

Aplikace pro správu časových razítek

Jakub Maňák



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Jakub Maňák**
Osobní číslo: **A17133**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Softwarové inženýrství**
Forma studia: **Prezenční**
Téma práce: **Aplikace pro správu časových razítek**
Téma práce anglicky: **An Application for Time-stamp Management**

Zásady pro vypracování

1. Popište problematiku časových razítek a aplikací pro jejich správu.
2. Navrhněte aplikaci pro správu časových razítek.
3. Zvolte vhodné metody a prostředky pro implementaci aplikace.
4. Implementujte aplikaci.
5. Vhodně vyhodnoťte a prezentujte výsledky.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam doporučené literatury:

1. DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. Velký průvodce infrastrukturou PKI a technologií elektronického podpisu. 2., aktualiz. vyd. Brno: Computer Press, 2009. ISBN 978-80-251-2619-6.
2. ETSI TS 101 861: Electronic Signatures and Infrastructures (ESI); Time stamping profile. V1.4.1. Francie: Cedex, 2011.
3. MENEZES, A. J., Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. ISBN 978-0849385230.
4. 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.
5. <https://tools.ietf.org/html/rfc3191>

Vedoucí bakalářské práce:

Ing. Petr Žáček

Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce:
Termín odevzdání bakalářské práce:

28. listopadu 2019
15. května 2020



doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Jméno, příjmení: Jakub Maňák

Název bakalářské práce: Aplikace pro správu časových razítek

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou 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/bakalářskou 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 bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 27.7.2020

Jakub Maňák, v. r. ...
podpis diplomanta

ABSTRAKT

Bakalářská práce se zabývá problematikou časových razítek a aplikacemi pro jejich správu. V teoretické části je objasněn pojem časové razítko, jak vzniká a kde se časová razítka využívají a jaké všechny funkce by měla obsahovat aplikace pro správu časových razítek. Praktická část obsahuje výběr prostředků a metod, pomocí kterých je aplikace implementována (C#, WPF). Dále je zde rozepsán způsob implementace aplikace a rozbor důležitých částí kódu. Výsledným produktem je funkční aplikace, pomocí které lze vytvářet a ověřovat časová razítka.

Klíčová slova: Časové razítko, C#, WPF

ABSTRACT

The bachelor thesis deals with the issue of time stamps and applications for their administration. The theoretical part explains the term time stamp, how it originates, where the timestamps are used, and which functions should timestamp management applications include. The practical part contains a selection of means and methods by which the application is implemented (C#, WPF). Furthermore, it describes the process of creating an application and discusses important parts of the code. The resulting product is a functional application that can create and validate time stamps.

Keywords: Time stamp, C#, WPF

V této části bych chtěl poděkovat vedoucímu mé bakalářské práce, konkrétně panu Ing. Petru Žáčkovi za jeho rady a pomoc při tvorbě této práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD.....	10
I TEORETICKÁ ČÁST	12
1 HASH (OTISKOVÉ) ALGORITMY	13
1.1 TYPY HASH ALGORITMŮ	13
2 ČASOVÉ RAZÍTKO	14
2.1 ŽÁDOST O ČASOVÉ RAZÍTKO.....	14
2.1.1 Povinné položky žádosti.....	14
2.1.2 Nepovinné položky žádosti	15
2.2 ODPOVĚĎ ČASOVÉHO RAZÍTKA	16
2.2.1 Status	17
2.3 TIMESTAMPTOKEN	18
2.3.1 Version	19
2.3.2 Policy.....	19
2.3.3 MessageImprint.....	20
2.3.4 SerialNumber	20
2.3.5 GenTime.....	20
2.3.6 Accuracy.....	20
2.3.7 Ordering	20
2.3.8 Nonce	21
2.3.9 Tsa	21
2.3.10 Extensions	21
2.4 PLATNOST ČASOVÉHO RAZÍTKA.....	21
2.5 PROCES TVORBY ČASOVÉHO RAZÍTKA	21
2.6 PROCES OVĚŘENÍ ČASOVÉHO RAZÍTKA	22
3 AUTORITY PRO VYDAVÁNÍ ČASOVÝCH RAZÍTEK.....	23
3.1 TYPICKÁ ARCHITEKTURA TSA	23
3.2 PRAVIDLA POŽADOVANÉ NORMOU RFC 3161 PO TSA.....	23
4 APLIKACE PRO SPRÁVU ČASOVÝCH RAZÍTEK.....	26
4.1 ZÁKLADNÍ FUNKCE.....	26
4.2 DOPLŇUJÍCÍ FUNKCE.....	27
II PRAKTICKÁ ČÁST	28
5 MOŽNOSTI IMPLEMENTACE APLIKACE	29
5.1 APLIKACE PRO DESKTOPOVÉ SYSTÉMY.....	29
5.1.1 CLI aplikace	29
5.1.2 Aplikace s grafickým rozhraním	29
5.2 MOBILNÍ APLIKACE	30
5.2.1 Nativní vývoj – Android	30

5.2.2	Nativní vývoj – iOS.....	31
5.2.3	Hybridní vývoj mobilních aplikací	31
6	ZVOLENÁ METODA IMPLEMENTACE.....	32
6.1	.NET FRAMEWORK	32
6.1.1	CLR	32
6.1.2	Knihovna tříd.....	32
6.1.3	NuGet balíčky	33
6.2	WPF.....	34
6.2.1	XAML	34
6.2.2	Kód na pozadí.....	35
6.3	MVVM A DATA-BINDING.....	36
6.3.1	Vrstvy MVVM	37
6.3.2	Data-Binding	37
6.3.3	Způsob propojení.....	39
6.3.4	Příznak aktualizace dat.....	39
7	IMPLEMENTACE APLIKACE	41
7.1	STRUKTURA APLIKACE	41
7.1.1	Properties.....	41
7.1.2	Communication	42
7.1.3	Converters	42
7.1.4	Enums.....	43
7.1.5	Files a Images.....	43
7.1.6	Models	43
7.1.7	Utils	44
7.1.8	ViewModels	44
7.1.9	Views.....	45
7.2	ZPRACOVÁNÍ NORMY RFC-3161 – TVORBA ČASOVÉHO RAZÍTKA	45
7.3	PŘÍSTUP KE SDÍLENÝM DATŮM V PROGRAMU	47
7.4	HLAVNÍ OKNO APLIKACE.....	52
7.5	VYTVOŘENÍ ČASOVÉHO RAZÍTKA	55
7.6	OVĚŘENÍ A OTEVŘENÍ EXISTUJÍCÍHO ČASOVÉHO RAZÍTKA	58
7.6.1	Ověření časového razítka	59
7.6.2	Otevření časového razítka	63
7.7	MANIPULACE S HISTORIÍ ČASOVÝCH RAZÍTEK.....	63
7.8	DETAIL ČASOVÉHO RAZÍTKA	64
7.8.1	Informace o časovém razítku	64
7.8.2	Dodatečné akce dostupné v detailu časového razítka	67
7.9	NASTAVENÍ APLIKACE.....	68
7.9.1	Nastavení připojení k internetu	68
7.9.2	Nastavení služby TSA	69
8	PREZENTACE VÝSLEDNÉ APLIKACE.....	71

