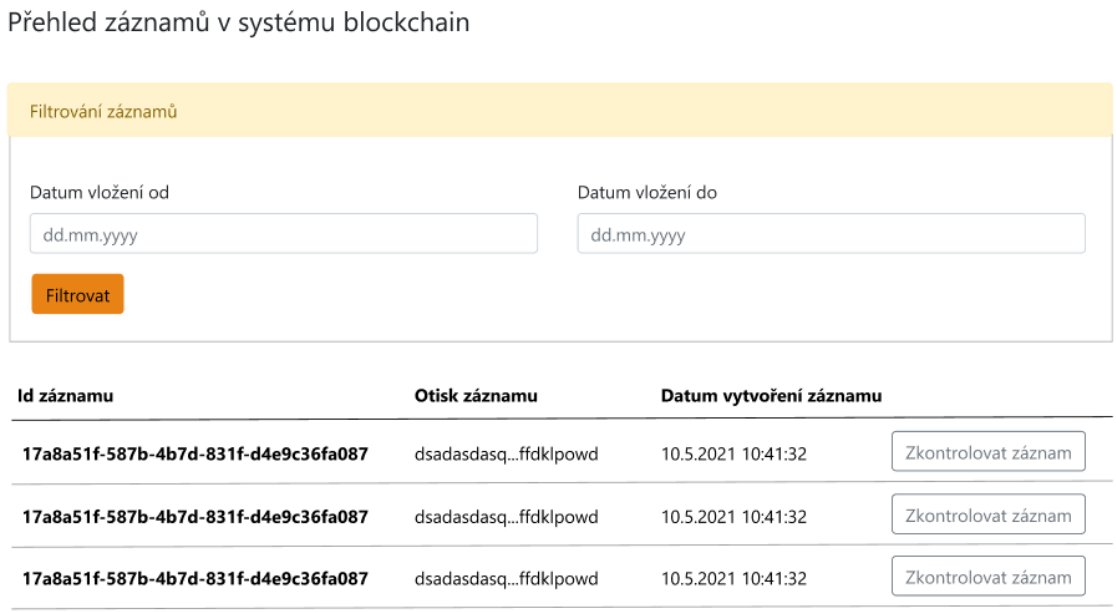


Obr. 47 Návrh části obrazovky pro zobrazení výsledku ověření (vlastní)

4.3.10 Obrazovka pro prohlížení záznamů v blockchain

Při přidání nového záznamu do systému blockchain, bude aplikace vždy do databáze aplikace ukládat základní informace o záznamu, aby se uživatel mohl kdykoli podívat na přehled

jeho záznamů, které do blockchain přidal. Pro zobrazení přehledu bude vytvořena speciální obrazovka, která dané záznamy zobrazí v tabulce. Záznamů v blockchain může uživatel po čase mít spoustu, proto bude pro jednoduší vyhledávání obrazovka obsahovat i sekci pro filtrování. Filtrovat bude možno pomocí datumu od a datumu do.



Obr. 48 Návrh obrazovky pro přehled záznamů v blockchain (vlastní)

5 TECHNOLOGIE POUŽITÉ PŘI VÝJOJI APLIKACE

V této kapitole budou stručně rozebrány zvolené technologie, pomocí kterých aplikace bude vyvíjena a z jakého důvodu byly vybrány. Dále zde budou uvedeny jednotlivé knihovny, které budou při vývoji využity, jak pro zjednodušení implementace či zlepšení vzhledu webové aplikace. Poté zde budou uvedeny technologie, které budou potřeba, aby běžely vedle vyvíjené aplikace. Jedná se zejména o databázové systémy, kdy první bude klasická relační databáze pro ukládání klasických entit, které jsou zmíněny v kapitole 4.2 a druhá bude samotný blockchain.

5.1 Volba technologií pro tvorbu BE a FE

První návrh realizace projektu CetiChain bylo vytvořit desktopovou aplikaci pomocí programovacího jazyka Java či C# .NET, kdy by většina všech procesů byla prováděna přímo v FE části aplikace a BE část by sloužila pouze pro správu uživatelů a ukládání podepisovacích profilů a komunikaci s blockchain. Na FE by poté probíhala samotná tvorba digitálních podpisů pomocí certifikátu a komunikace s blockchain databází.

Po konzultacích s vedoucím práce byl mírně změněn náhled na prvotní návrh projektu a změnil se i směr implementace, kdy pro jednoduchost používání a instalace bylo dohodnuto, že projekt CertiChain vznikne jako webová aplikace, ve které bude vytváření digitálního podpisu implementováno na klientské části webové aplikace a BE část by se starala o správu uživatelů a ukládání profilů a přehledu záznamů v systému blockchain.

Výběr technologie pro implementaci aplikace závisel na dostupností existujících knihoven, které implementovaly problematiku vytváření a ověřování digitálních podpisů pomocí digitálních certifikátů. Z nalezených řešení se vybízelo použití programovacího jazyka Java nebo C# .Net, kdy oba tyto jazyky mají k dispozici knihovny pro vytváření a ověřování digitálních podpisů, které jsou k dispozici zdarma. Z těchto dvou technologií byl vybrán programovací jazyk C# .Net a framework Blazor. Důvodem pro výběr této volby byly zkušenosti práce s těmito nástroji a zjednodušené správy uživatelů, kterou Blazor implementuje za vývojáře. Pro vytváření a ověřování digitálních podpisů byla vybrána knihovna .NET DIGITAL SIGNATURE LIBRARY.

Při zahájení implementace aplikace pomocí této prvotní volby ale přišly v rané fázi vývoje kritické problémy, které tuto volbu technologie finálně znemožnily použít. Překážkou se stala knihovna pro vytváření digitálních a ověřování podpisů, a nakonec byl potřeba změnit

i postup řešení, který byl navržen v první verzi analýzy. Při testovacím použití knihovny byl nejprve zjištěn fakt, že nebude možné vytvořit digitální podpis na FE části aplikace, aniž by se k aplikaci vytvořil doplněk pro webové prohlížeče, který by proces vytvoření a ověření digitálního podpisu dokázal zajistit. Z tohoto důvodu musel byl změněn plán způsobu vytváření digitálního podepisování, kdy nově se o tuto funkcionalitu bude starat BE část aplikace. Po testování vytvoření digitálního podpisu na BE části aplikace byl zjištěn další problém, kdy knihovna .Net Digital Signature Library nefunguje správně v Asp.Net projektech a v důsledku tohoto problému nebylo možné použít .Net pro vývoj BE části aplikace.

Z důvodů uvedených výše byla nakonec navržena druhá verze řešení implementace projektu CertiChain, kdy z použití .Net se neupustilo úplně, ale .Net a Blazor bude použit pouze pro tvorbu FE části aplikace, a na BE část aplikace bude použit programovací jazyk Java a framework Spring. Pro vytváření a ověřování digitálních podpisů byla zvolena knihovna DSS (DIGITAL SIGNATURE SERVICE). Po analýze a otestování této varianty bylo potvrzeno, že lze na BE části vytvořit a ověřit digitální podpis.

Ve finální variantě výběru technologií pro implementaci aplikace byla tedy zvolena C# .Net Blazor pro implementaci FE části aplikace a Java Spring pro část backendovou. Tento způsob implementace způsobil pár komplikací. Za použití této varianty bylo potřeba samostatně naimplementovat více funkcionalit (např. správa uživatelů, nemožnost využití sdílených částí kódu, kterých by bylo možno využít).

5.1.1 Technologie použité při tvorbě BE

Pro vývoj BE části aplikace byl tedy nakonec zvolen programovací jazyk Java a framework Spring. Pro vytváření aplikace bylo využito vývojové prostředí IntelliJ Idea od společnosti JetBrains, kdy studenti, kteří vlastní průkaz ISIC, si můžou na stránkách JetBrains vytvořit účet, a pomocí ISIC karty si můžou aktivovat studentský balíček od společnosti JetBrains, ve kterém se nacházejí všechny aplikace, které JetBrains poskytuje.

Pro vytvoření základní struktury projektu byl použit nástroj Spring Boot Initializr, pomocí kterého si lze vytvořit šablonu projektu podle potřeb řešeného problému. Spring Boot Initializr obsahuje řadu zaškrtávacích polí, pomocí kterých si lze vytvořit základní strukturu projektu. Obsahuje nastavení, zda se bude jednat o full stack aplikaci či jen REST API, lze si vybrat typ databáze, se kterou bude aplikace komunikovat, možnost přidání balíčků pro zakomponování autentizace a správy uživatelů, přidání balíčků pro emailové API a spoustu dalšího.

Pro ladění BE aplikace bylo použito základní nastavení, které nabízí vývojové prostředí IntelliJ Idea, která Backendové aplikace spouští na lokálním serveru Apache Tomcat, který IntelliJ automaticky spustí a nakonfiguruje takovým způsobem, aby bylo možno aplikaci ladit.

Při vytváření aplikace pomocí programovacího jazyka si lze vybrat ze dvou sestavovacích nástrojů Apache Maven nebo Gradle. Pro vývoj aplikace CertiChain byl vybrán nástroj Apache Maven.

Při vývoji byly používány knihovny přímo od vývojářů frameworku Spring, knihovny vytvořené projektem Digital Signature Service a několik knihoven třetích stran, které urychlovaly psaní boiler plate kódu, kterého je v programovacím jazyce Java spousta.

Od tvůrců frameworku Spring byly použity následující knihovny:

- Spring Boot Starter Mail – knihovna obsahuje třídy pro zakomponování odesílání emailů v aplikaci. Tato knihovna v aplikaci bude využita např. pro zasílání potvrzovacích emailů o aktivaci či při resetování zapomenutého hesla.
- Spring Boot Starter Security – knihovna pro zabezpečování jednotlivých koncových bodů aplikace. Knihovna obsahuje sadu základních tříd, které implementují autentizaci v aplikaci, nebo si vývojář může vytvořit vlastní třídu, která tyto základní třídy rozšiřuje, a naimplementovat si vlastní způsob autentizace. Poté lze libovolně nastavovat, které koncové body aplikace mají podléhat autentizaci.
- Spring Boot Starter Web – základní knihovna frameworku Spring, která obsahuje třídy pro implementaci samotných koncových bodů webového API.
- Spring Boot Starter Data JPA – Knihovna obsahující třídy a rozhraní pro implementaci ORM komunikace s databází.
- Spring Boot Starter Data MongoDB – Knihovna obsahující třídy, pomocí kterých se lze připojit k MongoDB databází a umožňuje provádět různé operace nad databází.

Pro vytváření a ověřování digitálních podpisů se používají knihovny, které jsou součástí projektu DSS:

- DSS Document – knihovna, která slouží pro vytváření a ověřování podpisů PAdES, CAdES a XAdES.
- DSS Token – knihovna obsahující třídy pro práci s digitálním certifikátem, pomocí kterého se vytváří digitální podpis.

- DSS Utils Google Guava – tato knihovna je vyžadována knihovnami DSS Document, kdy při vytváření podpisů jsou používány třídy z této knihovny. Tato knihovna lze případně nahradit knihovnou DSS Utils Apache Commons.
- DSS Service – v této knihovně jsou dostupné třídy, pomocí kterých lze např. vytvářet časová razítka, kontrolovat status OCSP a další. Použití knihovny je vyžadováno při vytváření digitálního podpisu (např. pro vytvoření časového razítka k digitálnímu podpisu) i při ověřování digitálního podpisu (např. online kontrola statusu OCSP, kontrola CRL listů certifikačních autorit)
- DSS PAdeS PdfBox – knihovna je vyžadována knihovnou DSS PAdES při vytváření podpisu v PDF dokumentu
- DSS SPI – knihovna obsahující rozhraní a třídy pro komunikaci mezi lokálními knihovnami DSS a webovou částí projektu DSS

V aplikaci jsou poté ještě využity knihovny od dalších tvůrců:

- JJWT – knihovna, která usnadňuje zakomponování autentizace pomocí JWT tokenů. Tato knihovna obsahuje třídy a funkce pro vytvoření, ověření a získání informací z JWT tokenů
- MySQL Connector Java – JBDC ovladač pro databázi MySQL. Tato knihovna je potřeba, aby mohla správně fungovat a komunikovat knihovna Spring Boot Starter JPA s databází MySQL.
- Lombok – tato knihovna je velký pomocník co se týče psaní boiler plate kódu v programovacím jazyku Java. Při psaní kódu v Java by se mělo přistupovat k jednotlivým proměnným třídy pomocí metod, tzv. „Getterů“ a „Setterů“. Psaní těchto metod je ale velice obtěžující a zdržující. Knihovna Lombok obsahuje anotace, které tyto metody vygenerují za vývojáře.

```
// Varianta bez použití Lombok
public class Example {
    private String variable;

    public String getVariable() { return variable; }
    public String setVariable(String val) { variable = val; }
}

// Varianta s použitím Lombok
public class ExampleLombok {
    @Getter @Setter private String variable;
}
```

Obr. 49 Ukázka využití knihovny Lombok (vlastní)

- GSON – knihovna od společnosti Google, pomocí které lze převádět Java objekty do struktury JSON a naopak.

5.1.2 Technologie použité při tvorbě FE

Jak bylo zmíněno na začátku kapitoly 5.1, FE část aplikace bude implementována pomocí programovacího jazyka C# a frameworku Blazor. Pro vývoj aplikace bylo využito vývojové prostředí Rider od společnosti JetBrains.

Pro vývoj FE části aplikace byly také využity knihovny třetích stran, které jsou v .Net distribuovány pomocí tzv. NuGet balíčků:

- Blazorise – Blazorise nabízí sadu knihoven, které obsahují kompletní balíčky komponent pro vývoj a stylování webových stránek. Blazorise obsahuje spoustu komponent, které nahrazují klasické html prvky. Pomocí těchto komponent je možno vytvářet kompletní obrazovky. Hlavním cílem Blazorise je zjednodušení stylování aplikace, kdy při použití Blazorise lze nastylovat aplikaci pomocí C# kódu v sekci, kdy se spouští aplikace a poté díky použití komponent, které Blazorise obsahuje, se již vývojář nemusí o starat o další stylování, pouze si případně vybere cílovou barvu. Blazorise nabízí také různé základní druhy stylů, přičemž si lze vybrat ze stylů Bootstrap, Material Design, Bulma a AntDesign. Pro aplikaci CertiChain je vybrán styl Bootstrap.
- Blazored Local Storage – knihovna Local Storage obsahuje třídy pro práci s Local Storage v prohlížeči.
- Blazored Toast – Blazored Toast je knihovna, která obsahuje komponentu Toast. Tato komponenta je vhodná pro zobrazování různých informačních hlášek uživateli. Komponenta se vždy zobrazí v určené části obrazovky se zadanou hláškou a po definovaném čase zmizí nebo ji lze kliknutím myši ručně zavřít.
- BlazorPro SpinKit – obsahuje komponentu pro znázornění průběhu operace na pozadí.
- Newtonsoft Json – hojně používaná knihovna pro práci s JSON v .Net aplikacích. Newtonsoft nabízí elegantní řešení pro převod C# tříd do formátu JSON a naopak.
- Microsoft AspNetCore WebAssembly Authentication – knihovna pro zakomponování autentizace do klientské části aplikace. Obsahuje sadu komponent, pomocí kterých lze definovat jaký obsah se má zobrazit pro autentizované uživatele a jaký obsah

se může zobrazit pro uživatele nepřihlášené. Dále definuje sadu základních tříd, od kterých lze dále dědit a vytvořit si vlastní implementační třídu pro autentizaci.

5.2 Docker

BE část aplikace CertiChain se neobejde bez přístupu k databázím. Otázkou je, jak danou databázi naistalovat a spustit. Databázi je možno spustit přímo na lokálním stroji, kde je např. aplikace vyvíjena, nebo si lze pro databázi vytvořit virtuální zařízení, ve kterém bude databáze spuštěna. V dnešní době začíná získávat na popularitě spouštění služeb v izolovaných kontejnerech pomocí aplikace Docker.

Rozdíl mezi kontejnery a virtuálními stroji je ve způsobu virtualizace. Kontejner v Dockeru obsahuje pouze požadovanou aplikaci (např. databázi) a další potřebné soubory, které daná aplikace vyžaduje. Výkon a paměť získávají od hostujícího operačního systému, kdežto virtuální stroj je kompletní operační systém. Ve výsledku jsou kontejnery úspornější, co se týče využitého místa na disku a spotřebovaného výkonu procesoru a paměti RAM.

Další výhodou je rychlost vytvoření kontejneru v Dockeru. Při vytváření virtuálního stroje je potřeba projít celým procesem vytvoření virtuálního stroje – instalování operačního systému a pak cílové služby, která má na daném stroji běžet. Při vytváření kontejneru v Dockeru stačí stáhnout cílový obraz dané služby (např. databáze MySql) a spustit daný kontejner. Celý tento proces lze vykonat spuštěním jednoho příkazu v příkazové řádce operačního systému.

V kombinaci s Dockerem lze používat nástroj Docker-Compose, který slouží pro definování více kontejnerových aplikací za využití syntaxe YAML. Do YAML souboru stačí nadefinovat potřebné kontejnery (např. databáze a nástroj Adminer, pomocí kterého lze nahlížet do databáze) a poté lze oba kontejnery spustit najednou pomocí jednoho příkazu.

Docker je v diplomové práci využíván pro běh databázových systémů. Na serveru, kde bude nasazená aplikace CertiChain poběží několik Docker kontejnerů, kdy jeden kontejner bude pro relační databázi, která bude obsahovat tabulky dle entit popsaných v kapitole 4.2. Další kontejnery budou sloužit pro provoz blockchain databáze.

5.3 Databázové systémy

BE část aplikace bude komunikovat se dvěma databázovými systémy. První databázový systém bude relační databáze a druhý databázový systém bude blockchain databáze.

5.3.1 Relační databáze

Relační databáze bude sloužit pro ukládání aplikačních entit. K databázi bude přistupováno pomocí knihovny Hibernate a Data JPA, proto výběr databáze musí být zvolen ze seznamu dostupných JBDC ovladačů pro dané databázové systémy.

K dispozici jsou JBDC ovladače např. pro databáze MySQL, MSSQL, PostreSQL, Oracle a další relační databázové systémy. Pro diplomovou práci byla nakonec zvolena databáze MySQL. MySQL je k dispozici zdarma a zakomponování této databáze do projektu Java Spring není nijak komplikovaná.

5.3.2 Blockchain databáze

S výběrem databáze pro systém blockchain byla situace komplikovanější. Nabízely se dvě cesty kudy se vydat při řešení blockchain databáze. První možnost byla nalézt řešení, které by bylo možno nasadit na lokálním serveru např. pomocí Docker. Druhá možnost byla najít vhodného poskytovatele služby, kde by byl vytvořen uživatelský účet a záznamy by se přidávaly do již existující databáze. Při hledání dostupných řešení v obou variantách bylo nakonec přihlédnuto k první variantě.

5.3.2.1 BigChainDB

Při hledání vhodné blockchain databáze, která by běžela na lokálním serveru vedle BE aplikace CertiChain byla nalezena databáze BigChainDB, kterou je možno spustit i v Docker kontejneru a pro účely použití v CertiChain byla naprosto dostačující. Komunikace s BigChainDB probíhá pomocí REST API, kdy je k dispozici sada komunikací, pomocí kterých lze vkládat nové transakce, ověřovat transakce atd. Manuální implementace konzumace API BigChainDB by byla časově náročná. Autoři ale vytvořili knihovny v různých programovacích jazycích, které obsahují funkce pro obsloužení komunikace s BigChainDB. Autoři tyto knihovny nazývají ovladače. Jeden z ovladačů pro databázi BigChainDB je implementován i v jazyce Java, proto se nabízelo ovladač pro komunikaci s databází využít.

Při zakomponování ovladače do BE části CertiChain ale nastal kritický problém. Ovladač byl do projektu přidán pomocí nástroje Maven, ovladač se podařilo bez problému zakomponovat do BE a podařilo se aplikaci s použitím ovladače i úspěšně sestavit. Problém nastal v momentě spuštění Apache Tomcat a při nasazování aplikace na server, kdy při nasazování BE CertiChain se vždy sekla v určitém bodě a nikdy se proces nasazení neposunul dále. Při

hledání zdroje problému bylo zjištěno, že viníkem je právě ovladač pro databází BigChainDB.

Na popsaný problém se nabízely dva způsoby řešení. Buďto vytvořit vlastní ovladač pro komunikaci s databází BigChainDB nebo najít jinou blockchain databázi. Po analýze jednotlivých komunikací, které by bylo potřeba vytvořit ve vlastním ovladači, bylo usouzeno, že bude jednodušší se nejprve ohlédnout po jiné blockchain databázi. V případě, že se nepodaří najít alternativní řešení, přistoupilo by se na variantu implementace vlastního ovladače.

5.3.2.2 ProvenDB

Při hledání náhrady za blockchainovou databázi BigChainDB byla nalezena databáze ProvenDB. ProvenDB je možno využít zdarma dvěma způsoby. Prvním způsobem je vytvořit si uživatelský účet na jejich cloudu a získat přístup do databáze, kdy uživatelský účet s licencí zdarma má k dispozici 100 MB místa v cloudu. Druhý způsob je nasadit si ProvenDB databázi na svém lokálním zařízení, kdy databázi lze nasadit i pomocí Docker. Pro účely aplikace CertiChain byla využita varianta vlastního nasazení.

ProvenDB není čistě jen blockchainová databáze, ale je to kombinace dokumentové databáze MongoDB doplněná o funkcionality typické pro blockchain systémy. ProvenDB neodesílá v transakcích do systému blockchain přímo otisky přidávaných dat. ProvenDB je založena na principu verzování. Při každé manipulaci s databází se v ProvenDB inkrementuje hodnota verze databáze. Po inkrementaci verze databáze lze tuto verzi databáze uložit do systému blockchain. Přidávání dat do systému blockchain tedy není tak přímočaré jako např. v databázi BigChainDB, ale přesto tato technologie lze použít pro potřeby aplikace CertiChain.

Díky použití verzování a ukládání verzí databáze do systému blockchain umožňuje ProvenDB i aktualizaci či smazání (v ProvenDB se proces nazývá zapomenutí, záznam se totiž fyzicky úplně nesmaže, aby byl dodržen princip blockchain systémů) záznamu. Tyto funkcionality ale aplikace CertiChain využívat nebude a bude s ProvenDB pracovat s jako klasickou blockchain databází. Bude tedy do databáze přidávat pouze nové záznamy, kdy po přidání záznamu vždy nechá nahrát otisk verze do systému blockchain.

CertiChain bude s databází ProvenDB komunikovat následovně. Uživatel bude vytvářet digitální podpis, který chce uložit do systému blockchain. Po vytvoření digitálního podpisu se z podepsaných (v případě tvorby externího digitálního podpisu z podepisovaných) dat,

z podepisovacího certifikátu a případně ze samotného externího podpisu se vytvoří otisky. Tyto otisky se poté vloží do databáze ProvenDB. Přidání nového záznamu do databáze spustí vytvoření nové verze databáze ProvenDB a ihned po přidání záznamu a inkrementace verze databáze se tato verze odešle do systému Blockchain, kdy ProvenDB umožňuje komunikovat se dvěma blockchain systémy, a to buď Bitcoin či Ethereum.

Při ověřování podpisu v systému blockchain musí uživatel na vstup přidat všechna potřebná data k ověření, tedy podepsaná data (nebo data, ke kterým byl vytvářen externí podpis), podpisový certifikát a případně externí digitální popdis. Tato data se odešlou na BE CertiChain, vypočtou se z nich otisky a poté se spustí vyhledání dat v databázi a ověření cílové verze databáze v systému blockchain.

Databáze ProvenDB podporuje všechny základní příkazy, kterými disponuje databáze MongoDB a ještě přidává sadu speciálních příkazů, které slouží pro práci s verzemi databáze a pro práci se systémem Blockchain. Mezi speciální příkazy, které BE CertiChain bude často využívat lze zařadit následující příkazy:

- submitProof – tento příkaz vytvoří otisk z dané verze databáze a odešle transakci do systému Blockchain. Tento příkaz se bude volat po přidání nového záznamu o digitálním podpisu do databáze ProvenDB.
- getDocumentProof – příkaz, který ověří, zda je vyhledávaný záznam dostupný v databázi, zda je dokument dostupný v dané verzi databáze (typicky nejnovější verzi) a zda je záznam dané verze databáze stále platný.

6 IMPLEMENTACE APLIKACE CERTICHAIN

Po zpracování analýzy aplikace a vybrání vhodných technologií pro implementaci lze přejít na samotnou implementaci samotné aplikace. Vývoj aplikace lze rozdělit do následujících částí:

- Založení projektu a vytvoření základní struktury
- Úvodní konfigurace projektů
- Implementace správy uživatelů
- Implementace podepisovacích profilů
- Implementace vytvoření digitálního podpisu
- Implementace ověření digitálního podpisu
- Implementace přidání záznamu o podpisu do systému blockchain a jeho ověření
- Implementace přehledu záznamů v systému Blockchain

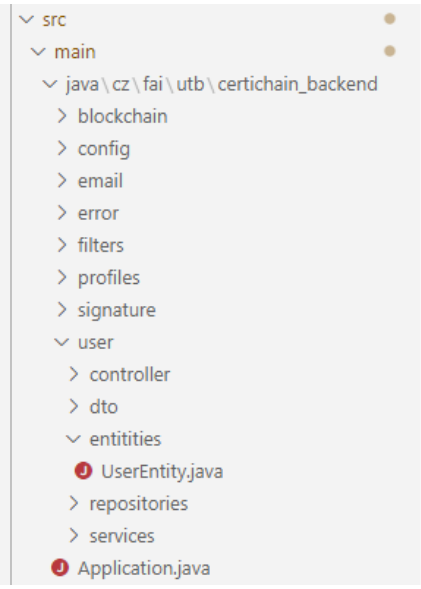
Všechny tyto části lze rozdělit ještě na dvě podčásti, a to na implementaci dané funkcionality na BE části aplikace a implementaci na FE části aplikace.

6.1 Založení projektu a vytvoření základní struktury

Při zahájení implementace aplikace byla nejprve vytvořena základní struktura projektů, do které se poté při implementaci jednotlivých funckionalit přidávaly jednotlivé třídy a rozhraní, které souvisely s danou částí aplikace. Struktura projektů je rozdílná z důvodu použití dvou různých technologií pro implementaci každé části aplikace. Celý projekt se nachází v kořenovém adresáři, který obsahuje tři další adresáře, ve kterých jsou umístěny projekty pro implementaci BE a FE části a poté adresář se sadami automatických testů.

6.1.1 Struktura projektu pro BE část aplikace

Struktura projektu BE aplikace je rozdělená podle funkcionalit, tedy např. pro správu uživatelů je dostupný adresář „user“, ve kterém je sada adresářů, které obsahují jednotlivé třídy, které plní určité povinnosti. V adresáři dané funkcionality jsou tedy adresáře pro uchování tříd kontroléru, databázových entit, JSON objektů, služeb, pomocných tříd a validátorů. Každý z těchto adresářů poté obsahuje třídy, které danou problematiku řeší. Obdobným způsobem jsou tvořeny adresáře i pro ostatní hlavní funkcionality aplikace. Cílem bylo vytvořit strukturu podle doporučení dokumentace Java Spring.

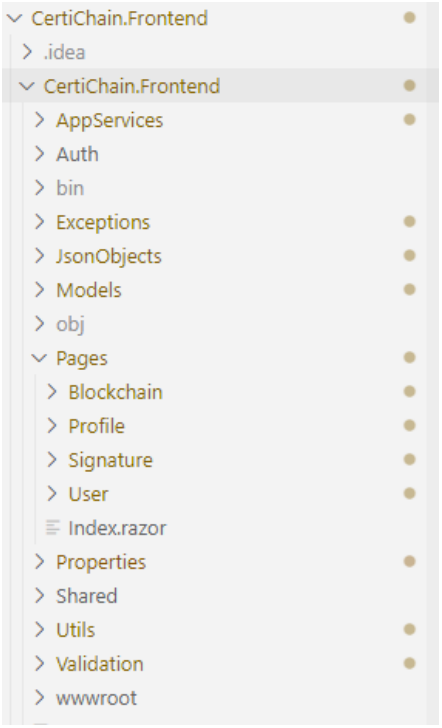


Obr. 50 Struktura projektu pro BE část aplikace (vlastní)

Výjimky v této struktuře činí adresáře config a filters. V adresáři config jsou umístěny všechny konfigurační třídy aplikace, které konfigurují např. databázi, zabezpečení nebo zasílaní emailů v aplikaci. V adresáři filters se poté nachází třídy pro implementaci vlastních aplikačních filtrů, co se týče zpracování http požadavků. O těchto filtrech se bude více psát v kapitole o implementaci správy uživatelů.

6.1.2 Struktura projektu pro FE část aplikace

Struktura projektu byla vytvořena podle doporučené struktury projektu pro Blazor, která funguje, dá se říct, obráceně vůči struktuře BE projektu. V kořenovém adresáři projektu se nachází adresáře pro jednotlivé části aplikace, tedy jeden adresář pro jednotlivé stránky, další adresář pro objekty, které slouží pro mapování JSON objektů, adresář pro validátory, aplikační služby atd. Tyto adresáře v sobě mají poté další adresáře, které v sobě uchovávají třídy pro specifickou funckionalitu aplikace, tedy např. aplikační služby pro správu uživatelů, správu podepisovacích profilů atd. Výslednou strukturu projektu lze vidět na obrázku 50.



Obr. 51 Struktura projektu FE části aplikace (vlastní)

6.2 Úvodní konfigurace projektu

Před zahájením implementace jednotlivých hlavních funckionalit aplikace bylo potřeba naimplementovat základní konfiguraci obou projektů. Na straně FE se jednalo zejména o zakomponování balíčků Blazorise pro stylování aplikace a na straně BE o konfiguraci databáze a dalších vyžadovaných služeb.

6.2.1 Konfigurace v projektu BE

Při úvodní konfiguraci projektu BE bylo potřeba nakonfigurovat přístup k databázi a vytvořit základní konfiguraci databáze. Konfigurace samotného přístupu knihoven k databázi je velice stručný, kdy stačí vytvořit třídu, u které se použije anotace, že se jedná o konfiguraci (*Configuration*) a poté anotaci *EnableTransactionManagement*, pro povolení správy transakcí s databází.

```
@Configuration
@EnableTransactionManagement
public class DbConfig {

}
```

Obr. 52 Konfigurační třída databáze (vlastní)

Pro dokončení konfigurace databáze bylo potřeba nastavit připojení k cílové databázi, tedy URL adresu k databázi, přihlašovací údaje a základní název třídy JBDC ovladače, který se má použít pro ORM přístup k databázi. Konfigurace těchto údajů je vložena v souboru application.properties, který je součástí zdrojů projektu.

```
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/CertiChain_db
spring.datasource.driver-class-name=org.mysql.jbdc.Driver
spring.datasource.username=[redacted]
spring.datasource.password=[redacted]
```

Obr. 53 Konfigurace připojení k databázi (vlastní)

Aplikace v procesu pro správu uživatelů vyžaduje zasílání emailů, proto bylo potřeba nakonfigurovat i údaje pro zasílání emailů. Pro zasílání emailů byla vytvořena speciální emailová adresa. Konfigurace třídy pro zasílání emailů se nakonfiguruje při spuštění aplikace a poté je instance třídy vložena do IoC kontejneru, ze kterého může být získána v ostatních třídách, kde je tato služba vyžadována.

```
@Bean
public JavaMailSender getJavaMailSender() {
    JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
    mailSender.setHost("smtp.gmail.com");
    mailSender.setPort(587);

    mailSender.setUsername([redacted]);
    mailSender.setPassword([redacted]);

    Properties props = mailSender.getJavaMailProperties();
    props.put("mail.transport.protocol", "smtp");
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.debug", "true");

    return mailSender;
}
```

Obr. 54 Konfigurace služby pro zasílání emailů (vlastní)

6.2.2 Konfigurace v projektu FE

V FE části aplikace bylo potřeba nejprve nakonfigurovat knihovnu Blazorise, která je použita pro vytváření uživatelského rozhraní. Konfigurace se skládá z několika kroků, kdy nejprve je třeba si vybrat požadovanou sadu stylů (např. Boostrap, Material, Ant atd.) a podle požadované sady stylů stáhnout cílový NuGet balíček. Pro aplikaci byla vybrána sada stylů Bootstrap. Dalším krokem je přidat odkazy na styly do souboru *index.html*, který se nachází v adresáři *wwwroot* FE projektu. Poté bylo ještě potřeba přidat namespace (jmenný prostor) knihovny Blazorise do souboru *_Imports.razor*, kde je vhodné přidávat jmenné prostory tříd, které jsou využívány ve více částech aplikací. Posledním krokem bylo přidat ve třídě Program.cs použití knihovny Blazorise a tím je konfigurace úspěšně dokončena.

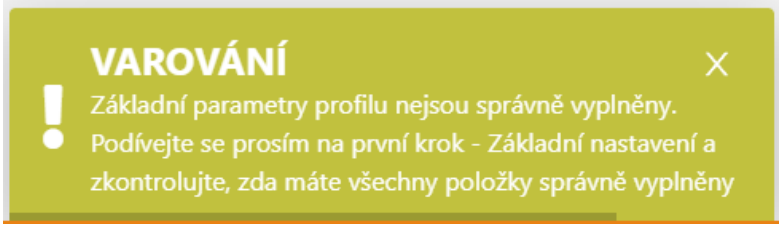
Po zakomponování knihovny bylo možno nastavit styl UI aplikace, tedy barvy jednotlivých komponent, barva textů pozadí atp. Právě v problematice nastavování UI je Blazorise velikým pomocníkem, kdy si všechny styly určitých prvků lze nastavit pomocí C# kódu vytvořením instance třídy *Theme*, kde si lze nastavit barvy textů, barvy pozadí aplikace, barvy pozadí prvků a mnoho dalšího. Při vytváření UI pomocí komponent z Blazorise, poté komponenty získávají jednotlivé parametry z instance *Theme*.

```
private Theme theme = new Theme
{
    IsRounded = true,
    ColorOptions = new ThemeColorOptions
    {
        Primary = CustomColors.PrimaryOrange,
        Secondary = CustomColors.Secondary,
        Danger = CustomColors.Danger,
        Dark = CustomColors.Dark,
        Success = CustomColors.Success,
        Info = CustomColors.Info,
        Light = CustomColors.Light,
        Warning = CustomColors.Warning
    },
    BackgroundOptions = new ThemeBackgroundOptions
    {
        Body = CustomColors.White,
        Primary = CustomColors.PrimaryOrange,
        Secondary = CustomColors.Secondary,
        Danger = CustomColors.Danger,
        Dark = CustomColors.Dark,
        Success = CustomColors.Success,
        Info = CustomColors.Info,
        Light = CustomColors.Light,
        Warning = CustomColors.Warning
    },
```

Obr. 55 Ukázka části definování stylů UI pomocí Blazorise (vlastní)

Na obrázku 54 je ukázána část konfigurace stylů vytvořením instance třídy *Theme*. Pro případnou jednoduchost změny jednotlivých barev byla v projektu ještě vytvořena třída *CustomColors*, kde jsou barvy definovány v hexadecimálním RGB formátu.

Dále bylo potřeba v části FE zakomponovat další UI doplňky. Aplikace využívá pro zobrazování různých informačních hlášek komponentu BlazoredToast. Při zakomponování tohoto doplňku bylo potřeba přidat instanci třídy komponenty do IoC kontejneru ve třídě Program.cs při spouštění aplikace, a nakonec přidat komponentu do hlavní obrazovky, kde je možné nakonfigurovat vzhled komponenty. Výslednou ukázku komponenty Toast lze vidět na obrázku 56.



Obr. 56 Ukázka komponenty Toast (vlastní)

V projektu jsou využívány ikony od poskytovatele FontAwesome. Blazorise poskytuje balíček s těmito ikonami, díky kterému lze k ikonám přistupovat pomocí výčtového typu místo jejich původního názvu. Pro přidání balíčku je potřeba jej přidat do IoC kontejneru při spouštění aplikace a přidat jmenný prostor balíčku do souboru *_Imports.razor.* Stejný proces ještě proběhl i u posledního UI balíčku, který obsahuje animace načítaní jménem SpinKit.

6.2.3 Příprava základního uživatelského rozhraní aplikace

V Blazor je již po vytvoření projektu pomocí základní projektové šablony rozdělena struktura tvorby jednotlivých stránek webové aplikace tak, že vývojář nemusí v každé obrazovce definovat celou stránku, ale její hlavní rozhraní se definuje jen jednou v souboru *MainLayout.razor* a ostatní stránky obsahují už pouze specifický obsah pro danou obrazovku. Ve finále po sestavení aplikace se poté celá obrazovka skládá z hlavního rozhraní a do ní se vždy vloží obsah specifické obrazovky.

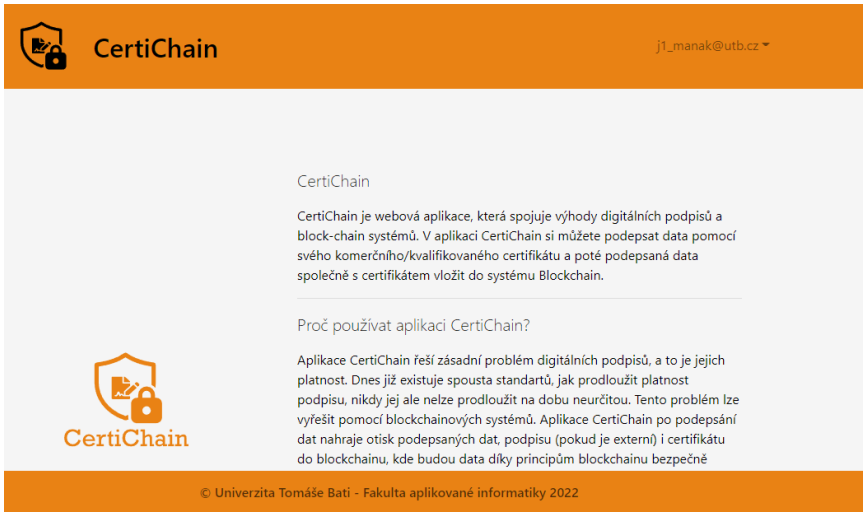
Před zahájením implementace jednotlivých obrazovek byla proto nejdříve vytvořena samotná základní struktura obrazovky. V návrhu obrazovek aplikace, v kapitole 3.3.1 je popsán základní vzhled aplikace. Základní rozhraní stránky se skládá ze tří hlavních částí. Hlavičky, která bude obsahovat logo s názvem aplikace, pomocí kterého se bude možno vždy vrátit hlavní Index obrazovku aplikace a poté menu, pomocí kterého lze přejít na jednotlivé funkcionality aplikace. Poté je hlavní část obrazovky, která je určena pro zobrazení dílčích obrazovek a vespod patička.

Výchozí rozložení hlavní obrazovky bylo vytvořeno pomocí komponent Blazorise, kdy pro jednotlivé části obrazovky jsou dostupné specifické komponenty. Celá obrazovka je obalená v komponentě *Layout*, ta obsahuje komponenty *LayoutHeader*, kde je dostupné menu a logo s názvem aplikace, poté je samotné tělo aplikace, které definuje komponenta *LayoutContent* a nakonec patička stránky, kterou definuje komponenta *LayoutFooter*. V hlavním rozhraní obrazovky je definována i komponenta pro zobrazování informačních hlášek *BlazoredToasts*.

```
7  <Layout Sider="false">
8      <LayoutHeader Background="Background.Primary">···
78     </LayoutHeader>
79     <LayoutContent Width="Width.Is100">
80         @Body
81     </LayoutContent>
82     <LayoutFooter Fixed="true" Background="Background.Primary">
83         <Div Flex="Flex.JustifyContent.Center">···
85         </Div>
86     </LayoutFooter>
87     <BlazoredToasts Position="Blazored.Toast.Configuration.ToastPosition.BottomRight"···
90 </Layout>
```

Obr. 57 Ukázka definování hlavního rozhraní UI aplikace (vlastní)

V obrázku 57 na řádku 80 lze vidět příkaz *@Body*, který slouží pro vložení těla dané obrazovky aplikace a ve výsledku je pak zobrazená kompletní obrazovka. Výsledné rozhraní po sestavení aplikace je zobrazeno na obrázku 58.



Obr. 58 Vzhled hlavního rozhraní po sestavení aplikace (vlastní)

6.3 Implementace správy uživatelů

Vzhledem k tomu, že všechny hlavní funkcionality jsou vázány na registrované uživatele, bylo potřeba v aplikaci nejdříve implementovat správu uživatelů. Zde se projevila první nevýhoda nutnosti použít pro tvorbu BE Java Spring místo .Net Asp.Net Core. Při použití .Net pro obě části aplikace by tuto situaci nebylo třeba řešit v takové míře, protože jak již bylo psáno i v minulých kapitolách, projektová šablona .Net Blazor nabízí zakomponování Identity serveru, který lze využít pro autentizaci v aplikacích, ihned při vytváření projektu.

Z důvodu nemožnosti vytvořit digitální podpis pomocí certifikátu pomocí .Net nebylo jiného stanoviska než použít pro tvorbu BE části aplikace Java Spring i za cenu vlastní implementace správy uživatelů.

Při hledání možností, jakým způsobem naimplementovat správu uživatelů, registraci, přihlášení a schopnosti uchovat si v prohlížeči informaci o tom, že je v aplikaci uživatel přihlášen, byla nalezena metoda autentizace pomocí JWT (JSON Web Token) tokenů.

6.3.1 Autentizace pomocí JWT tokenů

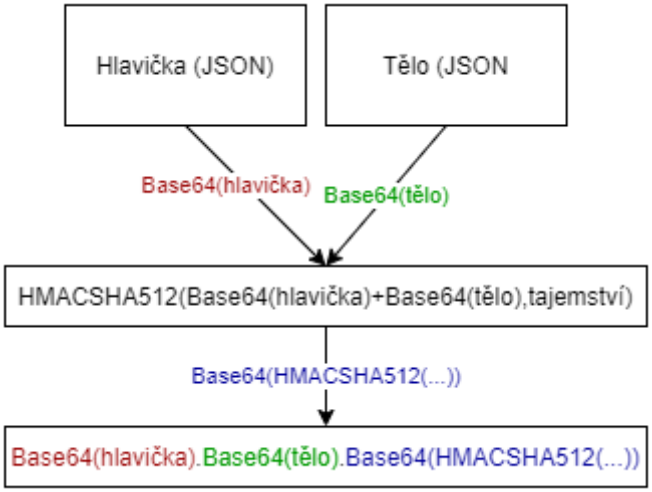
Metoda autentizace pomocí JWT je efektivním způsobem, jak si udržovat informaci o přihlášeném uživateli, aniž by se musel pro každý odeslaný požadavek na BE odesílat přihlašovací údaje. Při přihlašování klienta do systému se uživatel autentizuje svým přihlašovacím jménem a heslem. BE systém poté porovná správnost zadaných údajů a pokud nalezne shodu, vytvoří klientovi JWT token, který mu pošle zpět. Klient uloží JWT token k sobě do lokálního nebo session úložiště a při komunikaci s BE, která vyžaduje autentizaci, tento token poté odesílá na BE v autorizační hlavičce http požadavku.