8.1	PRVNÍ SPUŠTĚNÍ APLIKACE	71
8.2	VYTVOŘENÍ ČASOVÉHO RAZÍTKA	72
8.3	OVĚŘENÍ / OTEVŘENÍ ČASOVÉHO RAZÍTKA.....	73
8.4	ZOBRAZENÍ DETAILU ČASOVÉHO RAZÍTKA Z HISTORIE	74
ZÁVĚR		76
SEZNAM POUŽITÉ LITERATURY		78
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		79
SEZNAM OBRÁZKŮ		80
SEZNAM PŘÍLOH.....		82

ÚVOD

Časové razítko je pojem, o kterém většina lidí, kteří používají počítač dennodenně, ani netuší, že existuje. Přesto je to velmi užitečná věc, protože časové razítko je datový soubor, který potvrzuje existenci a konkrétní podobu dokumentu v čase, kdy bylo časové razítko vytvořeno. Časová razítka se často vytvářejí k dokumentům, jako jsou např. účetní a daňové doklady, celní doklady, dokumentace k výrobkům atd. Když si osoba vytvoří k dokumentu časové razítko, bude mu sloužit jako důkaz, že tento dokument byl vytvořen v dané podobě v čase vytvoření časového razítka. Díky časovému razítku lze jednoduše ověřit, zda bylo s dokumentem od doby vytvoření časového razítka manipulováno.

Cílem bakalářské práce je čtenáře seznámit s problematikou časových razítek a dále navrhnout a naprogramovat aplikaci, která umožní spravovat časová razítka. Aby byl cíl úspěšně dosažen, je práce rozdělena do 8 hlavních kapitol.

V první kapitole jsou rozebrány Hash algoritmy. Druhá kapitola se zabývá problematikou časových razítek. Co časové razítko obsahuje a jak lze ověřit, že soubor, ke kterému bylo časové razítko vytvořeno, od vzniku časového razítka nebyl změněn. Dále je ve druhé kapitole rozepsán proces tvorby a ověření časového razítka. Ve třetí kapitole je vysvětlen pojem poskytovatel časového razítka a jaké podmínky musí poskytovatelé časových razítek splňovat, aby časová razítka mohli vydávat. Teoretická rešerše čerpá zejména z normy, která definuje časová razítka.

Cílem čtvrté kapitoly je seznámit čtenáře s aplikacemi, které umožňují spravovat časová razítka a jejich dostupností. Součástí kapitoly je také návrh aplikace pro správu časových razítek. Jsou zde sepsány základní požadavky funkcí (vlastností), které by aplikace měla splňovat.

Důležité je navrhnout správné prostředky a metody, pomocí kterých by aplikace mohla být vytvořena. Tímto tématem se zabývá pátá kapitola. Z uvedených prostředků a metod byl vybrán programovací jazyk a framework, pomocí kterých je aplikace implementována. Seznámení s vybraným programovacím jazykem a frameworkem se zabývá šestá kapitola.

Sedmá kapitola práce se zabývá implementací aplikace a je zde postupně popsáno, jak aplikace vznikala. Jsou zde rozebrány nejdůležitější funkce aplikace, tedy vytváření, ověřování, zobrazení detailu a ukládání historie časových razítek.

Úkolem poslední kapitoly je prezentovat výslednou funkčnost aplikace, tedy zkontrolovat, zda aplikace splňuje všechny základní požadavky, zhodnotit funkčnost aplikace, a na závěr navrhnout možné vylepšení, které by aplikace dále mohla obsahovat.

I. TEORETICKÁ ČÁST

1 HASH (OTISKOVÉ) ALGORITMY

Ještě předtím, než se práce začne zabývat časovými razítky je vhodné stručně zmínit i hashovací algoritmy, které jsou nedílnou součástí časových razítek. Hashování je jednocestná funkce, která z libovolně dlouhých dat (např. textový řetězec nebo libovolný soubor či dokument) vytvoří krátký řetězec o konstantní velikosti.

Jednocestná funkce je speciální druh algoritmů, kdy jejich výpočet je poměrně jednoduchá záležitost, ale jejich zpětná konverze do původních dat je nemožná. Obecně se říká, že nemá cenu v otisku hledat zprávu či data, ze kterých byl otisk vytvořen. Jeden velice praktický příklad jednocestného algoritmu uvedli i autoři knihy *Velký průvodce infrastrukturou PKI*: „Je přece jednoduché se oženit či vdát, ale často velice obtížné se rozvést.“ [1]

Další vlastnost, kterou by kvalitní hashovací algoritmy měly obsahovat je, že pokud se původní řetězec nebo data, byť jen minimálně změní, výsledný otisk by měl být výrazně jiný. Protože lze otisk spočítat z libovolně dlouhých dat či textového řetězce, tak ke konkrétnímu otisku je teoreticky možné najít nekonečné množství původních dat (textových řetězců).

Jednocestné algoritmy jsou založeny na nízko úroňových výpočetních úrovních, nejčastěji bitové operace a posuny. [1]

1.1 Typy Hash algoritmů

Zde je seznam a délka výstupů nejznámějších Hashovacích algoritmů, které se používaly, a ještě dnes používají:

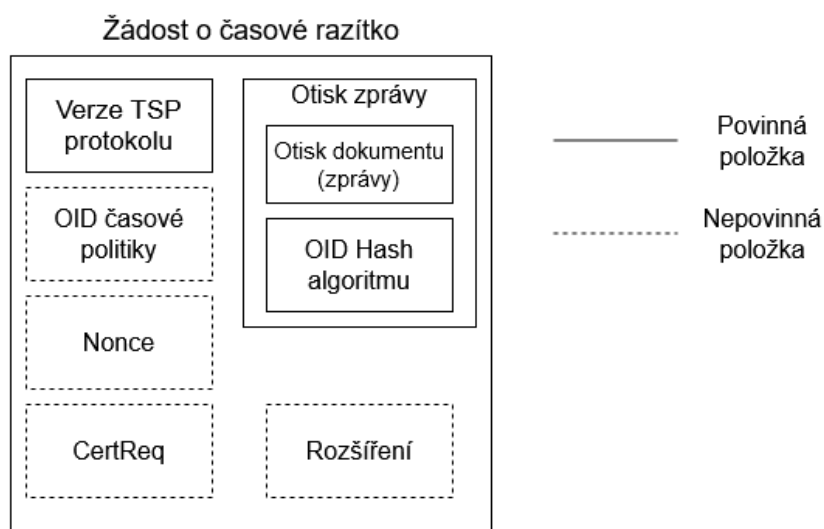
- SHA-1 s výstupní délkou 160 bitů;
- SHA-2 s výstupní délkou 224, 256, 384, 512 bitů;
- SHA-3 s výstupní délkou 224, 256, 384, 512 bitů;
- MD5 s výstupní délkou 128 bitů.