V minulém odstavci je princip JWT tokenu vysvětlen jen velmi okrajově a jednoduše. JWT samozřejmě obsahuje zabezpečovací operace, aby bylo co nejsložitější tuto metodu autentizace podvrhnout. JWT token se skládá ze tří hlavních částí, z hlavičky, těla a podpisu, kdy hlavička a tělo tokenu je ve formátu JSON:

- Hlavička tokenu – vždy obsahuje typ algoritmu, pomocí kterého je vytvořen otisk hlavičky a těla tokenu a identifikaci tokenu.
- Tělo tokenu – do těla tokenu lze umístit různá data, která může klient z tokenu získat a pracovat s nimi. Token by měl obsahovat minimálně čas vytvoření tokenu, čas expirace tokenu, vystavitele tokenu a pak další libovolná data, která jsou potřeba pro práci na klientské části (např. role přihlášeného uživatele atd.).
- Podpisová část – Poslední část tokenu je digitální podpis, který slouží k identifikaci manipulace s tokenem. Podpis se vytvoří tím, že se hlavička i tělo tokenu zakódují do formátu Base64. Base64 hlavičky a těla tokenu se přiloží k sobě a oddělí se znakem „.“ (tečka) a z této části se vytvoří podpisová část pomocí algoritmu (např. HMACSHA512) na jehož vstup jde otisk hlavičky a těla a přidá se k němu tajemství, které zná pouze vystavitel tokenu. Poté se i tato podpisová část zakóduje do formátu Base64.

Ve výsledku je tedy token jen textový řetězec v následujícím tvaru – „{Hlavička zakódována do Base64}.{Tělo zakódované do Base64}.{Podpis zakódovaný do Base64}“. V tomto formátu se poté odesílá klientovi, který si jej uloží a pokud je potřeba, tak jej rozparsuje a načte z něj potřebná data. Při odesílání dalších požadavků klient posílá token v autorizační

hlavičce. BE poté ověří jeho integritu pomocí ověření podpisu tokenu a jestli je token úspěšně ověřen, a zda ještě nevypršela platnost, tak je úspěšně provedena autorizace uživatele.



Obr. 59 Postup vytvoření JWT tokenu (vlastní)

Pro práci s JWT tokeny slouží na BE třída *JwtTokenService*, která obsahuje operace pro vygenerování nového tokenu, umožňuje získat jednotlivé položky z existujícího tokenu a umožňuje také token zvalidovat. Pro samotné generování a parsování JWT tokenů je použita třída *Jwts*, která je dostupná v použité knihovně JJWT.

Pro práci s JWT tokeny na FE je vytvořena třída *JwtParser*, která slouží k deserializaci tokenu a získání potřebných údajů, které jsou do tokenu získány (např. expirace tokenu či veřejné id autentizovaného uživatele).

6.3.2 Implementace správy uživatelů na BE

Při implementaci správy uživatelů na BE bylo nejprve potřeba vytvořit entitu uživatele a vytvořit v databázi tabulku. Pro entitu byla vytvořená třída s názvem *UserEntity*, která se skládá z následujících proměnných:

- id (long) – automaticky generované id v databázové tabulce
- publicId (UUID) – veřejné id, na které jsou poté vázány i další entity
- email (String) – emailová adresa registrovaného uživatele, která se používá i pro přihlášení do systému
- passwordHash (String) – otisk přihlašovacího hesla
- isDeleted (boolean) – příznak, zda je daný uživatel smazán

- emailConfirmed (boolean) – příznak, zda má daný uživatel potvrzen emailovou adresu

Pro práci se samotnou tabulkou byla vytvořeno rozhraní *UserRepository*, které má speciální anotaci *@Repository*. Tahle anotace při spouštění aplikace způsobí, že po připojení aplikace k cílové databázi dojde ke kontrole, zda daná tabulka existuje, případně ji automaticky vytvoří. Dále tato anotace zajistí, aby byla daná třída přidána do IoC kontejneru, aby bylo možno jí injektovat v dalších třídách. Rozhraní je poté ještě rozšířeno o další dva způsoby vyhledání záznamu v tabulce. Toho je docíleno pomocí definice metod v rozhraní, kdy implementaci se pokusí vytvořit automaticky knihovna JPA.

```
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    List<UserEntity> findByEmail(String email);
    Optional<UserEntity> findByPublicId(UUID publicId);
}
```

Obr. 60 Rozhraní pro práci s tabulkou uživatelů (vlastní)

Jednotlivé CRUD operace nad tabulkou uživatelů jsou naimplementovány ve třídě *UserService*, která používá anotaci *@Service*. Tato anotace frameworku Spring zajistí přidání dané třídy do IoC. Pro zahájení procesu registrace, přihlášení, potvrzení emailu, reset hesla, změnu hesla, získání informací o uživateli podle veřejného id, znovu odeslání emailu na potvrzení emailové adresy slouží třída *UserController*, označená anotací @RestController. Třída *UserController* obsahuje funkci pro každý výše uvedený proces. Všechny funkce ve třídě mají anotace, pomocí kterých je definováno, na jaký typ http požadavku cílí a definují koncovou adresu, kterou funkce obslouží, když uživatel na danou adresu odešle http požadavek.

6.3.2.1 Registrace uživatele

Registrace nového uživatele je implementována ve funkci *registerUser*. Tato funkce zpracovává http POST požadavek na koncové adrese */user/register*. V těle požadavku jsou předány potřebné údaje pro registraci, tedy email a heslo.

Před vytvořením daného účtu se nejprve zkontroluje, zda zadaná emailová adresa nemá v systému aktivní účet. Pokud v systému existuje aktivní účet s danou adresou, proces registrace uživatele je přerušen. Při neexistenci aktivního účtu pro zadanou emailovou adresu se vytvoří nová instance entity *UserEntity* a naplní se cílovými hodnotami, tedy emailovou adresu, hash otisk hesla, který se získá pomocí algoritmu BCrypt. Nová entita se vloží do

databáze, pro nově přidaného uživatele se vytvoří JWT token a na zadanou emailovou adresu se odešle email pro potvrzení jeho emailové adresy.

V odpovědi http požadavku se poté odesílá samotný token, uživatelské jméno (emailová adresa) a příznak, zda má potvrzenou emailovou adresu.

```
var pwdHash = bcrypEncoder.encode(user.getPassword());
var newUserEnt = new UserEntity(user.getEmail(), pwdHash);
var addedUser = userService.addUser(newUserEnt);
authManager.authenticate(new UsernamePasswordAuthenticationToken(user.getEmail(), user.getPassword()));

var token = jwtTokenService.generateNewToken(addedUser);
try {
    emailService.sendActivationEmail(addedUser.getEmail(), addedUser.getPublicId());
} catch (Throwable e) {
    System.out.println(e);
}

// return user and session
return ResponseEntity
        .status(HttpStatus.CREATED)
        .body(new UserLoginResp(addedUser.getEmail(), addedUser.isEmailConfirmed(), token));
```

Obr. 61 Vytvoření nového uživatele (vlastní)

6.3.2.2 Přihlášení uživatele

Implementace procesu přihlášení je implementována ve funkci *loginUser*, která se zavolá při zaslání http POST požadavku s koncovou adresou */user/login*. V těle požadavku se posílá přihlašovací emailová adresa a heslo.

Při procesu přihlášení se pomocí předané emailové adresy pokusí vyhledat existence účtu v databázi. Pokud záznam není nalezen nebo entita má nastaven příznak *isDeleted*, který značí, že je účet smazán, je navrácena http odpověď se statusem 404. Po nalezení platné entity uživatele se vypočítá otisk ze zadaného hesla a ten se porovná s otiskem uloženým v entitě. Při shodě se vytvoří autentizační záznam, aby BE věděl, že uživatel je autentizován, vytvoří se JWT token a ten se společně s uživatelským jménem a příznakem o potvrzené emailové adrese vrací v http odpovědi se statusem 200.

```
var userEnt = userService.getUserByEmail(user.getEmail());
if(userEnt != null && !userEnt.isDeleted()) {
    authManager.authenticate(new UsernamePasswordAuthenticationToken(user.getEmail(), user.getPassword()));
    var token = jwtTokenService.generateNewToken(userEnt);
    return ResponseEntity.ok(new UserLoginResp(userEnt.getEmail(), userEnt.isEmailConfirmed(), token));
}
else {
    return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(new ErrorResp(HttpStatus.NOT_FOUND.value(), ErrorReturnCodes.errNotFound, "Uživatel nebyl
}
```

Obr. 62 Část procesu přihlášení uživatele (vlastní)

6.3.2.3 Potvrzení emailové adresy

Proces potvrzení emailové adresy je spouštěn při registraci uživatele nebo při žádosti o opětovné zaslání. Tento proces zpracovává funkce *confirmEmail*, která se zavolá při přijetí http GET požadavku s koncovou adresou */confirmEmail/{id}*, kdy *id* je veřejné id daného uživatele.

Při úspěšném či neúspěšném potvrzení emailové adresy je poté přesměrován na obrazovku s danou informací, že emailová adresa byla úspěšně potvrzena či nikoli.

6.3.2.4 Resetování hesla

Resetování hesla implementuje funkce *resetPassword*, která obslouží http PUT požadavek s koncovou adresou */user/forgottenPassword*. Pro resetování hesla je v tělu požadavku vložena emailová adresa uživatele, pro kterou se má heslo resetovat. Dotyčnému je poté vygenerováno nové heslo a posláno na zadanou emailovou adresu s informací, ať si heslo po přihlášení změní, protože jej při posílání přes email může potenciálně odchytit třetí osoba.

```
// generate new random pwd
var newPwd = getRandomChars();
var newPwdHash = bcrypEncoder.encode(newPwd);
// send email
try {
    emailService.sendResetPwdEmail(user.getEmail(), newPwd);
} catch (Exception ex) {
    return ResponseEntity
            .status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body(new ErrorResp(HttpStatus.INTERNAL_SERVER_ERROR.value(),
}
// update password in database
user.setPasswordHash(newPwdHash);
try {
    userService.updateUser(user);
```

Obr. 63 Resetování hesla, část kódu implementace (vlastní)

6.3.2.5 Změna hesla

Proces změny hesla implementuje funkce *changePassword*. Ta je zavolána při přijetí http PUT požadavku s koncovou adresou *user/changePassword/{publicId}*, kdy *publicId* je id uživatele, pro kterého se heslo má změnit. V těle požadavku se nachází staré heslo a nové heslo.

Při změně hesla se nejprve získá entita uživatele, a porovná se správnost starého hesla (vypočtením otisku hesla z požadavku a následným ověřením s otiskem v entitě uživatele). Při

úspěšném ověření starého hesla se následně vypočte otisk z hesla nového a tento otisk se aktualizuje v entitě uživatele.

6.3.2.6 Smazání účtu

Pokud uživatel nechce nadále využívat svůj účet, může jej „smazat“. Při mazání nedochází k úplnému smazání záznamu entity uživatele z databáze, ale jen se u něj nastaví příznak *isDeleted* na hodnotu *true*.

Smazání účtu je implementováno ve funkci *deleteUser*, která je zavolána při přijetí http POST požadavku s koncovou adresou */user/delete*. V těle požadavku se poté posílá emailová adresa a heslo uživatele, který se má smazat. Poté se zkontroluje, zda je zadané heslo správně a pokud ano, dojde k aktualizování entity uživatele a nastaví se příznak *isDeleted* na true pro daného uživatele budou smazány všechny jeho podepisovací profily a přehled jeho záznamů v blockchain.

6.3.2.7 Znovu odeslání emailu pro potvrzení emailové adresy

V případě, že by uživateli nebyl doručen email na potvrzení emailové adresy, je dostupná možnost pro jeho znovuodeslání. Tento proces je implementován ve funkci *resendAcEmail*, která se provolá při přijetí http GET požadavku s koncovou adresou */user/resendActivationEmail/{publicId}*, kdy *publicId* je veřejné id daného uživatele. Funkce poté zkontroluje, zda je daný uživatel v databázi, a při jeho nalezení odešle znovu email s žádostí o potvrzení emailové adresy.

6.3.2.8 Ochrana nepovoleného přístupu ke koncovým bodům BE

BE část aplikace definuje sadu koncových adres pro jednotlivé funkce. Některé koncové body nevyžadují autentizovaný přístup (registrace, přihlášení), ale většina z těchto koncových bodů vyžaduje, aby je mohl použít pouze autorizovaný uživatel a v případě, že k těmto bodům přistoupí neautorizovaný uživatel, bude mu přístup zamítnut a vrátí se mu odpověď se statusem 403 – neautorizováno.

Pro tuto kontrolu autentizace před samotným zavoláním funkce pro obsloužení daného koncového bodu lze v Java Spring vytvořit tzv. filtr. Filtr je třída, která rozšiřuje třídu *OnePerRequestFilter* a přepisuje její funkci *doFilterInternal*. Implementace filtru poté získá hlavičky http požadavku a zjistí, zda obsahuje hlavičku autorizace s JWT tokenem. Pokud ano, token získá a ověří, jestli je platný. Pokud všechny operace ověření budou úspěšné, bude

požadavek povolen a nechá se vykonat cílová funkce. V opačném případě bude zpracování zakázáno a požadavek zpracován nebude.

Po přidání filtru do projektu je ještě potřeba specifikovat koncové části BE, které tomuto filtru nebudou podléhat (operace přihlášení a registrace provádí neautorizovaný uživatel právě za cílem se autorizovat). Toho lze docílit nastavením koncových bodů, které nemají podléhat danému filtru v konfigurační třídě pro nastavování bezpečnostních parametrů aplikace, která se jmenuje *SecurityConfiguration* a rozšiřuje třídu *WebSecurityConfigureAdapter*.

6.3.3 Implementace správy uživatelů na FE

Implementace správy uživatelů na klientské části aplikace se týkala zejména vytvoření obrazovek pro jednotlivé akce, které jsou dostupné na BE části (viz kapitola 6.3.2) a poté implementace třídy, která bude cílové služby na BE ohledně správy uživatelů volat pomocí komunikace http(s).

Při implementaci správy uživatelů tedy vznikly obrazovky pro registraci uživatele, přihlášení uživatele, domovská stránka pro přihlášeného uživatele a obrazovka pro správu uživatelského účtu. Dále vznikla třída *AccountService*, která obsahuje metody na komunikaci s BE částí pro správu uživatelů.

6.3.3.1 Registrace uživatele

Pro registraci uživatele je vytvořena obrazovka, která obsahuje vstupní pole, do kterých je potřeba vyplnit jednotlivé údaje pro založení účtu – pole pro zadání emailové adresy, hesla a zopakování hesla a poté tlačítko pro zahájení procesu registrace. Při zadávání údajů jsou jednotlivé údaje validovány.

Při samotném vytváření a odesílání http požadavku je nejprve z povinných údajů vytvořena instance třídy, která umožní daný objekt převést do formátu JSON. Poté se vytvoří samotný http požadavek s metodou POST, do těla požadavku se vloží JSON s údaji a poté se požadavek odešle na BE. Po přijetí odpovědi a získání dat z těla odpovědi a kontroly dat (odpověď s informací o chybě či odpověď s JWT tokenem pro nově registrovaného a přihlášeného uživatele). Pokud server vrátí chybu, klientská část zobrazí informace o chybě zaslané z BE nebo pokud byl uživatel úspěšně registrován, uloží JWT token do lokálního úložiště a uživateli zobrazí domovskou obrazovku pro přihlášeného uživatele.

```
var reqJson = JsonSerializer.Serialize(req);

var request = new HttpRequestMessage(HttpMethod.Post, endpoint);
request.Content = new StringContent(reqJson, Encoding.UTF8, CertiChainApiConstants.JsonMediaType);
request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue(CertiChainApiConstants.JsonMediaType));

var response = await _httpClient.SendAsync(request);
var loginRespString = await response.Content.ReadAsStringAsync();
```

Obr. 64 Ukázka kódu odeslání http požadavku na BE (vlastní)

6.3.3.2 Přihlášení uživatele

Pro přihlášení uživatele je vytvořena obrazovka, ve které jsou k dispozici pole pro zadání emailové adresy a hesla. Pro zahájení procesu přihlášení je k dispozici tlačítko *Přihlásit*. Při přihlášení se zadané údaje vloží do instance třídy, která se poté serializuje do JSON formátu. Poté se vytvoří se http POST požadavek, do těla požadavku se vloží vytvořený JSON a odešle se na BE. Při přijetí odpovědi BE se odpověď zkontroluje, zda obsahuje informace o chybě či JWT token pro autentizaci. Při úspěšném přihlášení se token uloží do lokálního úložiště a uživatel je automaticky přesměrován na domovskou obrazovku přihlášeného uživatele. Při chybném přihlášení je zobrazena hláška o chybě.

Dále obrazovka pro přihlášení obsahuje tlačítko pro řešení situace zapomenutého hesla. Pokud uživatel zapomněl heslo a bude jej chtít resetovat, po kliknutí na tlačítko „Zapomněl jsem heslo“ se mu zobrazí modální dialog, kde je potřeba zadat emailovou adresu. Po stisku tlačítka „Resetovat heslo“ v dialogu se vytvoří http PUT požadavek, kdy v těle požadavku se pošle emailová adresa žadatele (součást JSON). Po resetování hesla je uživatel informován, že nové heslo bylo zasláno na cílovou emailovou adresu.

6.3.3.3 Domovská obrazovka přihlášeného uživatele

V rámci správy uživatelů byla implementována i domovská stránka uživatele, která slouží jako rozcestník pro dostupné funkcionality, které má k dispozici přihlášený uživatel. Tato obrazovka nemá žádné další specifické funkcionality, obsahuje pouze sadu informací o dané funkcionalitě a poté tlačítko, které uživatele přesměruje na obrazovku, ve které je daná funkcionalita naimplementována.

6.3.3.4 Správa účtu

Pro změnu hesla, znovuzaslání emailu pro potvrzení emailové adresy či deaktivaci účtu je vytvořena obrazovka Správa účtu, která se skládá ze dvou částí. První část se zabývá změnou hesla. Pro změnu hesla jsou zde k dispozici pole pro zadání starého hesla, nového hesla a

ještě pro zopakování nového hesla. Všechny tato pole jsou validována (minimální počet znaků, zopakovaní nového hesla musí souhlasit s novým heslem). Proces změny hesla na BE lze zahájit tlačítkem „Změnit heslo“, kdy se nejprve ze zadaných údajů vytvoří instance třídy, pomocí které bude objekt převeden do JSON formátu, poté se vytvoří http PUT požadavek s koncovou adresou pro změnu hesla na BE. Po přijetí odpovědi z BE se zkontroluje, zda bylo heslo změněno a v případě úspěchu je zobrazena hláška o úspěchu pomocí komponenty Toast. V případě chyby je v Toast zobrazen detail chyby.

Druhá část obrazovky se zabývá smazáním účtu. Ta se skládá z informací, co se stane při procesu smazání a pak samotného tlačítka pro zahájení procesu deaktivace. Pro zabránění deaktivace při omylném stisknutí tlačítka se uživateli zobrazí dialog, kde je zobrazen dotaz, zda vážně chce účet smazat a pole pro zadání hesla, které je k deaktivaci vyžadováno. Po zahájení procesu se vytvoří http POST požadavek, kdy v těle požadavku je vloženo zadané heslo a požadavek je odeslán na BE. Po přijetí odpovědi se zkontroluje obsah odpovědi a pokud bylo úspěšné, je uživatel přesměrován na úvodní stranu aplikace a je odmazán jeho aktuální JWT token.

Pokud uživatel nemá potvrzenou emailovou adresu, v obrazovce je zobrazeno upozornění na tuto skutečnost. V případě že mu email pro potvrzení emailové adresy nebyl doručen, může si jej nechat pomocí tlačítka „Znovu odeslat aktivační email“ zaslat.

6.3.3.5 Odhlášení uživatele

Pro odhlášení uživatele je k dispozici tlačítko v hlavičce aplikace, které je zobrazeno pouze v případě, že je nějaký uživatel přihlášen. Při procesu odhlášení je odstraněn JWT token z lokálního úložiště a je přesměrován na úvodní stranu aplikace

6.3.3.6 Způsob implementace zobrazení částí pro přihlášené uživatele

Zobrazování částí obrazovek, které jsou dostupné pouze pro přihlášené uživatele, bylo naimplementováno za pomocí knihovny Microsoft.AspNetCore.Components.WebAssembly.Authentication. Tato knihovna obsahuje komponentu *AuthorizeView*, která může obsahovat komponenty *Authorized* a *NotAuthorized*. Do komponenty *Authorized* lze poté vložit část obrazovky, která se zobrazí uživateli, který je přihlášen a do komponenty *NotAuthorized* vložit sekci, která se zobrazí, pokud uživatel přihlášen není.

Rozhodování, zda je uživatel autentizován a má přistup do částí aplikace pro přihlášené uživatele, lze naimplementovat vytvořením třídy, která dědí od třídy *AuthenticationStateProvider,* a přepsat její metodu *GetAuthenticationStateAsync*.