Algoritmy SHA-1 a MD5 se dnes již považují za zastaralé a nahrazují se novějšími (SHA-256 a výš), protože u MD-5 a SHA-1 se dnes již daří nacházet data (textové řetězce se stejným otiskem - kolize). [1, 3]

2 ČASOVÉ RAZÍTKO

Časové razítko je datová struktura, která svazuje „orazítovaný“ dokument s časem. Časové razítko tedy udává, že nějaký dokument existoval v dané podobě, kdy bylo časové razítko vytvořeno. Pomocí časového razítka lze také ověřit, zda od jeho vytvoření bylo s dokumentem manipulováno, přesněji jestli jej někdo modifikoval. Časové razítko má pevně danou strukturu, je specifikováno protokolem pro vydávání časových razítek (TSP) neboli RFC-3161. Protokol TSP se skládá z žádosti o časové razítko a odpovědi na tuto žádost. [2]

2.1 Žádost o časové razítko



Obr. 1 Schéma žádosti o časové razítko

Na obr. 1 lze vidět, že žádost je přesně daná sekvence nebo objekt, který má povinné položky (verze protokolu TSP a otisk zprávy) a může obsahovat ještě dodatečné položky (OID časové politiky, nonce, CertReq a rozšíření). Po vytvoření se žádost o časové razítko odesílá na server časové autority. V praxi se také žádost často ukládá do souboru. Při ukládání žádosti do souboru se obvykle používá přípona .tsq. [1]

2.1.1 Povinné položky žádosti

Validní žádost o časové razítko musí obsahovat několik povinných položek, které budou rozepsány v níže.

2.1.1.1 Verze TSP protokolu

Verze protokolu TSP je typu integer a obsahuje verzi použitého TSP protokolu. Dnes existuje pouze verze č. 1 TSP protokolu, ale kvůli případným změnám v budoucnu musí být verze protokolu TSP uvedena. [2]

2.1.1.2 Otisk zprávy

Otisk zprávy je další sekvence (objekt), která musí být v žádosti obsažena a je to pro žadatele nejdůležitější věc, protože obsahuje otisk (hashedMessage) a identifikátor algoritmu (OID) (hashAlgorithm), který otisk vytvořil. OID algoritmu slouží k tomu, aby časová autorita mohla zkontrolovat, zda použitý algoritmus podporuje a zda otisk dokumentu (zprávy) má správnou délku (viz kapitola 1). Položka hashedMessage je textový řetězec převedený na pole Bytů (octet string). Položka hashAlgorithm je typu AlgorithmIdentifier. [2]

2.1.1.3 CertReq

CertReq je položka typu boolean. CertReq musí být v žádosti obsažena, ale uživatel jí není povinen ručně zadávat, protože výchozí hodnota CertReq je FALSE. Je to příznak, který časové autoritě udává, zda má časová autorita spolu s časovým razítkem zaslat také certifikát časové autority. Pokud je příznak TRUE, časová autorita musí certifikát zaslat s časovým razítkem. Pokud je příznak FALSE, certifikát není zaslán s časovým razítkem. [2]

2.1.2 Nepovinné položky žádosti

Žádost o časové razítko může být dále rozšířena o volitelné položky, které nemusí být součástí výsledné žádosti o časové razítko. Jednotlivé nepovinné položky jsou uvedeny níže.

2.1.2.1 OID časové politiky

Žadatel, který vytváří žádost o časové razítko si může specifikovat, jaká časová politika se má použít pro vytvoření časového razítka. Může nastat situace, kdy žadatel vloží politiku, kterou časová autorita nepodporuje. Pokud tato situace nastane, žádost o časové razítko bude zamítnuta, a tudíž nedojde k vytvoření časového razítka. [2]

2.1.2.2 Nonce

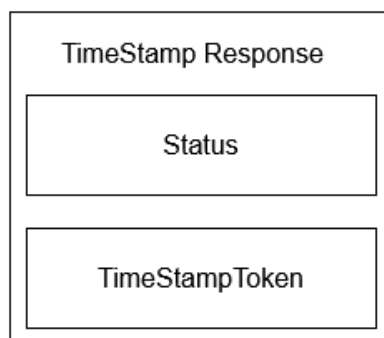
Nonce (šum) je objekt typu BigInteger. Jedná se o velmi velké, náhodně vygenerované číslo. Velikost je typicky 64 bitů. Nonce slouží k jednoduššímu spárování žádosti o časové razítko

s výsledným časovým razítkem, protože pokud je Nonce uveden v žádosti, musí být uveden i v časovém razítku. [2]

2.1.2.3 Extensions

Extensions (rozšíření) je pole prvků, které se k žádosti o časové razítko můžou přidat. Rozšíření se může do žádosti přidat i více. Existují dva typy rozšíření, kritické a nekritické. Při vydání časového razítka typ rozšíření, dá se říci, nehraje žádnou roli. Pokud žádost o časové razítko obsahuje nějaké rozšíření, časová autorita je musí projít a zjistit, zda je pro ni známá nebo neznámá. Pokud časová autorita najde nějaké rozšíření, které nezná (nezáleží, zda je kritická nebo nekritická), časová autorita musí tuhle žádost zamítnout a nevydá časové razítko. Typ rozšíření hraje roli např. při vydávání certifikátů, kdy certifikační autorita nesmí vydat certifikát, jen pokud je rozšíření kritické a certifikační autorita jej nerozpozná. [1]

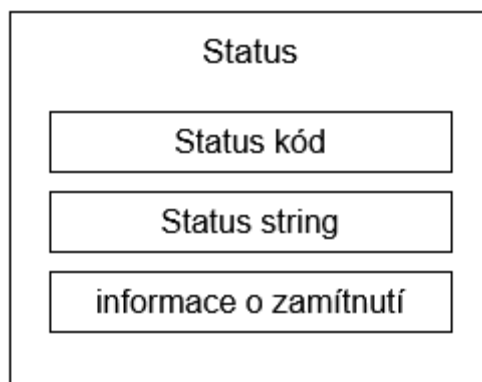
2.2 Odpověď časového razítka



Obr. 2 Schéma odpovědi z časové autority

Odpověď ze serveru obsahuje Status, který musí být v odpovědi vždy uveden a dále *TimeStampToken*, což je objekt, který v sobě ukrývá detaily časového razítka. *TimeStampToken* může být prázdný, pokud časová autorita zamítla žádost o časové razítko. Časová autorita vždy vrátí žadateli odpověď, ve které mu buď zašle časové razítko, které je obsaženo v *TimeStampToken* nebo detail chyby, který je obsažen ve Statusu, pokud byla žádost odmítnuta. Pokud se odpověď z časové autority ukládá do souboru, zpravidla se používá přípona .tsr. [1]

2.2.1 Status



Obr. 3 Schéma statusu

Status je objekt typu, jak je nazván v normě RFC-3161, *PKIStatusInfo*. Uvnitř objektu se nachází status kód, který indikuje stav, jakým dopadlo vydání časového razítka (viz. kapitola 2.2.1.1). Status kód musí být vždy ve Statusu uveden. Dále se zde nachází *StatusString*, který se odvíjí od Status kódu. Je to textový řetězec, který slovně popisuje význam daného statusu. *StatusString* je volitelná položka a ve Statusu nemusí být vůbec uveden. Poslední položkou je informace o zamítnutí (*Fail info*). Informace o zamítnutí obsahuje dodatečné textové informace o důvodu zamítnutí žádosti o časové razítko (viz kapitola 2.2.1.2). Informace o zamítnutí je nepovinná položka, tudíž ve statusu nemusí být nutně obsažena. [2]

2.2.1.1 Status kódy

Status kódy obsahují 6 stavů, které můžou nastat. Číselné označení stavů začíná číslicí 0:

- Granted (0, vydáno) – časové razítko bylo bez problému vydáno. Pokud je status Granted, odpověď časového razítka musí obsahovat objekt *TimeStampToken*;
- Granted with modification (1, vydáno s modifikacemi) – časové razítko bylo úspěšně vydáno, ale s jistými modifikacemi. Odpověď časového razítka obsahuje objekt *TimeStampToken* s modifikacemi;
- Rejected (2, zamítnuto) – žádost o časové razítko byla zamítnuta a časové razítko nebylo vydáno. Odpověď neobsahuje objekt *TimeStampToken*;
- Waiting (3, čeká se na vydání) – žádost o časové razítko se stále zpracovává. Odpověď neobsahuje *TimeStampToken*;
- RevocationWarning (4, varování o zrušení) – varování, že vydání časového razítka bude zrušeno. Odpověď neobsahuje objekt *TimeStampToken*;

- RevocationNotification (5, oznámení o zrušení) – vydání časového razítka bylo zrušeno. Odpověď neobsahuje objekt *TimeStampToken* [2].

Při vydávání razítek se žadatel nejčastěji setká se stavy 0–2. Stavy 3–5 se vyskytují jen výjimečně.

2.2.1.2 Dodatečné informace o zamítnutí

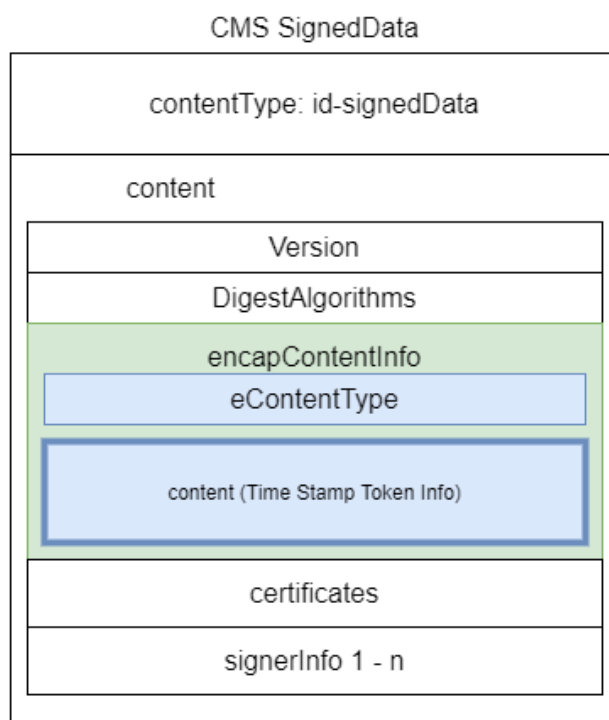
Dodatečné informace o zamítnutí můžou být přítomné, pokud vydání časového razítka nebylo vydáno a odpověď z časové autority neobsahuje objekt *TimeStampToken*. Dodatečné informace o zamítnutí žadateli sdělují, z jakého důvodu bylo časové razítko nejspíše zamítnuto.

- badAlg (0) – časová autorita nerozpoznala nebo nepodporuje použitý algoritmus pro vytvoření otisku zprávy nebo dokumentu, ke kterému mělo být vytvořeno časové razítko;
- badRequest (1) – žádost o časové razítko nebyla povolena nebo není podporována;
- badDataFormat (5) – žádost o časové razítko je ve špatném formátu (např. není vyplněna nějaká povinná položka);
- timeNotAvailable (14) – zaručený zdroj času, od kterého časová autorita odebírá přesný čas, není k dispozici, tudíž časové razítko nemohlo být vytvořeno;
- unacceptedPolicy (15) – požadovaná politika časové autority, která byla uvedena v žádosti o časové razítko není podporována časovou autoritou;
- unacceptedExtension (16) – požadované rozšíření, které mělo obsahovat časové razítko, není podporováno časovou autoritou;
- addInfoNotAvailable (17) – dodatečné informace, které byly obsaženy v žádosti o časové razítko, nebyly rozpoznány nebo nejsou dostupné na časové autoritě;
- systemFailure (25) – k vydání časového razítka nedošlo z důvodu systémové chyby, která nastala na straně časové autority [2].

2.3 TimeStampToken

Obsah časového razítka není dostupný ihned jako objekt, který bude následně popsán, ale je schován uvnitř tzv. *CMS Signed-Data*. CMS je standart pro digitální podpis. Obsah časového razítka (*TimeStampToken*) je tedy uložen do CMS zprávy. Tahle CMS zpráva obsahuje dvě položky, *contentType* a *content*. *ContentType* obsahuje identifikátor objektu, který se ukrývá v položce *content*. Objekt *content* obsahuje spoustu dalších položek, ale pro téma časových

razítek je nejdůležitější objekt *encapContentInfo*, který se opět skládá ze dvou dalších objektů, *eContentType* a *eContent*. *EContentType* je opět identifikátorem, co obsahuje objekt *eContent*. V případě časových razítek, *eContentType* obsahuje identifikátor, který udává, že se jedná o časové razítko a *eContent* již obsahuje časové razítko a informace o vydaném časovém razítku (*TSTInfo* – Time Stamp Token Info). Detail celého objektu CMS Signed Data je pro lepší představu zobrazen na obr. 4 – datová struktura objektu CMSSignedData. Pokud se token časového razítka ukládá na disk, používá se zpravidla přípona .tst. [1]



Obr. 4 Datová struktura objektu CMS SignedData (vlastní tvorba)