Pro vlastní implementaci zjištění stavu, zda je uživatel přihlášen, byla vytvořena třída *CustomAuthenticationProvider*, kdy v přepisované metodě se pokusí získat JWT token z lokálního úložiště, poté se z něj získá datum expirace a zkontroluje se, zda je token ještě platný. Pokud jsou všechny podmínky splněny, je na dané obrazovce zobrazena část uvnitř komponenty *Authorized.*

```
string token = await _localStorageService.GetItemAsync<string>(JwtParser.JwtTokenKeyName);
if (string.IsNullOrEmpty(token))
    return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity() { }));

var claims = JwtParser.ParseClaimsFromJwt(token);
if (claims is not null)
{
    // kontrola expirace tokenu, pokud je uživatel se bude muset přihlásit znovu
    var expClaim = claims.FirstOrDefault(x => x.Type == "exp");
    Console.WriteLine(expClaim?.Value);
    if (expClaim is not null)
    {
        var expTicks = long.Parse(expClaim.Value);
        var expDate = new DateTime(1970, 1,1, 0, 0, 0, DateTimeKind.Local).AddMilliseconds(expTicks * 1000);

        if (DateTime.Now > expDate)
        {
            await _localStorageService.RemoveItemAsync(JwtParser.JwtTokenKeyName);
            var anonymous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity() { }));
            return anonymous;
        }
        else
        {
            var userClaimPrincipal = new ClaimsPrincipal(new ClaimsIdentity(JwtParser.ParseClaimsFromJwt(token),
            var loginUser = new AuthenticationState(userClaimPrincipal);
            return loginUser;
        }
    }
}

return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity() { }));
```

Obr. 65 Kontrola stavu autentizace (vlastní)

Aby se při zjišťování stavu autentizace používala vlastní třída, je potřeba ji nastavit jako autentizační třídu při spouštění aplikace ve třídě *Program*, kdy při vytváření aplikace je nutno použít metodu *AddAuthorizationCore* a poté přidat vlastní třídu pro kontrolu stavu autentizace do IoC kontejneru.

```
builder.Services.AddScoped<AuthenticationStateProvider, CustomAuthenticationProvider>();
builder.Services.AddAuthorizationCore();
```

Obr. 66 Konfigurace kontroly stavu autentizace (vlastní)

6.4 Správa podepisovacích profilů

Pro vytváření digitálních podpisů jsou v aplikaci kritické podepisovací profily, ve kterých jsou uloženy parametry pro vytváření jednotlivých digitálních podpisů. Při implementaci správy profilů bylo potřeba na BE přidat entity pro profily dle analýzy v kapitole 3.2 a CRUD operace nad tabulkami. Na straně FE bylo potřeba přidat dvě nové obrazovky – obrazovku se seznamem profilů uživatele a obrazovku pro vytvoření nebo editaci vybraného podepisovacího profilu.

6.4.1 Implementace správy podepisovacích profilů na BE

První krok při implementaci podpory podepisovacích profilů spočíval ve vytvoření tabulek v databázi pro jednotlivé datové entity, které jsou rozepsány v kapitolách 3.2.2 až 3.2.6. Pro definici jednoho řádku v tabulkách byly tedy vytvořeny následující třídy s anotací *@Entity*:

- *ProfileGeneralEntity* pro obecný profil
    - *id* (long) – primární klíč, automaticky generovaná hodnota, id profilu obecného podpisu
    - *profileType* (ProfileTypes) – výčtový typ určující typ profilu (PAdES, XAdES, CAdES)
    - *userId* (UUID) – veřejné id uživatele, který profil vlastní
    - *fullProfileId* (long) – id na entitu se specifickými parametry pro daný typ podpisu
    - *signerInfoId* (long) – id entity s informacemi o podepisujícím
    - *hashAlgorithm* (HashAlgorithm) – výčtový typ určující typ hashovacího algoritmu použitého při vytváření digitálního podpisu
- *ProfileCadesEntity* pro specifické parametry podpisu CAdES
    - *id* (long) – primární klíč, automaticky generovaná hodnota, id profilu CAdES
    - *cadesType* (CadesTypes) – výčtový typ pro specifikaci druhu podpisu CAdES
    - *isDetached* (boolean) – příznak, zda je podpis součástí podepisovaných dat (FALSE) nebo je podpis vytvořen jako soubor zvlášť (TRUE)
- *ProfileXadesEntity* pro specifické parametry podpisu XAdES
    - *id* (long) – primární klíč, automaticky generovaná hodnota, id profilu XAdES
    - *xadesType* (XadesTypes) - výčtový typ pro určení typu podpisu XAdES

  - *isDetached* (boolean) - příznak, zda je podpis součástí podepisovaných dat (FALSE), nebo je podpis vytvořen jako soubor zvlášť (TRUE)
- *ProfilePadesEntity* pro specifické parametry podpisu PAdES
  - *id* (long) – primární klíč, automaticky generovaný, id profilu PAdES
  - *padesType* (PadesTypes) – výčtový typ určující typ podpisu PAdES
  - *isVisible* (booleant) – příznak, zda se jedná o neviditelný (FALSE), nebo viditelný (TRUE) podpis
  - *visibleSignatureType* (VisibleSignatureTypes) – výčtový typ určující typ viditelného podpisu (pouze text, pouze obrázek, text i obrázek)
  - *signatureImage* (byte[]) – obrázek podpisu
  - *pos_x* (double) – horizontální umístění podpisu na stránce v procentech celkové šířky stránky dokumentu
  - *pos_y* (double) – vertikální umístění viditelného podpisu v procentech celkové výšky stránky dokumentu
  - *width* (double) – šířka viditelného podpisu v procentech celkové šířky stránky dokumentu
  - *height* (double) – výška viditelného podpisu v procentech celkové výšky stránky dokumentu
  - *page* (int) – stránka, na které bude viditelný podpis umístěn
  - *signatureText* (String) – text, který bude zobrazen ve viditelném podpisu
  - *signerName* (String) – jméno podepisujícího
  - *signerLocation* (String) – lokace, kde byl podpis vytvořen (např. název města)
  - *signReason* (String) – důvod vytvoření digitálního podpisu
- *SignerInfoEntity* pro datovou entitu s dodatečnými informacemi o podepisujícím
  - *id* (long) – primární klíč, automaticky generovaná hodnota,
  - *country* (String) – název státu
  - *province* (String) – název kraje
  - *postalCode* (String) – Poštovní směrovací číslo města / obce
  - *city* (String) – název města / obce
  - *role* (String) – role podepisujícího
  - *isProofOfOrigin* (boolean) – příznak, zda podpis stvrzuje původ dokumentu (TRUE), nebo ne (FALSE)

- *isProofOfCreation* (boolean) – příznak, zda podpis stvrzuje původ vytvoření dokumentu (TRUE), nebo ne (FALSE)

Pro uvedené entity poté byla vytvořena rozhraní, která reprezentují samotné tabulky v databázi. Bylo vytvořeno tedy pět rozhraní, která implementují rozhraní *JpaRepository* s anotací @Repository:

- *ProfileGeneralRepository* pro *ProfileGeneralEntity*
- *ProfileCadesRepository* pro *ProfileCadesEntity*
- *ProfileXadesRepository* pro *ProfileXadesEntity*
- *ProfilePadesRepository pro ProfilePadesEntity*
- *SignerInfoRepository* pro *SignerInfoEntity*

Pro rozhraní reprezentující jednotlivé tabulky v databázi nebylo potřeba vytvářet vlastní definice funkcí pro vyhledávání pomocí jiných hodnot než samotné id.

Pro práci se všemi tabulkami byla vytvořena třída *ProfileGeneralService*, která implementuje funkce pro jednotlivé operace s profily a využívá CRUD operací nad výše zmíněnými databázovými entitami a práce s datovými entitami se odvíjí také od typu podpisu v profilu. Jednotlivé operace musí být naimplementovány důkladně, aby při různých operacích nevznikal nekonzistentní stav databáze.

Příklad možného vzniku nekonzistentního stavu databáze – uživatel má podepisovací profil typu CAdES. Obecné parametry profilu jsou uloženy v entitě *ProfileGeneralEntity*, ta obsahuje typ podpisu a poté id na odpovídající entitu v tabulce *ProfileCadesRepository*, která obsahuje specifické parametry podpisu. Uživatel při úpravě vybraného profilu změní typ podpisu na XAdES. Při aktualizaci záznamů v databázi je potřeba dbát na tyto situace a při této situaci smazat záznam z *ProfileCadesRepository* a vytvořit nový záznam s parametry podpisu XAdES v tabulce *ProfileXadesRepository* a id nového záznamu aktualizovat v entitě *ProfileGeneralEntity.* Pokud by systém pouze vytvořil nový záznam se specifickými parametry podpisu a staré parametry nesmazal, starý záznam by zůstal uložen v databázi, ale už by k tomuto záznamu nikdo nikdy nepřistoupil.

Pro komunikaci mezi FE a BE aplikace je pro podepisovací profily vytvořena jedna obecná třída *GeneralProfileModel*, která obsahuje všechny obecné parametry podepisovacího profilu i všechny specifické parametry jednotlivých podpisů. Při odesílání dat profilu jsou poté data z entity pře-mapována na tento obecný objekt, který se na pozadí převede do JSON a

v http odpovědi se odešle na FE. Obdobně funguje operace přidání / aktualizace podepisovacího profilu. Na BE dorazí parametry v obecném formátu a postupně jsou pře-apovány na jednotlivé třídy entit podle typu podepisovacího profilu.

```
{
    "id": "123",
    "name": "nameValue",
    "profileType": "PAdES",
    "hashAlgorithm": "SHA256",
    "signerInfo": {...},
    "xadesProfile": null,
    "cadesProfile": null,
    "padesProfile": {...}
}
```

Obr. 67 definice JSON obecného objektu profilu pro komunikace mezi FE a BE (vlastní)

6.4.1.1 Implementace CRUD operací pro podepisovací profily

Při přidávání nového podepisovacího profilu se z obecného objektu podepisovacího profilu nejprve vytvoří nová instance třídy *ProfileGeneralEntity* a nastaví se všechny obecné parametry (název profilu, použitý hash algoritmus, typ podpisu, id uživatele). Podle typu podpisu (XAdES, PAdES, CAdES) se vytvoří nová instance cílové entity a nastaví se všechny její proměnné a uloží se do cílové tabulky. Id entity se specifickými parametry podpisu se uloží do instance *ProfileGeneralEntity*. Pokud obecný objekt obsahuje i objekt *signerInfo*, vytvoří se nová instance entity *SignerInfoEntity*, třídní proměnné se nastaví dostupnými hodnotami a uloží se do databáze. Id entity s informacemi o podepisujícím se poté vloží do entity *ProfileGeneralEntity*, která se uloží do databáze. Tato funkčnost je implementována ve funkci *addNew* ve třídě *ProfileGeneralService*.

Při získávání profilů daného uživatele se získávají pouze záznamy z tabulky s obecným profilem. Při této akci není potřeba sestavovat kompletní objekt profilu, protože na klientské části by nebyly nikdy využity. Na klientské části se získává kolekce profilů pouze pro důvod jejich zobrazení v obrazovce se seznamem profilů nebo při výběru profilu při vytváření podpisu. Proto tedy stačí získat entity podle id uživatele, kdy samotné získání profilů probíhá získáním všech záznamů profilů. Při procházení se poté záznamy, které patří přihlášenému uživateli, pře-mapují na obecný objekt *GeneralSignatureModel,* vloží se do kolekce, která je nakonec navrácena zpět na FE. Získání profilů uživatele je implementováno ve funkci *getAllForUser* ve třídě *GeneralSignatureService*.

```
public List<GeneralProfileModel> getAllForUser(UUID userId) {
    var profiles = profileGeneralRepository.findAll();
    var models = new ArrayList<GeneralProfileModel>();
    for(var profileEnt : profiles) {
        if(profileEnt.getUserId().equals(userId)) {
            var model = new GeneralProfileModel();
            model.setId(profileEnt.getId());
            model.setName(profileEnt.getName());
            model.setHashAlgorithm(profileEnt.getHashAlgorithm());
            model.setProfileType(profileEnt.getProfileType());

            models.add(model);
        }
    }
    return models;
}
```

Obr. 68 Zdrojový kód pro získání profilů uživatele (vlastní)

Další funkce ve třídě *GeneralSignatureService* je *getOneFull* pro získání jednoho určitého profilu včetně specifických parametrů daného typu podpisu. Tato funkce je potřeba při aktualizaci profilu a poté při samotném vytváření digitálního podpisu, aby se parametry mohly nastavit službě, která samotný digitální podpis vytvoří, viz kapitola 6.5.1. Při sestavování kompletního profilu podpisu je nejprve získán záznam z tabulky s obecným nastavením profilu. Podle hodnoty typu podpisu, kterou záznam obsahuje a id entity se specifickými parametry pro profil, se získají zbylá potřebná data z cílové tabulky. Pokud je součástí záznamu id pro záznam s informacemi o podepisujícím, získají se i tato data a poté se vše pře-mapuje do obecného objektu *GeneralProfileModel*, ve kterém jsou poté odesílány na FE.

```
var generaProfileEnt = profileGeneralRepository.findById(id);
if(generaProfileEnt.isPresent()) {
    var genProfile = generaProfileEnt.get();

    if(!genProfile.getUserId().equals(userId))
        throw new EntityNotFoundException("GeneralProfileEntity with id = " + id + " not found");

    var model = new GeneralProfileModel();
    model.setId(genProfile.getId());
    model.setProfileType(genProfile.getProfileType());
    model.setName(genProfile.getName());
    model.setHashAlgorithm(genProfile.getHashAlgorithm());
    // append signer info if available
    if (genProfile.getSignerInfoId() != null) {···
    }
    // get sign detail setting using type
    switch(genProfile.getProfileType()) {
        case CAdES:
            var cades = getCadesSettings(genProfile.getFullProfileId());
            if(cades == null) throw new EntityNotFoundException("Cades settings not found");
            model.setCadesProfile(cades);
            break;
        case PAdES:
            var pades = getPadesSettings(genProfile.getFullProfileId());
            if(pades == null) throw new EntityNotFoundException("Pades settings not found");
            model.setPadesProfile(pades);
            break;
        case XAdES:
            var xades = getXadesSettings(genProfile.getFullProfileId());
            if(xades == null) throw new EntityNotFoundException("Xades settings not found");
            model.setXadesProfile(xades);
            break;
        default:
            throw new Exception("UnsupportedType - " + genProfile.getProfileType().toString());
    }
```

Obr. 69 Zdrojový kód sestavení kompletního profilu (vlastní)

Smazání podepisovacího profilu je implementováno ve funkci *deleteProfile* ve třídě *GeneralProfileService*. Při odmazání je nejprve získán záznam z tabulky s obecnými parametry podepisujícího profilu. Podle typu podpisu je poté nejprve smazán záznam se specifickými parametry podpisu, a dále pokud obecný záznam obsahuje id na záznam v tabulce s informacemi o podepisujícím, je smazán i tento záznam, a nakonec je smazán samotný záznam s obecnými informacemi o podepisovacím profilu.

```
// delete signer info id available
if(gpe.getSignerInfoId() != null)
    signerInfoRepository.deleteById(gpe.getSignerInfoId());

// delete profile settings
var fullProfileId = gpe.getFullProfileId();
switch (gpe.getProfileType()) {
    case XAdES:
        xadesRepository.deleteById(fullProfileId);
        break;
    case CAdES:
        cadesRepository.deleteById(fullProfileId);
        break;
    case PAdES:
        padesRepository.deleteById(fullProfileId);
        break;
}

// delete general profile
profileGeneralRepository.delete(entity.get());
```

Obr. 70 Část kódu pro smazání podepisovacího profilu (vlastní)

Nejkomplikovanější CRUD operací pro správu profilů je aktualizace existujícího profilu. Při aktualizaci je nejprve porovná typ podepisovacího profilu. Pokud se typy ve staré a nové verzi profilu liší, je smazán starý záznam se specifickými parametry a vytvořena nová entita se specifickými parametry a její id je aktualizováno v entitě obecného profilu. Pokud se typy profilů shodují, je provedena pouze aktualizace. Dále je potřeba kontrolovat změnu entity s informacemi o podepisujícím. Při aktualizaci podepisovacího profilu může dojít k jejímu smazání, přidání či aktualizaci.

6.4.1.2 Implementace metod pro obsluhu http požadavků

Po implementaci třídy *GeneralSignatureService* zbývá využít naimplementované metody v aplikaci a umožnit jejich provolání z FE části aplikace. Pro vytvoření API pro správu podepisovacích profilů byla vytvořena nová RestController třída *ProfileController*.

Pro získání profilů uživatele je k dispozici funkce *getProfiles*, která se provede při přijetí http GET požadavku s koncovou adresou /profiles/{userdId}, kdy *userId* je parametr v URL

adrese s id uživatele, který chce své profily získat. Samotné vyhledání profilů provede třída *GeneralSignatureService* a její funkce *getAllForUsers*.

Pro získání kompletního podepisovacího profilu slouží funkce *getFullProfile*. Ta se obslouží při přijetí http GET požadavku s koncovou adresou /profiles/{userId}/{profileId}, kde parametr *userId*, je id uživatele a parametr *profileId* je id podepisovacího profilu. Získání samotného profilu je naimplementováno ve funkci *getOneFull* uvnitř třídy *GeneralProfileService*. Profil se pomocí této funkce pokusí získat a v případě nalezení je poté navrácen v http odpovědi na FE. V případě že profil nebyl nalezen, je navrácena odpověď s chybovou hláškou neexistujícího profilu.

Pro přidání nového profilu je vytvořena funkce *addProfile,* která obslouží http POST požadavek s koncovou adresou */profiles/{userId}/add*, kdy parametr *userId* je id uživatele, který profil vytváří. Http požadavek v těle obsahuje JSON objekt, který je na pozadí převeden do třídy *GeneralProfileModel*. Tento objekt je poté validován, zda obsahuje všechny potřebné údaje. Samotná validace probíhá ve třídě *ProfileValidator* ve funkci *ValidateGeneralProfileModel*. Při úspěšné validaci je zavolána funkce *addNew* ve třídě *GeneralProfileService*, která provede samotné přidání profilu. Po úspěšném přidání záznamu je na FE navrácena http odpověď se statusem 200, jenž signalizuje úspěch operace. V případě neúspěchu je navrácen status kód podle zachycené chyby (např. 401 při neúspěšné validaci) a tělo odpovědi obsahuje informace o chybě.

Pro smazání profilu je k dispozici funkce *deleteProfile*, která obslouží http DELETE požadavek s koncovou adresou */profiles/{userId}/{profileId}*, kdy parametr *userId* je id uživatele a parametr *profileId* je id profilu. V této funkci je zavolána funkce *deletePofile* ze třídy *GeneralProfileService*, ve které je proces smazání naimplementován. Při úspěchu je uživateli vrácena odpověď se statusem 200 nebo detail chyby s příslušným statusem.

Poslední funkcí pro aktualizaci profilu je *updateProfile,* která obslouží http PUT požadavek na koncové adrese */profiles/{userId}/{profileId}*, kdy parametr *userId* je id uživatele a parametr *profileId* je id podepisovacího profilu. Aktualizace profilu je implementována ve funkci *udpateProfile* ve třídě *ProfileGeneralService*. Ve funkci kontroléru je nejprve validováno tělo požadavku, zda obsahuje povolenou kombinaci údajů, a poté je zavolána funkce pro samotnou aktualizaci záznamu podepisovacího profilu. V případě úspěchu je poté uživateli navrácena odpověď se statusem 200, v opačném případě se navrátí detail chyby s příslušným chybovým status kódem.

6.4.2 Implementace správy podepisovacích profilů na FE

Implementace správy podepisovacích profilů na FE spočívala ve vytvoření dvou nových obrazovek – obrazovka pro zobrazení podepisovacích profilů aktuálně přihlášeného uživatele a obrazovka pro vytvoření a editaci podepisovacího profilu. Dále bylo potřeba vytvořit službu pro implementaci komunikace mezi FE a BE pro jednotlivé funkce správy uživatelských profilů, které jsou popsány v kapitole 6.4.1.

6.4.2.1 Implementace třídy pro komunikaci s BE

Pro zakomponování komunikace s BE částí aplikace pro správu podepisovacích profilů byla vytvořena speciální třída *ProfileService*, která obsahuje funkce volat jednotlivé služby BE pro správu podepisovacích profilů, tedy vytvoření, aktualizaci, smazání a získání profilu či získání všech profilů vytvořené přihlášeným uživatelem. Pro každou tuto funkci je ve třídě vytvořena jedna metoda.

Pro přidání nového profilu slouží metoda *AddProfileAsync*. Parametrem této funkce je třída *ProfileGeneralReqResp*, která je identická s třídou *ProfileGeneralModel* na BE části, jen je naimplementována v .NET. Tato třída totiž slouží ke konverzi mezi formátem JSON a instancí objektu, proto musí mít tyto třídy stejné vlastnosti / proměnné, aby konverze vždy proběhla v pořádku. Funkce pro přidání vytvoří http POST požadavek, do jeho těla vloží JSON vytvořený konverzí instance třídy *ProfileGeneralReqResp*, přidá do požadavku autorizační hlavičku s JWT tokenem a požadavek odešle na příslušnou adresu BE. Po přijetí odpovědi zkontroluje její obsah a pokud obsahuje status značící úspěch operace (status 200), přesměruje uživatele na obrazovku s přehledem podepisovacích profilů.

Pro smazání existujícího profilu je k dispozici funkce *DeleteProfileAsync*. Parametrem této funkce je id profilu, který se má odstranit. Poté funkce vytvoří http DELETE požadavek, přidá mu autorizační hlavičku s JWT tokenem, nastaví koncovou adresu požadavku, která obsahuje dva dynamické parametry (id uživatele a id profilu) a odešle tento požadavek na BE. Po přijetí odpovědi zkontroluje status, který pokud neznačí úspěch, získá podrobnosti o chybě a předá zpět do obrazovky, která danou chybu zobrazí pomocí komponenty Toast.