Nyní je čas podívat se na to co obsahuje token vydaného časového razítka. Token časového razítka může obsahovat následující položky: *version*, *policy*, *messageImprint*, *serialNumber*, *genTime*, *accuracy*, *ordering*, *nonce*, *tsa* a *extensions*. [2]

2.3.1 Version

Objekt *version* obsahuje použitou verzi protokolu TSP. Je to stejná položka, která se nachází i v žádosti o časové razítko. [2]

2.3.2 Policy

Policy udává politiku časové autority, pod kterou bylo časové razítko vydáno. Pokud si v žádosti žadatel specifikuje politiku, pod kterou si přeje, aby bylo časové razítko vydáno

a časová autorita danou politiku podporuje, potom se tedy hodnota objektu *policy* musí shodovat s tou, která byla uvedena v žádosti o časové razítko (viz kapitola 2.1.2.1). [2]

2.3.3 MessageImprint

Tenhle objekt musí být opět shodný se stejnojmenným objektem, který se udává v žádosti. Je zde totiž uložen typ algoritmu a otisk dokumentu (zprávy), pro který bylo časové razítko vytvořeno. [2]

2.3.4 SerialNumber

SerialNumber, neboli sériové číslo, je identifikátor časového razítka, který musí být přidělen každému časovému razítku, které časová autorita vydala. Název časové autority a sériové číslo dávají časovému razítku unikátní identifikaci, která nesmí obsahovat žádné jiné časové razítko. Může nastat situace, kdy dvě časová razítka vydaná různými časovými autoritami mají stejná sériová čísla, ale po přidání jména časové autority je tento název unikátní, protože nesmí existovat dvě časové autority se stejným názvem. [2]

2.3.5 GenTime

GenTime (zkratka Generation Time) je čas ve kterém bylo časové razítko vytvořeno časovou autoritou. Čas vygenerování je udáván ve formě UTC. Čas se uvádí v UTC, aby se předešlo zmatkům ohledně lokálních časových zón. Tento čas časová autorita přebírá od poskytovatelů zaručeného času (více informací o poskytovatelích zaručeného času se nachází v kapitole 3.2). [2]

2.3.6 Accuracy

Accuracy (přesnost) udává, jak přesný je čas uvedený v objektu *GenTime*. *Accuracy* obsahuje tři položky: seconds, milis a micros. Všechny tři položky jsou volitelné, ale alespoň jedna z nich by měla být uvedena. *Accuracy* udává maximální hodnotu v sekundách, milisekundách, nebo mikrosekundách. Když je přesnost např. 1 ms a čas vygenerování je 5.3.2020 21:33:05.500, tak příslušná přesnost říká, že vytvoření časového razítka proběhlo mezi 5.3.2020 21:33:05.499 a 5.3.2020 21:33:05.501. [2]

2.3.7 Ordering

Objekt *Ordering* je typu BOOLEAN a pokud má hodnotu TRUE, tak specifikuje, zdali je čas uvedený v časových razítkách tak přesný, že pomocí něj mohou být vydaná razítka

seřazena v pořadí, jak byla vydána. Položka *Ordering* je nepovinná a nemusí být v tokenu časového razítka uvedena. [2]

2.3.8 Nonce

Nonce je nepovinná položka a v odpovědi časového razítka musí být uvedena pouze v případě, pokud byl *nonce* uveden i v žádosti o časové razítko. *Nonce* je detailně rozebrán v kapitole 2.1.2.2. [2]

2.3.9 Tsa

Položka *tsa* je také nepovinná položka. Objektem této položky je dát náповědu v identifikaci použité časové autority. Pokud je *tsa* obsažena v tokenu časového razítka, musí odpovídat jménu subjektu certifikátu, který byl použit při ověření časového razítka. [1]

2.3.10 Extensions

Extensions (rozšíření) je obecná cesta, jak k tokenu časového razítka přidat dodatečné informace. Položka *extensions* je nepovinná a v tokenu časového razítka nemusí být uvedena. [1]

2.4 Platnost časového razítka

Každý certifikát musí mít nějakou platnost. A protože časové razítko je podepsaná datová struktura, kterou vytvořila časová autorita a podepsala jej právě svým certifikátem, který slouží jen pro vydávání časových razítek, tak platnost časového razítka je stejně dlouhá jako platnost certifikátu časové autority. Časové autority si musí hlídat platnost svého certifikátu a obvykle certifikát na vytváření časových razítek vytvářejí s předstihem a svým zákazníkům poskytují službu obnovení časového razítka, kdy se vytvoří nové časové razítko z dat původního časového razítka a podepíše se novým certifikátem. [1]

2.5 Proces tvorby časového razítka

Nyní, když byly rozebrány žádost o časové razítko (viz. kapitola 2.1) a odpověď časového razítka (viz. Kapitola 2.2), lze popsat, jak jde tvorba časového razítka chronologicky za sebou. Aby mohl žadatel vytvořit časové razítko, musí mít předplacenou službu vytváření časových razítek od placeného poskytovatele časových razítek nebo musí použít poskytovatele, který umožňuje zdarma vytvořit časové razítko.

Nejprve si žadatel musí vybrat soubor nebo zprávu. Všeobecně nějaké data, ke kterým chce vytvořit časové razítko a hashovací algoritmus, pomocí kterého bude vypočten otisk dokumentu. Dále si ještě může zvolit, zda chce použít konkrétní politiku časové autority a zda chce do žádosti vygenerovat *Nonce*. Podle zvoleného hashovací algoritmu se vypočte otisk dokumentu, vloží se zároveň s OID hashovacího algoritmu do žádosti, vloží se zde volitelné položky, pokud je žadatel zadal. Nejčastěji se žádost vloží do http požadavku a odešle se na server časové autority.

Na serveru časové autority žádost přichází na front end časové autority, zde se kontroluje, zda je vyplněna položka *Politika časové autority*. Pokud není, žádost přechází na dostupnou TSU. Pokud je a časová autorita podporuje požadovanou politiku, žádost přechází na TSU, která má příslušnou politiku. Pokud není k dispozici, front end žádost zamítá a vrací žadateli odpověď s příslušnou chybovou hláškou. V TSU se provede pouze kontrola, zda sedí délka otisku s použitým hashovacím algoritmem. Pokud kontrola proběhne úspěšně, vytvoří se token časového razítka, ten se vloží do odpovědi časového razítka a zašle se zpět žadateli (opět nejčastěji pomocí http).