Získání kompletního profilu je implementováno ve funkci *GetFullProfileAsync*, která vytvoří http GET požadavek, přidá k němu autorizační hlavičku s JWT tokenem a nastaví požadavku koncovou adresu se dvěma dynamickými parametry, id uživatele a id profilu. Poté tento požadavek odešle na BE a po přijetí odpovědi a při statusu značící úspěch, metoda

konvertuje tělo odpovědi do objektu *ProfileGeneralReqResp*, který je vrácen jako návratová hodnota metody.

Odeslání požadavku na aktualizaci podepisovacího profilu je implementováno ve funkci *UpdateProfileAsync*, která vyžaduje jako vstupní parametr instanci třídy *ProfileGeneralReqResp*. Funkce vytvoří http PUT požadavek, do těla požadavku vloží JSON vytvořený z parametru funkce, přidá autorizační hlavičku, nastaví koncovou adresu požadavku, do které je potřeba vložit id uživatele a id aktualizovaného profilu, a poté jej odešle na BE. Po přijetí odpovědi zkontroluje status odpovědi. Pokud operace proběhla úspěšně (status 200), přesměruje uživatele na obrazovku přehledu podepisovacích profilů a v případě chyby zobrazí detaily chyby, které se v těle chyby nacházejí.

Poslední metoda *GetProfilesAsync,* kterou třída obsahuje, je pro získání podepisovacích profilů přihlášeného uživatele. Pro získání profilů je vytvořen http GET požadavek s autorizační hlavičkou a koncovou adresou pro získání profilů, která obsahuje jeden dynamický parametr – id přihlášeného uživatele. Po získání odpovědi zkontroluje status odpovědi. Ten, pokud značí úspěch, tak metoda z těla odpovědi získá JSON, který konvertuje do kolekce třídy *ProfileGeneralResResp*. Tuto kolekci poté vrací jako návratovou hodnotu. V případě neúspěchu je získán z těla detail chyby a detail je poté zobrazen uživateli pomocí komponenty Toast.

6.4.2.2 Obrazovka pro správu podepisovacích profilů

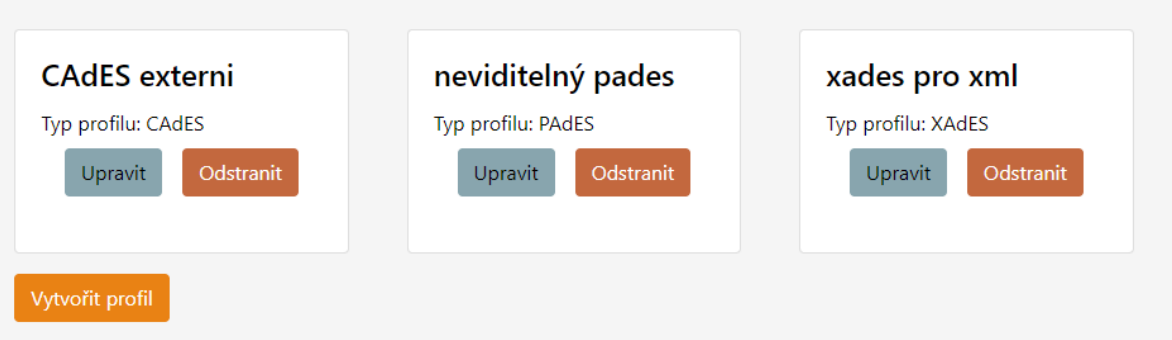
První obrazovkou je samotná správa podepisovacích profilů, na tuto obrazovku lze přejít z domovské obrazovky přihlášeného uživatele. Při zobrazování stránky se pomocí třídy *ProfileService* získá kolekce podepisovacích profilů, která patří přihlášenému uživateli. Pokud uživatel ještě nemá žádné podepisovací profily, zobrazí se mu upozornění na tuto situaci.

Pokud podepisovací profily uživatel má, jsou jednotlivé profily zobrazeny pomocí komponenty Card. Každá komponenta Card poté obsahuje název podepisovacího profilu, typ podpisu, který lze pomocí profilu vytvořit (CAdES, PAdES nebo XAdES) a tlačítka pro zahájení procesu úpravy nebo smazání daného profilu.

```
<Card ElementId="@profile.Id.ToString()" Margin="Margin.Is2">
    <CardBody>
        <CardTitle Size="4">@profile.Name</CardTitle>
        <CardText>Typ profilu: @profile.ProfileType</CardText>
        <Paragraph Flex="Flex.JustifyContent.Center">
            <Button ElementId="@GetId("btnEdit", profile.Id)" Color="Color.Secondary"
                    Clicked="@(() => EditProfile(profile.Id))" Margin="Margin.Is2">
                Upravit
            </Button>
            <Button ElementId="@GetId("btnDelete", profile.Id)" Color="Color.Danger"
                    Clicked="@(() => DeleteProfile(profile.Id, profile.Name))" Margin="Margin.Is2">
                Odstranit
            </Button>
        </Paragraph>
    </CardBody>
</Card>
```

Obr. 71 Ukázka použití komponenty Card (vlastní)

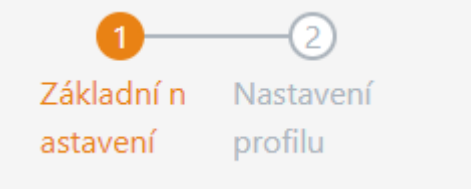
Při mazání profilu se nejprve zobrazí dotazovací dialog pro potvrzení zahájení procesu smazání vybraného profilu, aby se zamezilo nechtěnému smazání profilu. Po stisknutí tlačítka Upravit je uživatel přesměrován na obrazovku pro vytvoření / aktualizaci podepisovacího profilu. Na obrazovce je poté ještě k dispozici tlačítko pro přidání nového podepisovacího profilu.



Obr. 72 Přehled podepisovacích profilů (vlastní)

Druhá implementovaná obrazovka je pro samotné vytvoření a editaci existujícího profilu pro podepisování. Editace i vytvoření nového profilu lze implementovat pomocí jedné obrazovky, jediný rozdíl je ve volání potřebných komunikací s BE aplikace. Tento problém byl vyřešen vkládáním dodatečného parametru do adresy obrazovky, kdy při editaci profilu se do cesty vloží id editovaného profilu a pro nový profil se do adresy vloží parametr s hodnotou „new“. Pomocí tohoto parametru lze již pomocí podmínek naimplementovat požadované funkce – získání detailu profilu při zobrazování stránky a zavolání metody pro aktualizaci profilu v případě editace existujícího profilu. V případě vytváření nového profilu není při načítání stránky potřeba dělat nic a při ukládání profilu je potřeba zavolat metodu pro vytvoření profilu.

Obrazovka pro vytvoření / editaci profilu se skládá ze dvou částí. Tyto části byly vytvořeny pomocí komponenty *StepPanel*, která na výsledné obrazovce zobrazí průběh podle deklarovaných kroků.



Obr. 73 Indikátor dostupných kroků pomocí komponenty *StepPanel* (vlastní)

Jednotlivé části daného kroku se poté přidávají pomocí komponent *StepPanel*. První část obrazovky se týká nastavení obecných parametrů podepisovacího profilu, tedy název profilu, typ podepisovacího profilu, použitý Hash algoritmus. V základním nastavení je k dispozici ještě zaškrtávací pole, kterým lze zobrazit i dodatečné pole pro nastavení informací o podepisujícím (*SignerInfoEntity* na straně BE). Všechny položky jsou validovány a při chybném vyplnění každé položky je informován validační hláškou, která je vždy umístěna u cílového pole.

Pro nastavení informací o podepisujícím jsou k dispozici pole pro výběr státu, textové pole pro zadání kraje, PSČ, města a role podepisujícího. Původ dokumentu a původ vytvoření dokumentu lze nastavit pomocí dostupných zaškrtávacích polí. Všechny položky v informacích o podepisujícím jsou volitelné, proto nejsou nijak validovány a záleží pouze na uživateli, jestli je vyplní nebo ne.

Druhá část obrazovky pro vytvoření / editaci podepisovacího profilu je dynamická a její obsah závisí na typu podpisu, který je vybrán v sekci základního nastavení podepisovacího profilu. V sekci profilu jsou totiž vždy nastavovány parametry specifické pro daný typ podpisu. Podle typu podpisu se zobrazí následující formuláře:

- CAdES – Pro CAdES lze nastavit specifický typ podpisu CAdES (B, T). Tento parametr lze vybrat z výběrového menu. Poté je k dispozici ještě jedno výběrové pro specifikování, zda bude podpis součástí podepisovaných dat nebo bude digitální podpis vytvořen jako zvláštní soubor.
- XAdES – zde jsou formulářová pole obdobná jako u CAdES. Specifické parametry pro podpis XAdES je typ podpisu XAdES (B, T), který lze zvolit pomocí výběrového pole a druhý specifický parametr se týká, zda je dostupný podpis součástí podepisovaných dat, nebo se jedná o externí podpis.

- PAdES – obsahuje nejvíce specifických parametrů. Pro nastavení typu podpisu PAdES (B, T) je dostupné výběrové pole, dále pomocí textových polí lze nastavit parametry jméno podepisujícího, místo podepsání a důvod podepsání. Dále je na obrazovce dostupné zaškrtávací pole, které určuje, zda se jedná o viditelný podpis. Při zaškrtnutí pole se zobrazí další část obrazovky, která se týká parametrizace viditelného podpisu. Umístění a velikost viditelného podpisu si lze nastavit pomocí komponenty *Slider*. Stranu umístění viditelného podpisu lze zvolit nadefinovat pomocí pole pro zadání čísla. Poté je potřeba vybrat typ viditelného podpisu (text, obrázek, obojí). Podle vybraného typu viditelného podpisu se buď zobrazí pole pro zadání textu umístěného ve viditelném podpisu, nebo pole pro výběr obrázku, nebo obě pole případě, že se viditelný podpis má skládat z textu i obrázku.

6.5 Implementace vytváření digitálního podpisu

Implementace vytvoření nového digitálního podpisu pomocí digitálního certifikátu se opět dotkla obou částí aplikací. Většina změn byla vyžadována při implementaci BE, kde se samotný podpis vytváří. Na straně FE bylo potřeba naimplementovat novou obrazovku, která uživatele provede procesem vytvoření digitálního podpisu.

6.5.1 Vytvoření digitálního podpisu na straně BE

Při implementaci procesu vytvoření digitálního podpisu na BE části aplikace nebylo potřeba přidávat žádné nové databázové entity. Při procesu vytváření digitálního podpisu bude využita již naimplementovaná entita podepisovacího profilu. Při implementaci vytváření podpisu vznikla nová třída *SignatureService* s anotací *@Service*, ve které bude naimplementován kompletní proces vytvoření digitálního podpisu, a třída *SignatureController* s anotací *@RestController*, která bude obsluhovat funkce pro obsloužení požadavků na vytvoření digitálního podpisu.

Třída *SignatureService* obsahuje veřejnou funkci *createSignature*, která zaštiťuje proces vytvoření digitálního podpisu. Vstupní parametry této funkce instance třídy *GeneralProfileModel*, která obsahuje všechny parametry nastavení vybraného podepisovacího profilu a instanci třídy *CreateSignatureReg*, která obsahuje potřebná data na vytvoření podpisu – data k podepsání, certifikát ve formátu PKCS#12 (včetně privátního klíče), heslo k certifikátu a příznak, zda se má vytvořený podpis přidat do systému blockchain. Podle typu podpisu (CAdES, PAdES, XAdES) na který je vybraný podepisovací profil vytvořen, se dále funkce

createSignature větví do dalších funkcí *createPadesSignature, createXadesSignature* a *createCadesSignature.*

```
public CreateSignatureResp createSignature(GeneralProfileModel signingProfile, CreateSignatureReq signatureReq)
    switch (signingProfile.getProfileType()){
        case PAdES:
            return createPadesSignature(signingProfile.getPadesProfile(), signingProfile.getHashAlgorithm(),
                    signatureReq, signingProfile.getSignerInfo());
        case XAdES:
            return  createXadesSignature(signingProfile.getXadesProfile(), signingProfile.getHashAlgorithm(),
                    signatureReq, signingProfile.getSignerInfo());
        case CAdES:
            return createCadesSignature(signingProfile.getCadesProfile(), signingProfile.getHashAlgorithm(),
                    signatureReq, signingProfile.getSignerInfo());
        default:
            throw new NotSupportedException(signingProfile.getProfileType() + "není podporován");
    }
}
```

Obr. 74 Funkce *createSignature* (vlastní)

6.5.1.1 Vytvoření podpisu CAdES

Vytvoření podpisu CAdES je implementováno ve funkci *createCadesSignature*. Pro vytvoření jsou použity třídy z knihovny DSS.

Nejprve je potřeba nastavit parametry podpisu CAdES. K tomu slouží třída *CAdESSignatureParams* a nastavené parametry v profilu podpisu aplikace. Podle hodnot v profilu je nastaveno, zda podpis bude součástí podepisovaných dat, nebo bude podpis externí. Poté je nastaven hashovací algoritmus, který bude použit při vytváření podpisu. Dále se do parametrů podpisu CAdES vloží podpisový certifikát, nastaví se informace o podepisujícím, pokud jsou v profilu dostupné, nastaví se typ podpisu CAdES.

Poté je potřeba z podepisovaných dat vytvořit instanci třídy, která implementuje rozhraní *DSSDocument*. Při vytváření digitálního podpisu má BE data k podepsání uložena v paměti, pro tuto situaci disponuje DSS knihovna třídou *InMemoryDocument*, která rozhraní *DSSDocument* implementuje.

Pro vytvoření podpisu je využita třída *Pkcs12SignatureToken*, která pomocí podpisového certifikátu vytvoří samotný podpis pro podepisovaná data, a nakonec pomocí třídy *CAdESService* je celý dokument podepsán (přidání podpisu k podepisovaným datům nebo vytvoření samotného externího podpisu).

Na závěr se vytvoří nová instance třídy *CreateSignatureResp*, do které se vytvoří podepsaná data nebo externí podpis, MimeType podepisovaných dat a příponu souboru podepsaných dat.

```
InMemoryDocument documentToSign = new InMemoryDocument(signatureReq.getDataToSign(),
        signatureReq.getFileName(), MimeType.fromMimeTypeString(signatureReq.getFileMimeType()));

var dataToSign = service.getDataToSign(documentToSign, cadesParams);

var signingToken = new Pkcs12SignatureToken(signatureReq.getSignCertificate(),
        new KeyStore.PasswordProtection(signatureReq.getCertPwd().toCharArray()));
var signatureValue = signingToken.sign(dataToSign, cadesParams.getDigestAlgorithm(), certificate);

var signedDocument = service.signDocument(documentToSign, cadesParams, signatureValue);
var signature = DSSUtils.toByteArray(signedDocument);
var signedDataMimeType = signedDocument.getMimeType();

var signatureResp = new CreateSignatureResp();
signatureResp.setSignature(signature);
signatureResp.setSignatureMimeType(signedDataMimeType.getMimeTypeString());
```

Obr. 75 Ukázka kódu vytvoření podpisu CAdES (vlastní)

6.5.1.2 Vytvoření podpisu XAdES

Proces vytvoření podpisu je velice podobný jako při vytváření podpisu CAdES, jen se používají třídy týkající se podpisu XAdES. Nejprve je potřeba nastavit parametry podpisu XAdES pomocí instance třídy *XAdESSignatureParameters*. Nastaví se podpisový certifikát, typ podpisu XAdES, použitý hashovací algoritmus, umístění podpisu (externí podpis nebo podpis jako součást podepisovaných dat) a konfigurace informací o podepisujícím (pokud jsou k dispozici).

Poté se vytvoří instance třídy *InMemoryDocument* z dat, pro které bude vytvořen digitální podpis, vytvoří se instance třídy *Pkcs12SignatureToken*, který za pomocí podepisovacího certifikátu vytvoří samotný podpis, a nakonec pomocí instance třídy *XAdESService* je digitální podpis připojen k podepisovaným datům nebo je vytvořen externí podpis (podle nastavení podepisovacího profilu).

Na závěr je stejně jako u podpisu CAdES vytvořena instance třídy *CreateSignatureResp*, do které se vloží podepsaná data, MimeType podepsaných dat a přípona podepsaných dat.

```
InMemoryDocument documentToSign = new InMemoryDocument(signatureReq.getDataToSign(), signatureReq.getFileName(),
        MimeType.fromMimeTypeString(signatureReq.getFileMimeType()));

var dataToSign = service.getDataToSign(documentToSign, xadesParams);

var signingToken = new Pkcs12SignatureToken(signatureReq.getSignCertificate(),
        new KeyStore.PasswordProtection(signatureReq.getCertPwd().toCharArray()));
var signatureValue = signingToken.sign(dataToSign, xadesParams.getDigestAlgorithm(), dssCert);

var signedDocument = service.signDocument(documentToSign, xadesParams, signatureValue);
var signature = DSSUtils.toByteArray(signedDocument);
var signedDataMimeType = signedDocument.getMimeType();
```

Obr. 76 Ukázka kódu pro vytvoření podpisu XAdES (vlastní)

6.5.1.3 Vytvoření podpisu PAdES

Vytvoření podpisu PAdES je implementováno ve funkci *createPadesSignature*. Proces vytvoření podpisu PAdES je opět podobný procesu tvorby předchozích typů podpisu, jen má z důvodu možnosti vytvoření viditelného podpisu více možností konfigurace. Podpis PAdES je nejprve potřeba na parametrizovat. Je potřeba nastavit typ podpisu PAdES (B, T), hashovací algoritmus a podepisovací certifikát. Poté se můžou nastavit parametry specifické pro podpis PAdES – jméno podepisujícího, místo podepsání a důvod vytvoření podpisu. Pokud je podepisovací profil viditelný, je potřeba nastavit i všechny potřebné parametry viditelného podpisu (výška a šířka podpisu, horizontální a vertikální umístění podpisu, strana umístění, typ viditelného podpisu a podle typu buď nastavit obrázek, text či oboje). Parametry viditelného podpisu se nastavují pomocí instance třídy *SignatureImageParameters*, která je součástí třídy *PadesSignatureParameters.*

Po nastavení všech parametrů podpisu PAdES, může se přistoupit k vytvoření podpisu. Nejprve se vytvoří instance třídy *InMemoryDocument* z podepisovaných dat, vytvoří se instance třídy *Pkcs12SignatureToken* pomocí které se vytvoří samotný podpis a na závěr pomocí instance třídy *PAdESService* je podpis přidán k podepisovaným datům.

Nakonec se vytvoří instance třídy *CreateSignatureResp*, do které se vloží podepsaná data, jejich MimeType a přípona (vždy .PDF) a která je i výstupem funkce *createPadesSignature*.

```
var documentToSign = new InMemoryDocument(signatureReq.getDataToSign(), signatureReq.getFileName(),
        MimeType.fromMimeTypeString(signatureReq.getFileMimeType()));

var dataToSign = padesService.getDataToSign(documentToSign, padesParams);

var signingToken = new Pkcs12SignatureToken(signatureReq.getSignCertificate(),
        new KeyStore.PasswordProtection(signatureReq.getCertPwd().toCharArray()));
var signatureValue = signingToken.sign(dataToSign, padesParams.getDigestAlgorithm(), dssCert);

var signedDocument = padesService.signDocument(documentToSign, padesParams, signatureValue);
```

Obr. 77 Ukázka kódu vytváření podpisu PAdES (vlastní)

6.5.1.4 Implementace funkce pro obsluhu požadavku pro vytvoření dig. podpisu

Funkce *createSignature* se poté volá ve třídě kontroléru *SignatureController* a její funkci *createSignature*, která se provede jako obsluha http POST požadavků s koncovou adresou */signature/create*. Tato funkce v těle požadavku vyžaduje přítomnost objektu *CreateSignatureReq*, který obsahuje následující data:

- Id použitého profilu pro vytvoření podpisu,

- Id uživatele
- Data k podepsání jako pole bytů
- Jméno souboru podepisovaných dat
- MimeType podepisovaných dat
- Příponu souboru podepisovaných dat
- Podepisovací certifikát ve formátu PKCS#12
- Heslo k certifikátu
- MimeType souboru certifikátu
- Příznak, zda se přidat záznam do systému blockchain

Před zahájením vytváření digitálního podpisu se data nejprve validují. Validace probíhá ve třídě *CreateSignatureValidator* a její funkci *validateCreateSignatureReq*. V případě chyby validace je na FE navrácena chyba s detailem problému, který při validaci nastal. Po úspěšné validaci se poté vytvoří samotný digitální podpis pomocí třídy *SignatureService* a funkce *createSignature*. Podepsaná data, která funkce vrátí, jsou poté vrácena zpět na klientskou část.

6.5.2 Vytvoření digitálního podpisu na FE

Při implementaci vytvoření digitálního podpisu na FE bylo potřeba vytvořit novou obrazovku pro provedení procesem vytvoření digitálního podpisu a třídu, která vytvoří http požadavek na vytvoření digitálního podpisu a odešle jej na BE část aplikace.

Samotná obrazovka pro vytvoření digitálního podpisu je rozdělena na tři části. Jednotlivé části jsou vytvořeny pomocí komponenty *Steps*, podobně jako v obrazovce pro vytvoření / editaci existujícího profilu. V komponentě jsou definovány tři kroky – výběr podepisovacího profilu, výběr dat k podepsání a poté uložení vytvořeného digitálního podpisu. Definice jednotlivých částí se poté nachází v komponentách *StepPanel*.

První část obrazovky tedy slouží pro výběr podepisovacího profilu, který se použije při vytváření digitálního podpisu. Proto nejprve při zobrazování obrazovky jsou získány profily uživatele pomocí třídy *ProfileService* a její metody *GetProfilesAsync*. Stažené profily jsou zobrazeny pomocí komponenty *ListGroup*, která disponuje funkcí výběru položky ze seznamu. Toho je využito pro samotný výběr profilu.

```
<ListGroup Mode="ListGroupMode.Selectable" @bind-SelectedItem="SelectedProfileId">
    @foreach (var profile in profiles)
    {
        <ListGroupItem ElementId="@profile.Id.ToString()" Name="@profile.Id.ToString()">
            <Div>
                <Heading Size="HeadingSize.Is4">@profile.Name</Heading>
                <Heading Size="HeadingSize.Is5">(@profile.ProfileType)</Heading>
            </Div>
        </ListGroupItem>
    }
</ListGroup>
```

Obr. 78 Zobrazení seznamu podepisovacích profilů pomocí *ListGroup* (vlastní)

Druhá část obrazovky se zabývá výběrem potřebných dat pro vytvoření digitálního podpisu. Je zde dostupné pole pro výběr souboru, který se má digitálně podepsat. Poli pro výběr souboru je možno definovat filtr povolených přípon souborů. Zde je filtr dynamicky měněn na základě vybraného podepisovacího profilu a typu podpisu pro který je podpis vytvářen. Tato akce je vytvořena z důvodu, aby uživatel nemohl vybrat data, která by nešla pomocí daného profilu podepsat. Např. pokud je vybraný podepisovací profil PAdES, lze podepisovat pouze PDF dokumenty apod.

V obrazovce se dále nachází pole pro výběr souboru podepisovacího certifikátu ve formátu PKCS#12, pole pro zadání hesla k certifikátu a poté příznak, zda se má po vytvoření digitálního podpisu nahrát jeho záznam do systému blockchain. Proces vytvoření digitálního podpisu lze zahájit pomocí tlačítka Podepsat, které je součástí druhé části obrazovky.

Třetí část obrazovky se zobrazí po úspěšném vytvoření digitálního podpisu. V této části je informace o úspěšném vytvoření digitálního podpisu a poté tlačítko pro jeho uložení. Při ukládání digitálního podpisu dojde ke stažení podpisu do počítače.

Pro samotné vytvoření požadavku a odeslání na BE část aplikace byla vytvořena třída *SignatureService* a metoda *CreateSignatureAsync*, která jako vstupní parametr vyžaduje instanci třídy *CreateSignatureReq*. Tato třída v sobě uchovává data souboru k podepsání, id vybraného profilu, id přihlášeného uživatele, základní informace o podepisovaném souboru, podepisovací certifikát, heslo k podepisovacímu certifikátu a příznak, zda se má podepsaný soubor vložit do systému blockchain. Tato třída je obdobou třídy *CreateSignatureReq* v projektu BE.

V metodě *CreateSignature* se poté získá JWT token přihlášeného uživatele, vytvoří se http POST požadavek s autorizační hlavičkou, do těla požadavku se vloží instance třídy *CreateSignatureReq* převedená do formátu JSON a odešle se na BE aplikace. Po přijetí odpovědi

poté zjistí status odpovědi. V případě úspěchu je tělo http odpovědi převedeno z JSON do instance třídy *CreateSignatureResp*, která obsahuje podepsaná data, jejich MimeType a jejich příponu. V případě chyby je z těla http odpovědi získán detail chyby, která je následně zobrazena uživateli.

```
var token = await _localStorage.GetItemAsync<string>(JwtParser.JwtTokenKeyName);
var userId = JwtParser.GetUserIdFromToken(token);
if (!string.IsNullOrWhiteSpace(token) && !string.IsNullOrWhiteSpace(userId))
{
    var request = new HttpRequestMessage(HttpMethod.Post, CertiChainApiConstants.SignatureCreateUri);
    createSignatureReq.UserId = userId;
    var json = JsonConvert.SerializeObject(createSignatureReq);
    request.Content = new StringContent(json, Encoding.UTF8, CertiChainApiConstants.JsonMediaType);
    request.Headers.Authorization = new AuthenticationHeaderValue(CertiChainApiConstants.BearerScheme, token);

    var response = await _httpClient.SendAsync(request);
    if (response.IsSuccessStatusCode)
    {
        var jsonResp = await response.Content.ReadAsStringAsync();
        Console.WriteLine(jsonResp);
        return JsonConvert.DeserializeObject<CreateSignatureResp>(jsonResp);
    }
    await _httpClient.ParseErrorResp(response);
}
```

Obr. 79 Metoda odesílající požadavek na vytvoření dig. podpisu na BE (vlastní)

6.6 Ověření digitálního podpisu

Implementace ověření digitálního podpisu také zasáhla do obou částí aplikace CertiChain. Na straně FE byla vytvořena nová obrazovka pro provedení uživatele procesem ověření, která je dostupná z domovské obrazovky přihlášeného uživatele a na straně BE byl implementováno samotné ověření digitálního podpisu pomocí tříd z doplňku DSS.

6.6.1 Implementace ověření podpisu na BE

Implementace procesu ověření digitálního podpisu byla naimplementována přidáním nových funkcí do již existujících tříd *SignatureService* a *SignatureController*. Ve třídě *SignatureService* je implementován celý proces ověření podpisu a ve třídě *SignatureController* je přidána funkce pro obsloužení cílového http požadavku.

Funkce ve třídě *SignatureController* pro ověření podpisu se nazývá *verifySignature*. Tato funkce je zavolána jako obsluha http POST požadavků s koncovou adresou */signature/verify*. V těle požadavku se nacházejí všechna potřebná data pro provedení ověření podpisu ve formátu JSON, která jsou na pozadí převedena na instanci třídy *VerifySignatureReq*, která obsahuje následující proměnné:

- Podepsaná data a potřebná metadata (jméno, MimeType). V případě že se ověřuje externí podpis, jedná se data, ke kterým byl vytvořen digitální podpis.
- Pokud se ověřuje externí podpis, obsahuje i data a metadata externího podpisu (název souboru a MimeType)

Před zahájením samotného ověření digitálního podpisu jsou nejprve data validována. Validace probíhá ve třídě *VerifySignatureValidator* ve funkci *validateVerifySignature*. Validátor kontroluje žádost, zda obsahuje všechna potřebná data. Po úspěšné validaci je zavolána funkce *verifySignature* ze třídy *SignatureService*, kde je implementováno samotné ověření digitálního podpisu.

Samotná implementace ověřování podpisu je jednodušší oproti vytváření v tom, že při ověřování podpisu není potřeba každý typ podpisu ověřovat ve speciální třídě, ale stačí použít jednu obecnou třídu z knihovny DSS, která umožňuje ověřit všechny druhy podpisu podporované v aplikaci CertiChain.

Samotné ověření digitálního podpisu provádí instance třídy *SignedDocumentValidator*. Tuto instanci je ale nejprve potřeba parametrizovat pomocí instance třídy *CommonCertifierVerifier*, které je potřeba nastavit zdroj CRL certifikátu, OCSP zdroj, zdroj pro získání důvěryhodných certifikátů. Na všechny tyto parametry existují v DSS třídy, které tyto parametry nastaví z veřejných online zdrojů.

```
var commonVerifier = new CommonCertificateVerifier();
commonVerifier.setCrlSource(new OnlineCRLSource());
commonVerifier.setOcspSource(new OnlineOCSPSource());
//commonVerifier.setAIASource(new DefaultAIASOurce());

commonVerifier.addTrustedCertSources(certTrustedList);
```

Obr. 80 Parametrizace ověřovače podpisu (vlastní)

Další nastavení validátoru záleží na typu umístění podpisu, tedy jestli je podpis součástí podepisovaných dat, nebo je uložen jako soubor zvlášť. Pokud je podpis součástí podepisovaných dat, stačí vytvořit instanci samotného validátoru pomocí třídy *SignedDocumentValidator* a její statické funkce *fromDocument*. Pokud se jedná o externí podpis, je potřeba instanci vytvořit z dat externího podpisu (opět pomocí statické funkce *fromDocument)*, ale ještě je potřeba přidat do instance originální data, ke kterým byl podpis vytvořen.

```
SignedDocumentValidator validator = null;
if (req.getExternalSignature() == null) {
    validator = SignedDocumentValidator.fromDocument(verifiedDoc);
} else {
    var extSig = new InMemoryDocument(req.getExternalSignature(), req.getExternalSignatureFileName(),
            MimeType.fromMimeTypeString(req.getExternalSignatureMimeType()));
    validator = SignedDocumentValidator.fromDocument(extSig);
    var detachedData = new ArrayList<DSSDocument>();
    detachedData.add(verifiedDoc);

    validator.setDetachedContents(detachedData);
}
```

Obr. 81 Vytvoření instance validátoru (vlastní)

Po všech výše pospaných konfiguracích může být spuštěn proces samotného ověření. Po dokončení ověření je získán report výsledku, ze kterého se získají nejdůležitější data z ověření (výsledek ověření, čas ověření, čas vytvoření podpisu, informace o podepisovacím certifikátu, chyby a varování nalezené při ověření atd.). Tyto informace jsou vloženy do instance třídy *VerifySignatureResp* a jsou navráceny zpět do funkce *verifySignature* ve třídě *SignatureController*, která je poté vrátí zpět na FE.

6.6.2 Implementace ověření podpisu na FE

Na FE byla vytvořena pro ověření digitálního podpisu nová obrazovka, která uživatele procesem ověření provede. Obrazovka byla opět rozdělena na tři části pomocí komponenty *Steps.*

První část slouží pro výběr dat k ověření. Část obrazovky obsahuje pole pro výběr souboru podepsaných (v případě externího podpisu podepisovaných) dat. Další pole slouží pro vložení souboru externího podpisu. Pomocí tlačítka Ověřit podpis, je po úspěšné validaci dat zahájen samotný proces ověření.

Zahájení procesu ověření přesune uživatele na druhou část obrazovky, kde je zobrazena animace načítání a informace, že probíhá proces ověřování. Ověření může trvat i desítky sekund, proto byl zvolen tento způsob implementace procesu čekání.

Po dokončení ověření je uživateli zobrazen výsledek ověření digitálních podpisů (dokument může obsahovat vícero digitálních podpisů). Podle výsledku ověření jsou informace zobrazeny v podbarveném rámečku (zelená – úspěšné ověření, žlutá – ověření skončilo nejistým výsledkem, červená – ověření selhalo).

Pro implementaci komunikace s BE byla přidána do třídy *SignatureService* metoda *VefiySignatureAsync.* Ta vytvoří http POST požadavek s autorizační hlavičkou, do těla

požadavku vloží potřebná data k ověření (viz druhý odstavec kapitoly 6.6.1) a odešle jej na BE. Po obdržení odpovědi od BE zjistí výsledek podle statusu odpovědi a případě úspěchu získá z těla informace výsledku ověření a předá je zpět do obrazovky ověření, která tyto informace zobrazí. V případě neúspěchu jsou získány detaily chyby, a ty jsou poté zobrazeny uživateli.

6.7 Zakomponování systému Blockchain

Implementaci zakomponování použití systému blockchain v aplikaci lze rozdělit do tří částí – přidání záznamu do blockchain, ověření záznamu v blockchain a implementace přehledu přidaných záznamů v blockchain.

6.7.1 Implementace přehledu záznamů v systému blockchain.

Přehled záznamů v blockchain umožňuje uživateli podívat se, kolik, transakcí již přidal do systému blockchain. V přehledu nejsou uloženy kompletní informace o transakci, ale jen základní informace jako id transakce, kdy daný záznam přidal začátek a konec otisku v transakci a id uživatele, který záznam vytvořil pro budoucí vyhledávání v databázi. Poté může danou transakci vyhledat v systému blockchain a zkontrolovat, zda je validní. Na BE části aplikace bylo potřeba přidat novou databázovou entitu, CRUD operace nad entitou, a nakonec vytvořit třídu kontroléru, který umožní získat záznamy a zkontrolovat stav požadovaného záznamu. Na FE části bylo potřeba naimplementovat novou obrazovku pro přehled záznamů v blockchain.

6.7.1.1 Implementace přehledu záznamů v blockchain na BE

Nejprve bylo potřeba vytvořit samotnou tabulku v databázi. Proto byla vytvořena entita *BlockchainRecordEntity* s anotací *@Entity* a poté rozhraní *BlockchainRecordRepository* s anotací *@Repository*, které implementuje samotnou databázovou tabulku.

```
@Entity
public class BlockchainRecordEntity {
    @Getter @Setter @Id @GeneratedValue private Integer id;
    @Getter @Setter private UUID userId;
    @Getter @Setter private String creationDate;
    @Getter @Setter private String hash;
    @Getter @Setter private UUID proofId;
}
```

Obr. 82 Entita přehledu záznamu v blockchain (vlastní)

Dále byla vytvořena třída *BlockchainService*, do které bylo potřeba naimplementovat funkce pro přidání záznamu (*addNewRecordToDB)*, získání záznamů uživatele na základě jeho

veřejného id (*getUserRecords*), smazání záznamů z databáze (*deleteUserRecords*), která je použita v případě smazání uživatele a ověření stavu transakce v blockchain (*getProof*).

Na závěr byla implementována třída kontroléru *BlockchainController* s anotací *@RestController* se dvěma funkcemi – *getUserRecords* pro získání záznamů pro uživatele podle jeho veřejného id a *verifyRecord* pro ověření transakce v blockchain podle jejího id.

Funkce *getUserRecords* je zavolána při obsluze http GET požadavků s koncovou adresou */blockchain/getUserRecords/{userId}*, kdy *userId* je dynamický parametr s veřejným id uživatele. Ve funkci je poté volána funkce pro získání uživatelských záznamů ze třídy *BlockchainService* a po získání jsou vloženy do instance třídy *GeUserBlockchainRecords*, která je ve formátu JSON zaslána zpět na FE.

Druhá funkce kontroléru pro ověření záznamu *verifyRecord* je volána při obsluze požadavků http GET s koncovou adresou */blockchain/verifyRecord/{recordId}*, kdy *recordId* je id daného záznamu. Při zpracování požadavku se získají informace o transakci, ve kterých se nachází i stav transakce (zda je validní, nevalidní, nebo stále čeká na přidání do blockchain apod.) pomocí speciálního příkazu *getProof*, který databáze ProvenDB obsahuje. Získaný stav transakce je poté vložen do instance třídy *VerifyRecordResp*, která je zaslána ve formátu JSON zpět na FE.

6.7.1.2 Implementace přehledu záznamů na FE

Implementace přehledu záznamů na FE zahrnovala vytvoření obrazovky, ve které budou jednotlivé záznamy zobrazeny a poté vytvoření třídy, která bude implementovat komunikaci s BE částí CertiChain.

Pro komunikaci s BE částí je vytvořena na FE třída *BlockchainService*, která obsahuje metody *GetUserRecordsAsync* a *CheckRecordValidityAsync*. Metoda *GetUserRecordsAsync* vytvoří http GET požadavek, přidá do něj autorizační hlavičku s JWT tokenem a koncovou adresu požadavku na hodnotu */blockchain/getUserRecords/{userId},* kde *userId* je veřejné id přihlášeného uživatele. Poté požadavek odešle na BE a po přijetí odpovědi na základě statusu odpovědi v případě úspěchu získá z těla odpovědi informace o záznamech v blockchain a vrátí je jako návratovou metodu funkce. V případě chyby se z těla odpovědi získají informace o chybě a následně se zobrazí uživateli.

Metoda *CheckRecordValidityAsync* také vytvoří http GET požadavek, vloží do něj autorizační hlavičku s JWT tokenem, nastaví koncovou adresu požadavku na hodnotu

/blockchain/verifyRecord/{recordId}, kde {recordId} se nahradí id vybraného záznamu v blockchain a odešle požadavek na BE. Po přijetí odpovědi z BE se zkontroluje stav odpovědi a v případě úspěchu se z těla odpovědi získá stav kontrolovaného záznamu. Poté je stav zobrazen uživateli pomocí komponenty Toast. V případě neúspěchu je získán z těla odpovědi detail chyby, který se také uživateli zobrazí pomocí komponenty Toast.

Samotná obrazovka pro vizualizaci přehledů se skládá z tabulky, ve které jsou zobrazeny jednotlivé záznamy. Pro případ velkého počtu záznamů je v obrazovce implementována funkce filtrování, kde si lze parametrizovat zobrazení záznamů podle jejich data vytvoření. Při filtrování se z kolekce záznamů v blockchain vyberou pouze položky, které splňují podmínky filtrování.

6.7.2 Přidání záznamu do blockchain

Záznam do blockchain je vždy potřeba přidat ihned po vytvoření digitálního podpisu. V kapitole 6.5.1.3 je napsáno, že při odesílání požadavku na vytvoření dig. podpisu je posílán i příznak, zda se má přidat záznam do systému blockchain. Pokud je tento příznak nastaven, po vytvoření dig. podpisu v kontroléru *SignatureService* ve funkci *createSignature* je nově voláno ještě přidání záznamu do systému blockchain.

Pro samotné přidání záznamu do systému blockchain byla do třídy *BlockchainService* přidána funkce *addRecord*. Funkce na svém vstupu požaduje podepsaná data, externí podpis, pokud je k dispozici, a podepisovací certifikát. Poté jsou z těchto dat vypočteny otisky. Z těchto otisků je vytvořen JSON dokument, který se vloží do databáze ProvenDB. Přidat záznam do databáze ProvenDB lze udělat pomocí klasického příkazu pro přidání záznamu do MongoDb. Na proces přidání je využita knihovna pro komunikaci s databází MongoDb.

```
// přidání záznamu do databáze
var reqDoc = Document.parse("{" +
        "signedDataHash: \"" + signedDataDigest + "\"," +
        "externalDataHash: \"" + externalSignatureDigest + "\"," +
        "signCertHash: \"" + certDigest + "\"" +
        "}");

var collection = mongoDb.getCollection("provendb");
collection.insertOne(reqDoc);
```

Obr. 83 Přidání záznamu s otisky dat do ProvenDB (vlastní)

Přidání nového záznamu zapříčiní vznik nové verze databáze (viz kapitola 5.3.2.2), kterou je možné uložit do blockchain. Pro vytvoření záznamu v blockchain je potřeba použít

speciální příkaz databáze ProvenDB *submitProof*. Tento příkaz lze spustit pomocí funkce *runCommand*, která je k dispozici po vytvoření instance třídy *MongoDatabase*, která implementuje připojení k samotné ProvenDB databázi. Po vytvoření a odeslání transakce do blockchain pomocí příkazu *submitProof* je navrácena informace o transakci, tedy její stav, id, otisk, čas vytvoření a spoustu dalších metadat, která pro uživatele nenesou žádnou informační hodnotu. Tyto údaje jsou vloženy do instance třídy *CreateTransactionResp* a jsou navráceny zpět do funkce *createSignature* kontroléru *SignatureController*.

6.8 Ověření přítomnosti záznamu v blockchain

Ověření přítomnosti záznamu v systému blockchain se může provést při ověřování digitálního podpisu. V první části obrazovky ověření dig. podpisu byla přidána možnost vyžádání ověření záznamu i v systému blockchain. Aby bylo možno záznam ověřit v blockchain, je potřeba do žádosti o ověření přidat i podepisovací certifikát, kdy při ověřování certifikát již nemusí být ve formátu PKCS#12. Pro výběr souboru certifikátu bylo do obrazovky přidáno pole pro výběr souboru certifikátu a pokud je vybraný soubor certifikátu ve formátu PKCS#12, je v obrazovce dostupné i pole pro zadání hesla.

Nově zmíněné položky (příznak pro ověření v blockchain, soubor certifikátu, MimeType souboru certifikátu a případně heslo k certifikátu) poté byly přidány i do tříd *VerifySignatureReq* (viz kapitola 6.6.1 pro BE a kapitola 6.6.2 pro FE). Na BE bylo ověření v systému blockchain doimplementováno do funkce ověření dig. podpisu *verifySignature* ve třídě *SignatureController*.

Pokud je nastaven příznak na ověření blockchain, tak se po ověření digitálního podpisu pomocí knihoven DSS provede ještě ověření v systému blockchain. Samotný proces ověření je implementován ve třídě *BlockchainService* a v její funkci *verifySignatureInBlockchain*, která přebírá jako parametr instanci třídy *VerifySignatureReq*.

Funkce *verifySignatureInBlockchain* nejprve získá otisky podepsaných (podepisovaných, pokud se ověřuje externí podpis) dat, externího podpisu (pokud je k dispozici) a podepisovacího certifikátu. Databáze ProvenDB obsahuje speciální příkaz *getDocumentProof* pro vyhledání informace, zda je v databázi určitý záznam a verze této databáze je v systému blockchain. Jedním z parametrů tohoto příkazu je i možnost nastavení filtru podle kterého se má záznam vyhledat. Tomuto filtru se tedy nastaví vypočtené otisky a příkaz se spustí.

Po získání výsledku je navrácená odpověď převedena z formátu JSON na objekt a objekt se poté analyzuje, zda obsahuje informace o záznamu v blockchain. Pokud ano, znamená to, že v databázi existuje záznam s danými otisky v databázi a verze databáze je v systému blockchain.

Z informací o nalezeném záznamu v blockchain se získají základní informace (id, stav záznamu, čas odeslání záznamu a otisk umístěný v záznamu), které se vloží do instance třídy *VerifyInBlockchainResp* a vrátí se zpět do funkce *verifySignature* v kontroléru, data se vloží do výsledku ověření digitálního podpisu a ten je poté poslán zpět na FE.

Po navrácení odpovědi na FE je poté ve výsledku ověření vizuálně zobrazen výsledek ověření v blockchain.