Po přijetí odpovědi od serveru časové autority by se celá odpověď nebo token časového razítka měl uložit do stejného adresáře ve kterém se nachází soubor, ke kterému bylo časové razítko vytvářeno. [1, 2]

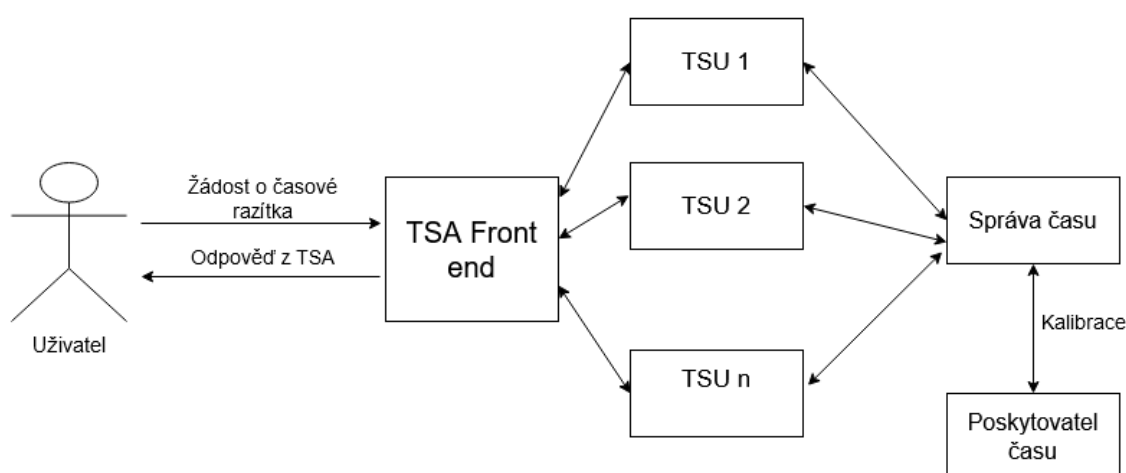
2.6 Proces ověření časového razítka

Protože je token časového razítka typu *CMS Signed data*, tzn. že token je digitálně podepsán vydanou časovou autoritou, je potřeba tento podpis ověřit pomocí certifikátu dané autority. Poté je potřeba ověřit, zda certifikát časové autority vydala důvěryhodná certifikační autorita. Takhle je nutno s ověřováním certifikátu dojít až ke kořenovému certifikátu. Pokud je certifikát časové autority platný, musí se ověřit podpis, kterým bylo podepsáno časové razítko. Pokud je ověření podpisu platné, musí se ověřit otisky dokumentu, tedy znovu vypočíst otisk dokumentu, ke kterému bylo časové razítko vytvořeno a porovnat jej s otiskem, který je uveden v tokenu časového razítka. Pokud žadatel vložil požadovanou politiku časové autority a *Nonce* do žádosti, tyhle položky se musí rovnat těm, které zadal v žádosti o časové razítko. Pokud všechny zmíněné kontroly projdou, lze prohlásit že ověřované časové razítko je validní a dokument je v totožném stavu jako v čase kdy vzniklo časové razítko. [1]

3 AUTHORITY PRO VYDAVÁNÍ ČASOVÝCH RAZÍTEK

Autority pro vydávání časových razítek (anglicky Time Stamp Authority – TSA) jsou objekty, které zákazníkům mohou poskytnout časové razítko po zaslání validní žádosti. Pro úvodní zjednodušení si lze TSA představit jako úředníka, který orazítkuje jakýkoliv dokument, co mu přijde pod ruku. To ale ve skutečnosti samozřejmě není pravda. Každá TSA se skládá z více částí, které mezi sebou komunikují. [1]

3.1 Typická architektura TSA



Obr. 5 Schéma TSA (vlastní tvorba)

Na obr. 5 lze vidět, jak vypadá typické schéma TSA. Poté co uživatel vytvoří žádost o časové razítko (viz kapitola 2.1) nejčastěji pomocí komunikace HTTPS přichází žádost na front-endovou část. Odtud poté žádost postupuje ke zpracování do konkrétní TSU (Time Stamp Unit), což je množina hardwaru a softwaru, která vyřizuje žádosti o časové razítka. Vyřízení žádosti se skládá ze dvou základních částí, a to vydání časového razítka a jeho následné podepsání pomocí privátního klíče. Každá TSU má zpravidla svou vlastní politiku vydávání časových razítek, a každé této politice se přiřadí jedinečný identifikátor objektu (OID), který je poté uložen do časového razítka. [1]

3.2 Pravidla požadované normou RFC 3161 po TSA

Aby TSA mohla fungovat a vydávat časová razítka, musí splňovat určité podmínky, přesněji 11 pravidel, kterým se přezdívá jako „jedenáct základních příkázání TSA“. Těchto jedenáct pravidel zní:

- 1) **TSA musí používat důvěryhodný (zaručený) zdroj času** – na světě existuje síť laboratoří, které mají specializované prostředky, pomocí kterých měří čas. Všechny laboratoře mezi sebou své časy koordinují, a z toho vychází tzv. střední hodnota, která je poté poskytována lidem. Laboratoře samozřejmě poskytují čas veřejnosti zdarma (např. pomocí rozhlasového a televizního vysílání, GPS, rozhlasových vln), TSA však používají tzv. čas zaručený, který však již zdarma často není. [1]
- 2) **Vkládat důvěryhodný čas do každého vydaného časového razítka** – TSA je povinna do každého vytvořeného časového razítka vložit zaručený čas. [1]
- 3) **Každé vydané časové razítko musí mít jedinečné pořadové číslo** – každé vydané časové razítko v rámci jedné TSA musí mít své jedinečné pořadové číslo. Pokud jedna TSA má více TSU, je potřeba ošetřit, aby TSU nevydali časová razítka se stejným pořadovým číslem. [1]
- 4) **TSA vydá časová razítka vždy, když je to možné, po obdržení platné žádosti o časové razítko** – z tohoto pravidla si lze vyvodit, že jediná věc, kterou je potřeba dodržet, je podat správnou žádost o časové razítko. Myšlenka je to správná, ale i časové autority musí mít nějaké výdělky, proto se často služby časové autority zpoplatňují. Praxe je často taková, že si zákazník, který potřebuje vytvářet časová razítka objedná určitý balíček o určitém množství časových razítek a v balíčku obdrží i nějaké autentizační údaje, aby službu mohl čerpat (např. autentizační certifikát nebo přihlašovací údaje). [1]
- 5) **Do každého vydaného časového razítka vložit identifikátor politiky (OID) TSA pod kterou bylo razítko vydáno** – v každém časovém razítku musí být obsažen OID dané TSU, která časové razítko vytvořila (viz. kapitola 2.2) [1]
- 6) **Algoritmus, který byl použit pro výpočet otisku dokumentu musí být označen svým OID** – tzn. že časové razítko nesmí obsahovat název použitého algoritmu pro výpočet otisku, ale jeho OID (např. algoritmus SHA-256 má OID 2.16.840.1.101.3.4.2.1) [3]
- 7) **Zkontrolovat délku otisku, zda odpovídá použitému otisku** – délka otisku musí odpovídat použitému algoritmu pro vytvoření otisku. Např. když je použit pro vytvoření otisku algoritmus SHA-256, otisk musí být dlouhý 256 bitů (32 Byte) [3]
- 8) **Kontrolovat pouze délku otisku** – časová autorita nesmí nijak analyzovat otisk, např. že by se pokusila k otisku najít originální dokument [2]