7 TESTOVÁNÍ APLIKACE CERTICHAIN

Po dokončení implementace všech funkcionalit aplikace je bylo potřeba otestovat. Při testování aplikace bylo využito jak automatických, tak i manuálních testů. Automatické testy byly naimplementovány v jazyce Python pomocí knihoven RobotFramework. Reporty a logy všech provedených automatických testů jsou součástí příloh DP.

Automatické testy jsou využity pouze pro obecné situace, které nejsou příliš dynamické a automatický test bylo možné provést vždy, aniž by bylo potřeba s ním manipulovat. Situace, které by bylo příliš komplikované automaticky otestovat, byly testovány manuálně.

7.1 Automatické testovací sady

Automatické testovací scénáře zahrnují testovací sady pro registraci uživatele, přihlášení uživatele, správu uživatelských profilů, vytvoření a ověření dig. podpisu a zobrazení přehledu záznamů v blockchain.

Všechny testovací sady kromě přihlášení a registrace mají při testování vstupní prerekvizitu, aby byl v aplikaci přihlášen uživatel. Proto byl pro testy v aplikaci vytvořen speciální účet, kterému byla vytvořena i základní sada podepisovacích profilů a vytvořena i základní sada digitálních podpisů a záznamů v blockchain a tato data jsou poté použita při samotných automatických testech.

Dále byly všechny potřebné XPaths jednotlivých prvků přidány jako proměnné do testovacího projektu, aby se k nim mohlo v testovacích scénářích přistupovat pomocí daných proměnných namísto kopírování. Tento přístup byl zvolen z důvodu snadnějších oprav špatných XPpath, kdy takhle stačí XPath upravit na jednom místě v projektu.

7.1.1 Testovací sada pro registraci uživatele

Testovací sada pro registraci uživatele má dohromady pět testovacích případů. Testuje je úspěšná registrace nového uživatele, kde nastává problém s potřebnou editací vstupních dat testovacího případu, kdy při každém spuštění tohoto testovacího případu je potřeba měnit vstupní data, protože aplikace neumožňuje registrovat dva uživatele na jednu emailovou adresu. Další testovací případy testují kontrolu vyplnění údajů pro registraci, kdy jsou otestovány jednotlivé varianty nevyplnění nějaké položky, použití slabého hesla, špatné zopakování hesla apod. Poslední testovací případ se zabývá pokusem registrace do aplikace emailovou adresou, která je v aplikaci již registrována.

```
TC_01_02_02 - Not Filled Form
    Open Chrome Page Go Register
    click button  ${BtnRegister}
    wait until element is visible  ${RegisterEmailRequiredElement}
    wait until element is visible  ${RegisterPwdRequiredElement}
    wait until element is visible  ${RegisterPwdAgainRequired}
```

Obr. 84 Zdrojový kód testovacího případu nevyplněného formuláře (vlastní)

7.1.2 Testovací sada pro přihlášení uživatele

Testovací sada pro přihlášení uživatele obsahuje tři testovací případy. První testovací případ se zabývá správným přihlášením uživatele do systému a zbylé dva testovací případy testují přihlášení do systému pomocí neexistujícího emailu nebo použitím špatného hesla pro přihlášení registrované emailové adresy.

```
TC_01_01_01 - Success Login
    Open Chrome And Page
    Click Link  ${NavBarLogin}
    Wait Until Element Is Visible  ${LoginTitle}
    Put Text Into Field  ${LoginMailVal}  ${LoginEmail}  ${LoginTitle}
    Put Text Into Field  ${LoginPwdVal}  ${LoginPwd}  ${LoginTitle}
    Click Button  ${BtnLogin}
    Wait Until Element Is Visible  ${UserHomeElement}
```

Obr. 85 Ukázka testovacího scénáře pro úspěšné přihlášení do aplikace (vlastní)

7.1.3 Testovací sada pro správu podepisovacích profilů

Testovací sada pro správu uživatelských profilů obsahuje pouze základní testy pro vytvoření podepisovacích profilů pro každý typ podpisu. Testování aktualizace profilů a mazání by bylo z důvodů příliš mnoha dynamických faktorů složité implementovat, proto jsou tyto situace otestovány pomocí manuálních testovacích sad.

Testovací sada tedy obsahuje pouze tři testovací případy, kdy v každém testovacím případu je vytvořen podepisovací profil pro každý podporovaný typ podpisu (PAdES, CAdES, XAdES).

7.1.4 Testovací sada pro vytvoření digitálního podpisu

Sada pro testování vytvoření digitálního podpisu se skládá z devíti testovacích scénářů. Pro úspěšné provedení těchto testů je předpřipravena sada testovacích podepisovacích profilů, které jsou využívány v testovacích případech a sada testovacích souborů, ke kterým bude vytvářen digitální podpis. Podepisovací certifikát je z důvodů bezpečnosti vždy potřeba použít svůj vlastní. Před spuštěním testovací sady je potřeba nastavit cestu k

vlastnímu podepisovacímu certifikátu a heslo k podepisovacímu certifikátu do proměnných v souboru *CustomVariables.robot*.

Testovací případy jsou poté zaměřeny na úspěšné vytvoření digitálního podpisu pro všechny podporované typy podpisu. Dále jsou připraveny testovací případy, které testují chování systému při odeslání nevalidní žádosti o vytvoření digitálního podpisu (chybí data, není vybrán podepisovací certifikát, špatné heslo k certifikátu apod.). Poslední testovací případy poté testují vytvoření digitálního podpisu včetně přidání záznamu do blockchain.

7.1.5 Testovací sada pro ověření digitálních podpisů

Pro úspěšné provedení testovacích případů uvnitř této testovací sady byla do testovacího projektu přidána sada testovacích podepsaných dat, která se využívají jako vstupní data při vytváření žádosti o ověření dig. podpisu.

Testovací sada obsahuje devět testovacích případů, kdy testovací případy jsou zaměřeny na provedení ověření všech podporovaných typů podpisů, dále je testováno chování aplikace při nenastavení všech potřebných vstupů žádostí o ověření a poslední testovací případy se zabývají ověřením digitálního podpisu včetně kontroly přítomnosti záznamu v blockchain.

7.1.6 Testovací sada pro přehled záznamů v blockchain

Pro úspěšné provedení testů je potřeba, aby měl testovací uživatel alespoň pár vytvořených záznamů v systému blockchain, jak je zmíněno v kapitole 7.1. Testovací sada poté obsahuje pár testů pro ověření základní funkčnosti obrazovky.

Testovací sada obsahuje testovací případy pro zobrazení přehledu záznamů v blockchain, poté testovací případ pro ověření stavu záznamu v blockchain a testovací případy pro ověření funkčnosti filtrování záznamů.

7.2 Manuální testovací scénáře

Jak bylo zmíněno v úvodu 7. kapitoly, v aplikaci se vyskytují části, které nelze úplně jednoduše implementovat pomocí automatických testů, ale lze jejich správnost otestovat rychle i manuálně. Jedná se zejména o funkcionality editace a mazání podepisovacích profilů a poté administrace uživatelského účtu.

Při editaci profilu nelze nějak jednoduše v automatických testech přejít na samotnou editaci profilu, protože všechny položky v seznamu profilů mají tlačítko „Upravit“, a proto by

nebylo úplně triviální vybrat vždy potřebný profil, ne bez zásahu do zdrojového kódu obrazovky a implementace speciálního nastavování id jednotlivých tlačítek u profilů. Proto byly tyto funkcionality vždy testovány manuálně.

V administraci účtu lze testovat změnu hesla a samotnou deaktivaci účtu. Tyto funkcionality je opět jednodušší testovat manuálně, protože hlavně pro účely deaktivace účtu je vhodnější vytvořit si nový účet a na něm testovat změnu hesla a deaktivaci, protože při použití testovacího účtu z automatických testů by se při deaktivaci o celý účet přišlo a bylo by potřeba vždy připravit nový účet a vytvořit mu podepisovací profily a záznamy blockchain, aby se opětovně mohly spustit manuální testy.

8 VÝSLEDNÁ PREZENTACE APLIKACE

Po dokončení celého procesu vývoje aplikace zbývá aplikaci nasadit na server, webové aplikaci nastavit doménové jméno, zařídit přístup z internetu, a nakonec přidat aplikaci SSL certifikát, aby běžela na šifrovaném https protokolu. Po nasazení aplikace lze ukázat základní použití aplikace.

8.1 Nasazení aplikace

Pro nasazení aplikace byl vybrán stroj s operačním systémem Ubuntu. Na stroj byly nainstalovány všechny potřebné nástroje – Docker, GIT, .NET SDK, Java JDK, Maven, aby bylo možné si naklonovat projekt aplikace a spustit na zařízení všechny potřebné služby. V Dockeru poběží systémy MySQL a ProvenDB a pomocí maven a .NET SDK budou sestaveny a spuštěny obě částí aplikace CertiChain.

Druhá část nasazování aplikace spočívá v nastavení sítě, konkrétně bylo potřeba nastavit, aby po přijetí požadavku na výchozí bod (typicky Wifi router nebo switch) domácí sítě byl požadavek přesměrován právě na zařízení, na kterém běží aplikace CertiChain. Při nasazování serveru pro aplikaci CertiChain bylo přesměrování implementováno tak, že se v nastavení routeru nejdříve serveru vyhradila IP adresa v síti. Poté pomocí směrování byly vytvořeny v nastavení routeru pravidla pro přesměrování požadavků, které když cílí na výchozí bod sítě, tak se přepošlou na zarezervovanou IP adresu serveru. Tímto způsobem bylo zařízeno, že nasazená aplikace na serveru bude dostupná i z veřejné sítě.

Po dokončení druhé části aplikace sice byla dostupná z veřejné sítě, ale aby k ní uživatel mohl přistoupit, musel znát IP adresu výchozího bodu sítě ve které je připojen server. Tento problém byl vyřešen vytvořením doménového jména na DNS server, který automaticky převede doménové jméno na cílovou IP adresu. Pro vytvoření doménového jména byla využita služba DuckDNS.org, která registrovaným uživatelům umožňuje vytvořit si až tři doménové jména zdarma s tím že konec jména domény bude obsahovat „.duckdns.org“. Pomocí DuckDNS.org bylo vytvořeno doménové jméno „certichain.duckdns.org“, kdy po zadání této adresy do prohlížeče je uživatel přesměrován na IP adresu sítě, ve které je připojen server s aplikací CertiChain.

Poslední část nasazení se týkala zprovoznění komunikace pomocí https. Nejprve bylo potřeba zajistit SSL certifikát. SSL certifikát byl zřízen u poskytovatele ZeroSSL, který nabízí zdarma SSL certifikát s platností na tři měsíce, což pro vyzkoušení CertiChain bude stačit.

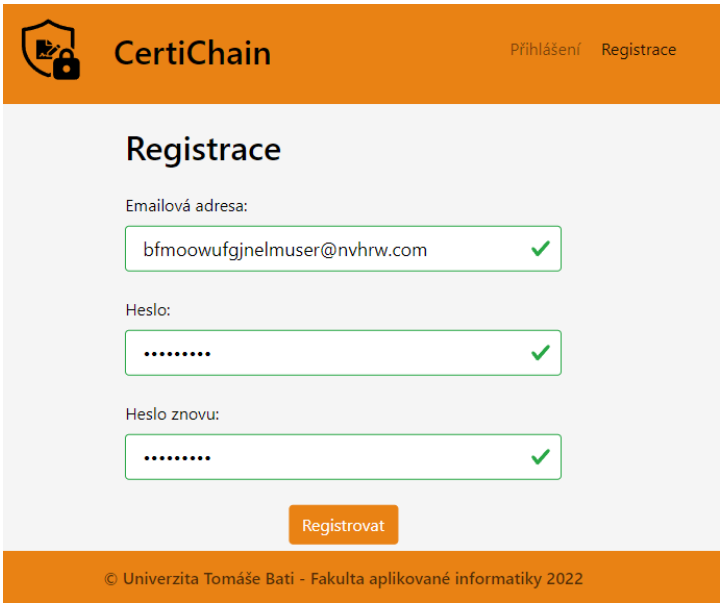
Získaný certifikát byl poté uložen na server, kde běží aplikace CertiChain a v konfiguracích projektu FE a BE byla povolena htts komunikace a nastavena cesta k cílovému SSL certifikátu.

Poté stačilo již pouze spustit databázi MySQL a ProvenDB pomocí Dockeru, sestavit a spustit obě části aplikace a k přistoupit k aplikaci pomocí https protokolu při zadání URL adresy *https://certichain.duckdns.org.*

8.2 Ukázka použití aplikace

Při přejití na URL adresu aplikace (*https://certichain.duckdns.org*) se uživateli zobrazí úvodní strana aplikace. Pokud není uživatel ještě přihlášen, v horním navigačním menu aplikace jsou dostupná tlačítka pro přihlášení nebo registraci do systému. V případě, že je uživatel přihlášen má místo těchto tlačítek k dispozici své přihlašovací jméno s rozbalovacím menu, kde může přejít na rozcestník dostupných funkcionalit nebo se odhlásit.

V případě prvního použití aplikace je potřeba nejprve se registrovat. Při registraci se zadá emailová adresa, heslo pro přihlášení a heslo pro zopakování z důvodu vyhnutí se chybně zadaného hesla. Po zadání údajů je možno provést registraci pomocí tlačítka *Registrovat.*

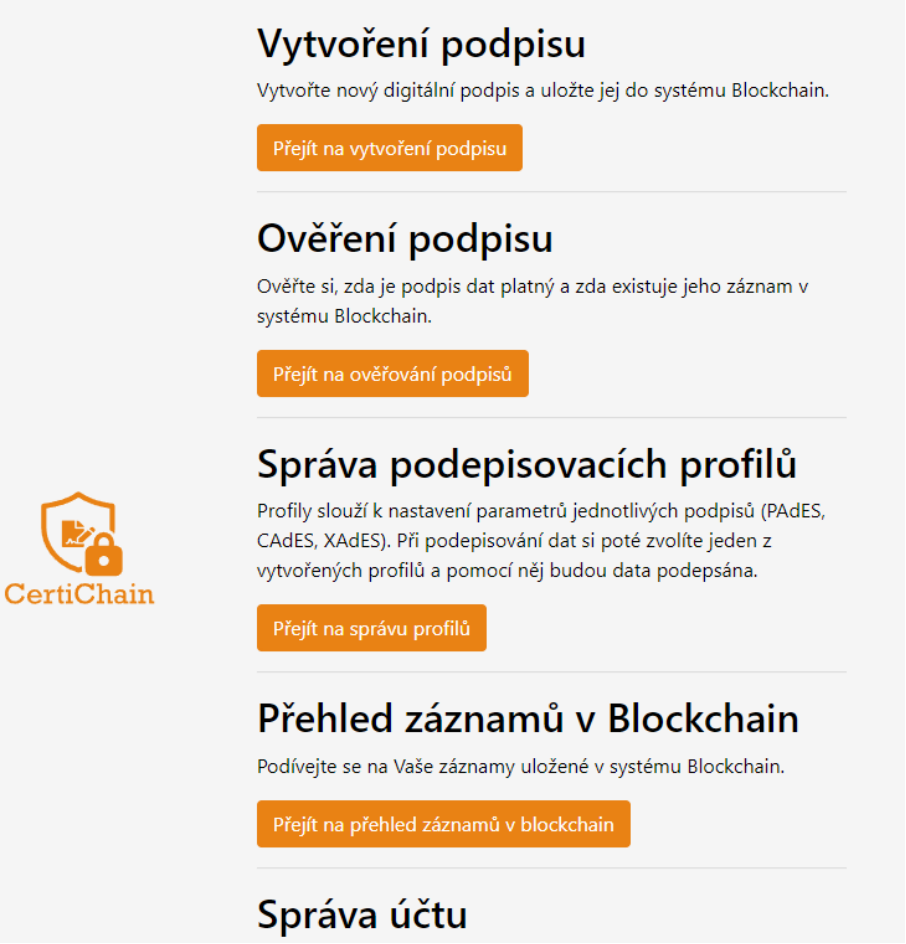


Obr. 86 Registrace uživatele (vlastní)

Po úspěšné registraci je uživatel ihned i přihlášen a již může využívat funkcionality aplikace. Po registraci by měl na registrovanou emailovou adresu dorazit email o potvrzení registrace a emailové adresy. Může nastat situace, že ověřovací emaily nebudou doručeny, protože aplikace používá SMTP server od Google a také je pro aplikaci vytvořen emailový účet od

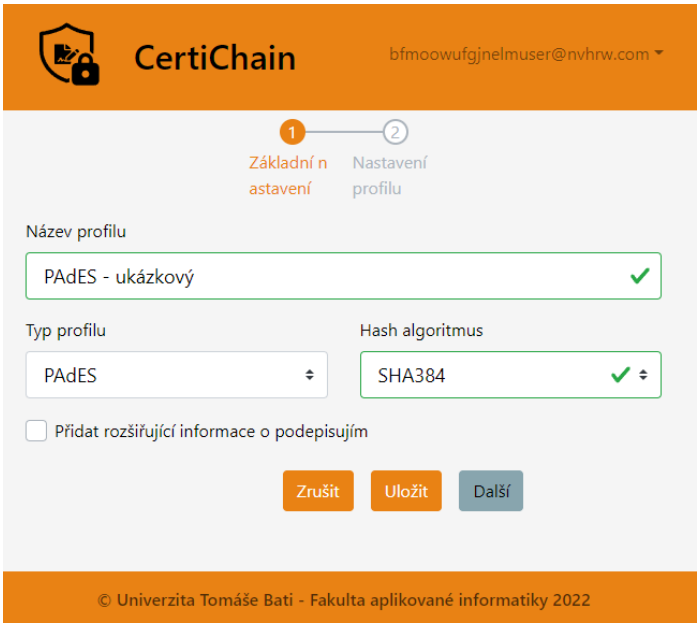
Google. Google ale v průběhu května má přestat podporovat odesílání emailů pro účty Google z aplikací třetích stran, proto se může stát, že email pro potvrzení emailové adresy nebude doručen. V používaní aplikace to ale nijak nebrání a jedná se jen o malý detail.

Po registraci je uživatel přesměrován na rozcestník funkcionalit aplikace. Z rozcestníku se poté může vydat na: vytvoření digitálního podpisu, ověření digitálního podpisu, správa podepisovacích profilu, přehled záznamů v systému blockchain a správa účtu.



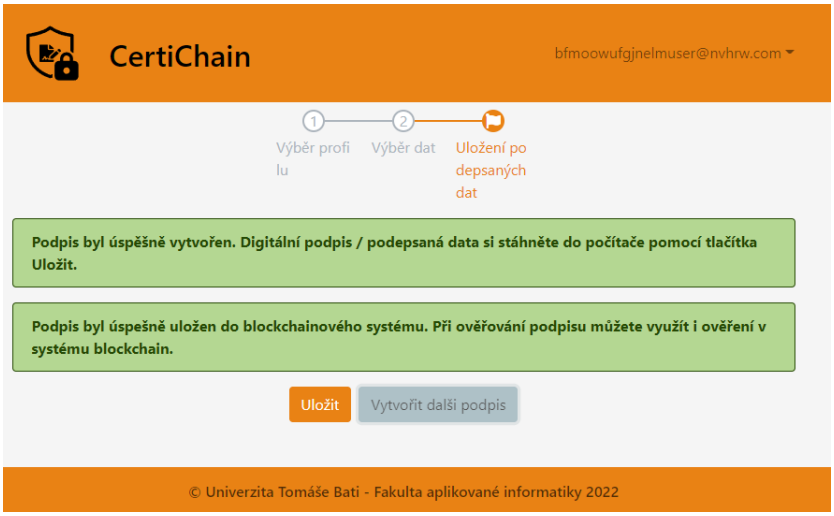
Obr. 87 Rozcestník funkcionalit aplikace (vlastní)

Před vytvořením digitálního podpisu je nejprve potřeba vytvořit si podepisovací profil. Tato operace lze zahájit pomocí tlačítka *Přejít na správu profilů*, ve které je dostupné tlačítko *Vytvořit profil*. Při vytváření profilu je potřeba vyplnit všechny potřebné parametry a poté je uložit pomocí tlačítka *Uložit.* Po uložení je uživatel přesměrován zpět na obrazovku správy profilů a měl by vidět svůj nově přidaný profil.