- 9) **Nevkládat do časového razítka identifikaci žadatele** – i přes to, že služba časových autorit je často zpoplatněna, časové razítko nesmí obsahovat žádnou identifikaci o žadateli (např. uživatel se ke službě přihlašuje pomocí přihlašovacích údajů, tak by se do časového razítka vložilo přihlašovací jméno žadatele) [2]
- 10) **K podepsání časových razítek se musí použít klíč výhradně určený pro tento účel** – klíče, které se používají v oboru mají často svůj nějaký důvod použití, který se nastaví při jeho vygenerování (např. šifrování, dešifrování, podepsání, ověření podpisu – časové razítko se nesmí podepsat klíčem, který se používá na ověření podpisu) [1]
- 11) **Pokud je v žádosti uvedeno rozšíření, které časová autorita nepodporuje, časové razítko nesmí být vydáno** – žadatel si do žádosti může vložit požadované rozšíření (viz. kapitola 2.1). [1]

4 APLIKACE PRO SPRÁVU ČASOVÝCH RAZÍTEK

Pro seznámení s dostupností aplikací pro správu časových razítek byl proveden velice krátký průzkum na internetu, jehož cílem bylo najít veřejně dostupné aplikace, které umožňují vytvářet a různě spravovat časová razítka. Výsledkem průzkumu byla jen jedna aplikace, která umožňovala časová razítka jen vytvářet. Důvodem, proč nelze najít spoustu aplikací, které umí spravovat časová razítka je fakt, že tyto aplikace dostanou uživatelé často jako součást balíčku, který si zakoupí u poskytovatele časových razítek. Proto tyto aplikace nejsou často veřejně dostupné.

Často se vytvářejí dva typy aplikací pro správu časových razítek:

- Prvním typem jsou aplikace pro koncové uživatele. To jsou aplikace, které mají GUI rozhraní, přes které uživateli umožňují provádět základní a často i nějaké funkce navíc, které jsou uvedeny v kapitole 4.1;
- Druhým typem jsou knihovny, které se můžou integrovat do aplikací třetích stran a v těchto aplikacích umožňují provádět základní funkce, které by aplikace pro správu časových razítek měly obsahovat (viz. kapitola 4.1).

4.1 Základní funkce

Základní požadavky, které by měly aplikace splňovat jsou operace, které jsou rozebrány v kap. 2 a normě RFC-3161:

- Vytvořit žádost o časové razítko – aplikace by měla umožnit uživateli vybrat soubor nebo zprávu ke které chce vytvořit časové razítko, umožnit mu zvolit algoritmus otisku dokumentu, umožnit uživateli zadat požadovanou politiku časové autority a možnost použití atributu Nonce.
- Odeslat žádost – aplikace by měla umožnit odeslat vytvořenou žádost o časové razítko na vybranou webovou adresu poskytovatele časových razítek, např. pomocí http nebo https požadavku.
- Přečíst odpověď časové autority – aplikace by měla umět přečíst odpověď časové autority, z validovat status odpovědi, podle kterého buď pokračovat validací časového razítka, nebo zobrazením chybové hlášky.
- Z validovat token časového razítka – přijatý token z validovat podle kroků, které jsou uvedeny v kapitole 2.6.

- Uložení platného časového razítka – pokud validace tokenu časového razítka proběhne v pořádku, měla by aplikace uložit data časového razítka jako soubor s příponou (.tsr nebo .tst) do stejného adresáře, kde se nachází soubor, ke kterému bylo časové razítko vytvářeno.

4.2 Doplnující funkce

Doplnující funkce nejsou kriticky nutné pro aplikace na správu časových razítek, ale uživatelům mohou zpříjemnit zážitek z použití aplikace a ulehčit mu práci. Tyhle funkce se můžou přidávat do aplikací i na požadavek uživatele.

Zde je seznam doplňujících funkcí, které mohou být pro uživatele užitečné a ulehčit jim práci s časovými razítky:

- Ukládání historie časových razítek – ukládání historie vydaných časových razítek může být pro uživatele velmi užitečná funkce. Pokud chce např. najít, kde má dané časové razítko uloženo, otevře si historii a v ní má zobrazen seznam vydaných časových razítek. Díky historii tedy velmi rychle a efektivně nalezne požadované časové razítko.
- Zobrazení tokenu časového razítka – v detailu tokenu časového razítka může uživatel nalézt všechny hodnoty časového razítka (viz. kapitola 2.3).
- Včasné přerazítkování časových razítek, kterým končí platnost – jak bylo zmíněno v kapitole 2.4, časové razítko má svou dobu platnosti. Aplikace by tedy kontrolovala platnost časového razítka a v čase před ukončením platnosti by vytvořila nové časové razítko, které by nahradilo staré (tzv. přerazítkování).
- Filtrování historie vydaných časových razítek – pokud počet vydaných razítek nabírá na počtu je vhodné uživateli usnadnit vyhledávání v historii pomocí různých filtrů (např. podle poskytovatele, použité politiky, rozmezí data vytvoření atd.).
- Dodatečné ověření časového razítka – další užitečná funkce je dodatečné ověření časového razítka. Uživatel najde již vydané časové razítko a může si jej pomocí aplikace ověřit, popř. přidat do historie vydaných razítek.

II. PRAKTICKÁ ČÁST

5 MOŽNOSTI IMPLEMENTACE APLIKACE

Možností, jak implementovat aplikaci, je velmi mnoho a základní rozdělení možností implementace je podle zařízení, na jaké má aplikace cílit. Aplikace může být vytvořena pro desktopy, kde lze rozdělení rozvést dle operačního systému, tedy jestli má aplikace cílit jen na Windows, nebo má být multiplatformní, tedy být funkční i na linuxových operačních systémech. Dále si lze na desktopových systémech zvolit, zda se bude jednat o aplikaci s grafickým rozhraním, nebo se bude jednat jen o .dll knihovnu. Další zařízení, na které by mohla být aplikace vytvořena, jsou mobilní zařízení. Zde lze opět řešení rozvést dle operačního systému, tedy zda by aplikace byla čistě jen na Android či iOS, nebo by byla implementována tzv. hybridně, tedy na oba operační systémy zároveň.

5.1 Aplikace pro desktopové systémy

Aplikace pro desktopové systémy by se mohly vytvořit dvěma způsoby. Prvním způsobem je vytvoření aplikace pro použití v příkazové řádce a druhý způsob je vytvořit aplikaci s grafickým rozhraním.