Obr. 88 Vytvoření/Editace podepisovacího profilu (vlastní)

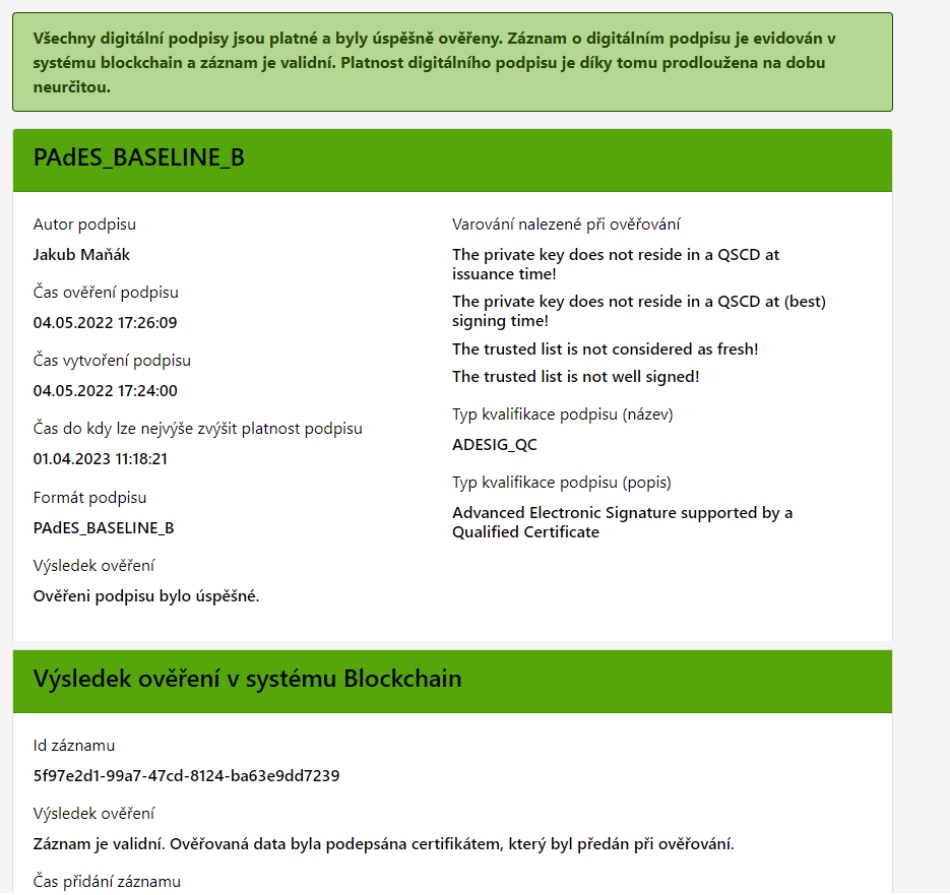
Po vytvoření podepisovacího profilu lze začít vytvářet samotné digitální podpisy. Na obrazovku vytvoření digitálního podpisu lze přejít z rozcestníku funkcionalit pomocí tlačítka *Přejít na vytvoření podpisu*. Pří vytváření podpisu si uživatel nejprve vybere podepisovací profil, v dalším kroku vybere data k podepsání, podepisovací certifikát, zadá heslo k certifikátu, a vybere, zda chce podpis přidat do blockchain. Poté lze odeslat požadavek na vytvoření podpisu pomocí tlačítka *Podepsat* a vyčkat na dokončení procesu tvorby podpisu. Po přijetí podepsaných dat je může uživatel stáhnout pomocí tlačítka *Uložit.*



Obr. 89 Dokončení procesu vytvoření digitálního podpisu (vlastní)

Další funkcionalitou je možnost ověření podpisu, na kterou lze přejít z rozcestníku pomocí tlačítka *Přejít na ověřování podpisů*. V prví části vybere podepsaná data k ověření a pokud

chce zjistit přítomnost záznamu v blockchain, musí vybrat i podepisovací certifikát. Poté zašle požadavek na ověření na BE pomocí tlačítka *Ověřit podpis*. Po dokončení procesu ověření je uživateli zobrazen výsledek ověření.

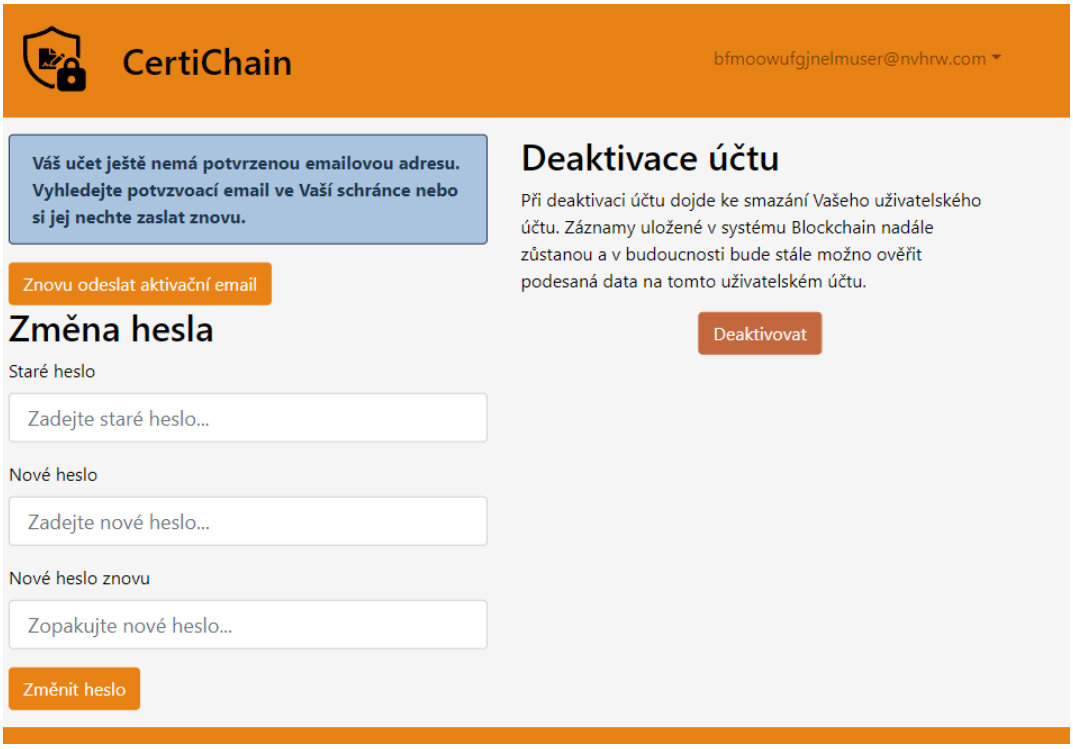


Obr. 90 Výsledek ověření digitálního podpisu (vlastní)

Přehled záznamů uživatele v blockchain si lze zobrazit na obrazovce Přehled záznamů v blockchain, na kterou se lze dostat z rozcestníku pomocí tlačítka *Přejít na přehled záznamů v blockchain.* Na obrazovce jsou dostupné záznamy v blockchain jejichž vytvoření inicioval uživatel při vytváření digitálních podpisů. V přehledu jsou umístěny pouze informace, které by neměly nijak ohrozit samotný záznam v blockchain a uživatel u každého záznamu může získat jeho aktuální stav pomocí tlačítka *Zkontrolovat záznam*. V případě velkého množství záznamů může jednotlivé záznamy filtrovat pomocí zadání data od a do.

Obr. 91 Přehled záznamů v blockchain (vlastní)

Poslední funkcionalitou v rozcestníku je možnost správy účtu, na kterou se z rozcestníku dá přejít pomocí tlačítka *Přejít na správu účtu*. Na obrazovce je možno si změnit heslo, znovu si odeslat aktivační email na potvrzení emailové adresy nebo v situaci, kdy uživatel již nadále nechce používat aplikaci CertiChain může si zde účet z deaktivovat.



Obr. 92 Obrazovka pro správu účtu (vlastní)

ZÁVĚR

DP Aplikace CertiChain – využití technologie blockchain pro podepisování dat se skládala ze šesti hlavních bodů. První dva body práce byly teoretického zaměření a jejich cílem bylo seznámit se s problematikou digitálních podpisů a technologií blockchain. Třetím bodem DP bylo vybrat vhodné technologie pro implementaci aplikace, ve čtvrtém bodě pomocí vybraných technologií naimplementovat vlastní řešení aplikace, v pátém bodě aplikaci vhodně otestovat a v posledním šestém bodě seznámit čtenáře s naimplementovaným řešením. Pro splnění všech bodů DP byla práce celkově rozdělena na osm kapitol.

První kapitola DP se soustředila na první bod zadání DP, tedy seznámit se s problematikou digitálních podpisů a certifikátů. Nejprve byl popsán princip samotného algoritmu pro vytváření digitálního podpisu, který byl uveden již v roce 1991. Poté práce na problematiku digitálního podepisování nahlížela ze dvou stran. Nejprve byla rozebrána ze strany zákonů, kdy v ČR je dostupný zákon pro digitální podepisování již v od roku 2000. Tento zákon poté nahradilo v roce 2016 evropské nařízení EiDAS. Zákon i evropské nařízení jsou v první kapitole rozebrány a čtenář je zde seznámen se základními pojmy jako elektronický podpis, kvalifikovaný elektronický podpis, kvalifikovaný certifikát apod. Poté bylo na problematiku elektronického podepisování nahlédnuto ze strany technické, kdy práce stručně rozebrala technické normy pro vytváření podpisů CAdES, XAdES a PAdES.

Druhá kapitola DP cílila na problematiku blockchain technologií, tedy druhý bod zadání DP. V druhé kapitole je rozebrán základní princip blockchain a jakém „základním kameni“ je tahle technologie postavena. Dále byly v kapitole rozebrány základní rozdíly mezi tradičními databázemi (např. SQL databáze) a blockchain. Poté byly rozebrány základní struktury v blockchain systémech, tedy transakce a bloky, pojem MerkleTree. Na závěr kapitoly bylo rozepsáno o kategoriích využití systémů blockchain a jaké využití bude mít blockchain v samotné diplomové práci.

Třetí kapitola DP se soustředila na výběr možných technologií pro implementaci aplikace CertiChain, kdy aplikace se složena ze dvou částí – Front End a Back End. V kapitole byly rozebrány použitelné technologie pro obě části aplikace kdy pro BE část byly navrhnuty technologie jako C# .Net, Java, Python a PHP. Implementace FE byla rozdělena na více možností implementace, kdy FE část primárně cílí na desktopové zařízení. Proto by FE část mohla být implementována jako klasická aplikace nebo webová aplikace. Z desktopových technologií byly zmíněny pro macOS SwitfUI, pro Windows C# .Net (WinForms, WPF,

UWP a WinUI), bylo zmíněno řešení multiplatformní aplikace pomocí C# .NET (UNO Plafrom, Mavalonia) nebo pomocí C++ (Qt, GTK) a dále řešení pomocí webové aplikace, kdy možnosti implementace byly zmíněny opět C# .Net, Javascriptové a TypeScriptové frameworky a PHP Laravel.

Čtvrtá kapitola se zabývala návrhem aplikace CertiChain, kdy nejprve byly vytvořeny a sepsány funkcionální a nefunkcionální požadavky, které byly následně dopodrobna vysvětleny. Po sepsání požadavků byly z popisu požadavků sepsány samotné datové entity, které se budou v aplikaci vyskytovat. Z návrhu entit byl na závěr vytvořen i samotný relační model datových entit, který zobrazil vtahy mezi jednotlivými datovými entitami. V poslední části analýzy byly z jednotlivých funkcionálních požadavků navrženy i jednotlivé obrazovky aplikace pomocí nástroje Figma.

Pátá kapitola se zabývá výběrem technologií pro implementaci FE a BE části aplikace CertiChain. Ke každé vybrané technologii bylo zmíněno, proč byla vybrána právě tahle technologie. Dále je v kapitole zmíněna technologie Docker, která je použita pro spuštění databázových systémů, které aplikace používá. Na závěr páté kapitoly jsou rozebrány ještě použité databáze – databáze MySQL pro ukládání aplikačních datových entit a databáze ProvenDB, která představuje samotný systém blockchain. Cílem čtvrté a páté kapitoly bylo tedy splnit třetí bod DP – vybrat vhodné technologie pro vlastní implementaci aplikace.

Šestá kapitola popisuje postup při implementaci vlastního řešení aplikace neboli čtvrtý bod zadání DP. Šestá kapitola je tedy rozdělena na podkapitoly podle implementace jednotlivých funkcionalit aplikace. Nejprve je rozebrán samotný postup, jak byly oba projekty aplikace založeny, byla popsána jejich základní struktura a konfigurace, která byla potřeba provést před samotnou implementací. Poté už se kapitola věnovala implementaci hlavních funkcionalit – byla rozepsána implementace správy uživatelů, správy podepisovacích profilů, vytvoření digitálního podpisu, ověření digitálního podpisu a na závěr zakomponování blockchain do aplikace. Postup implementace každé části byl popsán ze dvou stran – implementace v projektu pro FE část a pro BE část aplikace.

Sedmá kapitola byla věnovávána testování aplikace, tedy pátému bodu zadání práce. Při testování aplikace bylo využito jak manuálního testování, tak i vytvoření sady automatických testovacích případů pomocí Robot Framework. Automatické testování pokrylo skoro všechny části aplikace (přihlášení, registrace, tvorba profilů, tvorba podpisu, ověření

podpisu a přehled záznamů v systému blockchain), některé situace (editace a smazání podepisovacích profilů a funkce správy účtu) byly testovány pouze manuálně.

Poslední osmá kapitola se zabývala výslednou prezentací naimplementované aplikace. Nejprve byl popsán postup nasazení aplikace, kdy aplikace byla nasazena na vlastní zařízení a pomocí konfigurace domácí sítě bylo zařízeno, aby aplikace byla dostupná na veřejné síti, byl popsán popis získání doménového jména a záznamu v DNS a postup zprovoznění https aktivace nasazením SSL certifikátu. V druhé části osmé kapitoly byla prezentována výsledná aplikace a její základní použití. Při prezentaci bylo ukázáno, jak se do systému zaregistrovat, jak si vytvořit podepisovací profily, jak vytvořit a ověřit digitální podpis včetně přidání a ověření záznamu v blockchain.

Závěrem lze říct, že aplikace byla naimplementována podle navrhnutých kritérií a obsahuje všechny funkcionality, které byly požadovány. Při práci na DP bylo potřeba překonat pár problémů, zejména použití Java Spring pro tvorbu BE z důvodu takřka nulových zkušeností s touto technologií, a obtíže s výběrem technologie blockchain. Všechny problémy se ale nakonec podařilo překonat.

Aplikace by určitě mohla být v budoucnosti dále rozšiřována – nabízí se např. vylepšení vzhledu aplikace, kdy nyní aplikace jistě nenabízí úplně nejlepší uživatelský zážitek. Další vhodné vylepšení by mohlo být přesunutí tvorby digitálního podpisu na straně klienta, např. pomocí doplňku, který by se přidal do prohlížeče. Poté by uživatel mohl využít pro vytvoření podpisu i certifikát s privátním klíčem umístěným na kvalifikovaném zařízení (např. čipová karta) a mohl by tedy pomocí aplikace vytvářet i kvalifikované digitální podpisy.

SEZNAM POUŽITÉ LITERATURY

[1] Zákon č. 227/2000 Sb., o elektronickém podpisu. Ministerstvo vnitra České republiky [online]. 2012, 1. 10. 2012 [cit. 2021-11-19]. Dostupné z: https://www.mvcr.cz/clanek/zakon-c-227-2000-sb-o-elektronickem-podpisu.aspx
[2] ČESKÁ REPUBLIKA. Nařízení Evropského parlamentu a Rady (EU) č. 910/2014 ze dne 23. července 2014 o elektronické identifikaci a službách vytvářejících důvěru pro elektronické transakce na vnitřním trhu a o zrušení směrnice 1999/93/ES. In: Https://eur-lex.europa.eu. EU, 2014, ročník 2014, částka 914, číslo 914. Dostupné také z: https://eur-lex.europa.eu/legal-content/CS/ALL/?uri=celex:32014R0910
[3] ČESKÁ REPUBLIKA. Zákon o službách vytvářejících důvěru pro elektronické transakce. In: Sbírka zákonů. Praha: Tiskárna Ministerstva vnitra, 2016, ročník 16, částka 115, číslo 297. Dostupné také z: https://www.zakonyprolidi.cz/cs/2016-297
[4] Normy ETSI: CAdES. Earchivace.cz [online]. [cit. 2021-11-26]. Dostupné z: http://www.earchivace.cz/legislativa-a-normy/aplikace-norem-pro-elektronickou-archivaci/
[5] Normy ETSI: XAdES. Earchivace.cz [online]. [cit. 2021-11-26]. Dostupné z: http://www.earchivace.cz/legislativa-a-normy/aplikace-norem-pro-elektronickou-archivaci/
[6] Normy ETSI: PAdES. Earchivace.cz [online]. [cit. 2021-11-26]. Dostupné z: http://www.earchivace.cz/legislativa-a-normy/aplikace-norem-pro-elektronickou-archivaci/
[7] Přehled kvalifikovaných poskytovatelů certifikačních služeb a jejich kvalifikovaných služeb. Mvcr.cz [online]. ČR, 30.5.2016 [cit. 2022-03-22]. Dostupné z: https://www.mvcr.cz/clanek/prehled-kvalifikovanych-poskytovatelu-certifikacnich-sluzeb-a-jejich-kvalifikovanych-sluzeb.aspx
[8] ETSI TS 101 733 V2.2.1: Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES). 221. Francie, 2013.
[9] ETSI TS 101 903 V1.4.1: XML Advanced Electronic Signatures (XAdES). Francie, 2009.
[10] KALOUSEK, Zbyněk. Blockchain – o co se jedná a jak funguje (3. díl): Rozdíly mezi základními druhy databází. In: Www.kurzy.cz [online]. ČR: Zbyněk Kalousek, 2021, 23.09.2021 21:16:41 [cit. 2022-04-20]. Dostupné z: https://www.kurzy.cz/zpravy/611089-blockchain--o-co-se-jedna-a-jak-funguje-3-dil/
[11] Blockchain: Co je blockchain a jak blockchain u kryptoměn funguje?: Transakce a bloky – Základní principy fungování. In: Finex [online]. ČR: Portál Finex.cz, 2022, 21. 3. 2022 [cit. 2022-04-20]. Dostupné z: https://finex.cz/blockchain/
[12] Blockchain. In: Http://voho.eu: Blockchain [online]. ČR: Vojtěch Hordějčuk, c2008-2022 [cit. 2022-04-21]. Dostupné z: http://voho.eu/wiki/blockchain/
[13] Merkle Tree in Blockchain: What is it, How does it work and Benefits. In: SimpliLearn [online]. USA: SimpliLearn, 2021, Nov 8, 2021 [cit. 2022-04-21]. Dostupné z: https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain
[14] Building REST services with Spring. In: Spring [online]. Spring, © 2022 [cit. 2022-04-22]. Dostupné z: Building REST services with Spring
[15] CAMPBELL, Steve. Flask vs Django: What's the Difference Between Flask & Django?. In: Guru99 [online]. guru99, 2022, March 5, 2022 [cit. 2022-04-22]. Dostupné z: https://www.guru99.com/flask-vs-django.html
[16] KREBS, Bruno. Developing RESTful APIs with Python and Flask. In: Auth0 [online]. Seatle: Auth0, 2021, August 30, 2021 [cit. 2022-04-22]. Dostupné z: https://auth0.com/blog/developing-restful-apis-with-python-and-flask/
[17] KREBS, Bruno. SQLAlchemy ORM Tutorial for Python Developers. In: Auth0 [online]. Auth0, 2017, November 09, 2017 [cit. 2022-04-23]. Dostupné z: https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/

[18] Controllers. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/controllers
[19] Eloquent: Serialization. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/eloquent-serialization#serializing-to-json
[20] Eloquent: Getting Started. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/eloquent#generating-model-classes
[21] Database: Getting Started. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/database
[22] Laravel Sanctum. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/sanctum
[23] Laravel Passport. In: Laravel [online]. Laravel, © 2011-2022 [cit. 2022-04-23]. Dostupné z: https://laravel.com/docs/9.x/passport
[24] What's a Universal Windows Platform (UWP) app?. Microsoft: Windows App Development [online]. Microsoft, 2022, 02/16/2022 [cit. 2022-04-24]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
[25] ČÁPKA, David. Https://www.itnetwork.cz/java/javafx/java-tutorial-uvod-do-javafx. In: Itnetwork [online]. čr: itnetwork, © 2022 [cit. 2022-04-24]. Dostupné z: https://www.itnetwork.cz/java/javafx/java-tutorial-uvod-do-javafx
[26] How Uno Platform works. In: Uno Platform [online]. Uno Platform 215 rue St-Jacques, Suite 500 Montréal QC, H2Y 1M6: Uno Platform [cit. 2022-04-24]. Dostupné z: https://platform.uno/docs/articles/how-uno-works.html
[27] Avalonia: Documentation. Avalonia [online]. Tallinn, Kesklinna linnaosa, Ahtri tn 12, 10151, Estonia: Avalonia [cit. 2022-04-24]. Dostupné z: https://docs.avaloniaui.net
[28] Qt Documentation: Qt Widgets [online]. Qt, © 2022 [cit. 2022-04-25]. Dostupné z: https://doc.qt.io/qt-5/qtwidgets-index.html
[29] Qt Documentation: Qt QML [online]. Qt, © 2022 [cit. 2022-04-25]. Dostupné z: https://doc.qt.io/qt-5/qtqml-index.html
[30] What is Angular?. In: Angular [online]. Angular, ©2010-2022. [cit. 2022-04-26]. Dostupné z: https://angular.io/guide/what-is-angular
[31] Důvěryhodné e-služby v EU mají své logo a eIDAS klepe na dveře. In: Lupa.cz [online]. ČR: Lupa, 2015 [cit. 2022-05-13]. Dostupné z: https://i.iinfo.cz/images/232/znacka-duvery-eu-logo-1.png

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

BE – BackEnd

CRL – Cert Revocation List

č. – číslo

DP – Diplomová práce

DSS – Digital Signature Service

EU – Evropská unie

FE – FrontEnd

JWT – JSON Web Token

MVVM – Model-View-ViewModel

OCSP – Online Certificate Status Protocol

ORM – Object Relational Mapping (Objektově Relační Mapování)

Sb. – Sborník

UWP – Universal Windows Platform

WPF – Windows Presentation Foundsation

XAML – Extensible Application Markup Language

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář Text DP – obsahuje text DP ve formátu PDF
- Adresář Zdrojové kódy – obsahuje zdrojové kódy obou částí aplikace. Zdrojový kód klientské části aplikace se nachází v adresáři CertiChain.Frontend. Zdrojový kód pro serverovou část aplikace se nachází v adresáři Backend. Dále adresář obsahuje ještě projekt s automatickými testy, který se nachází v adresáři Tests.
- Adresář Výstupy testů – obsahuje výstupy automatických testů. Pro každou testovací sadu je dostupný adresář, který obsahuje reporty o provedení dané sady.