5.1.1 CLI aplikace

CLI (Command Line Interface) je prvním způsobem vytvoření aplikace. Aplikace by se celá ovládala z příkazového řádku operačního systému a všechny dostupné funkce, které by aplikace obsahovala, by se musely volat pomocí klíčových slov a parametrů. Např. pro funkci vytvoření časového razítka by se do příkazového řádku napsal příkaz `createTimeStamp „zdrojový soubor“ „server časové autority“`. Tenhle způsob aplikace je velmi neefektivní, ale jedná se o funkční řešení, proto je zde uvedeno. Aplikace by se mohla naprogramovat v jazyku C# .NET Framework, pokud se by aplikace měla cílit pouze na systém Windows a C# .NET Core, nebo C++ pokud by aplikace měla být multiplatformní.

5.1.2 Aplikace s grafickým rozhraním

Aplikace s grafickým rozhraním bude vhodnější řešení než aplikace v příkazové řádce. Díky grafickému rozhraní uživatel nebude muset studovat různé příkazy pro různé funkce aplikace a bude moci jednoduše klikat na tlačítka a vyplňovat textová pole, což je pro něj mnohem jednodušší. Pro vytvoření aplikace pro správu časových razítek je velký výběr programovacích jazyků a jejich grafických knihoven:

- C# – lze použít grafickou knihovnu WinForms, nebo její novou verzi WPF, která umožňuje tzv. Data-Binding ve vývojovém prostředí Visual Studio od společnosti Microsoft;
- C++ – s použitím grafické knihovny od společnosti Qt ve vývojovém prostředí Qt Creator;
- Java – pro programovací jazyk Java lze použít vývojové prostředí od společnosti JetBrains IntelliJ IDEA, které nabízí i grafický designér pro vyvíjenou aplikaci, kde grafické prvky jsou z knihovny Swing;
- Python – pro grafické rozhraní lze použít knihovnu PyQt, která je nadstavbou grafické knihovny od společnosti Qt pro C++, nebo knihovnu Tkinter.

5.2 Mobilní aplikace

Aplikace pro správu časových razítek lze vytvořit i pro mobilní zařízení. Při vývoji aplikace pro mobilní zařízení je důležité nejprve se rozhodnout, jakým způsobem aplikaci vyvíjet. Aplikaci je možno vytvořit tzv. nativně, což znamená, že vývojář si vybere platformu, na jakou chce vytvořit aplikaci (Android, iOS), a použije dostupné prostředky pro vývoj aplikace, které nabízí daná platforma. Další možnost tvorby aplikace je tzv. hybridní vývoj, kdy se aplikace vyvíjí hybridně pro obě platformy najednou.

5.2.1 Nativní vývoj – Android

Nativní aplikace pro platformu Android lze vytvořit dvěma způsoby:

1. Android Studio – je oficiální vývojové prostředí pro tvorbu aplikací pro operační systém Android. Design aplikace se zde může tvořit pomocí .XML souborů, kdy se jednotlivé prvky jedné obrazovky (v Androidu se nazývá aktivita) vkládají pomocí klíčových slov jednotlivých prvků. Tenhle postup je trochu složitější, ale pro profesionální vývojáře rychlejší. Android studio také obsahuje designer, kdy si jednotlivé prvky jedné aktivity lze interaktivně naskládat. Použití designéru se hodí spíše pro začátečníky. Programová logika aplikace se poté může programovat buď v jazyce Kotlin nebo Java, kdy tým Android Developers poskytuje vývojářům knihovny, které vývojáři používají pro zjednodušení tvorby programové logiky.
2. Xamarin.Android – je nadstavba nad knihovny, které poskytuje tým Android Developers, přepsané do programovacího jazyka C# od společnosti Microsoft. Pro

vývoj aplikací se používá vývojové prostředí Visual Studio. Vzhled jednotlivých obrazovek (aktivit) se tvoří pomocí XML stejně jako v Android Studiu, ale programová logika se programuje v C# .NET Core.

5.2.2 Nativní vývoj – iOS

Vytvořit nativní aplikaci pro zařízení od společnosti Apple lze také dvěma způsoby:

1. Xcode – Xcode je vývojové prostředí pro nativní vývoj aplikací pro iOS. Vývojové prostředí je dostupné pouze na operačním systému macOS. Vzhled jednotlivých obrazovek (View) aplikace lze vytvořit interaktivně pomocí designeru, který se v Xcode jmenuje Storyboard, nebo ručně editovat soubor s příponou .xib, kde se jednotlivé ovládací prvky obrazovky vkládají pomocí jejich klíčových názvů. Logika jednotlivých obrazovek se poté programuje v tzv. Controllerech, kdy si lze vybrat z programovacích jazyků Objective-C, nebo novinky od společnosti Apple Swift.
2. Xamarin.iOS – Microsoft udělal podobnou nadstavbu knihoven od Apple do C# jako v Androidu, a díky tomu lze aplikace pro iOS vytvářet i v C#. Díky této nadstavbě lze aplikace pro iOS vyvíjet i na počítačích se systémem Windows, ale pro otestování aplikace je potřeba vlastnit nějaké zařízení se systémem iOS a vývojové prostředí Visual Studio spárovat se zařízením Mac, které se musí nacházet ve stejné síti. Vývojové prostředí Visual Studio je dostupné i na operační systém macOS, vyvíjet aplikace pomocí Xamarin.iOS lze tedy i na Mac zařízeních.

5.2.3 Hybridní vývoj mobilních aplikací

Jeden z hybridních druhů tvorby aplikací pro mobilní zařízení, pomocí kterého by bylo možno implementovat aplikaci na správu časových razítek, je použití Xamarin.Forms. Xamarin.Forms je open source architektura, která umožňuje vývojářům sestavit aplikace pro Android i iOS z jediného sdíleného základu kódu. Uživatelské rozhraní aplikace se vytváří pomocí XAML a programová logika se programuje pomocí C#. V Xamarin.Forms lze také použít Data-Binding. Po sestavení Xamarin.Forms se uživatelské rozhraní aplikace zobrazí v nativní podobě každé platformy (ve výsledku je vzhled jednotlivých uživatelských prvků jiný v Android aplikaci a iOS aplikaci). [4]

6 ZVOLENÁ METODA IMPLEMENTACE

Hlavním kritériem volby metody a prostředku, pomocí kterých je aplikace implementována, byly zkušenosti autora, jaké s nimi má. Z popsanych možností, které jsou uvedeny v kapitole 5, autor zvolil použít programovací jazyk C#, s použitím knihoven .NET Framework ve verzi 4.7.2 (v době implementace aplikace se jednalo o nejnovější verzi .NET Frameworku). Grafické rozhraní je vytvořeno pomocí grafických knihoven WPF. Aplikace je vytvořena pomocí architektury MVVM (Model-View-ViewModel). Vývoj aplikace probíhal ve vývojovém prostředí od společnosti Microsoft Visual Studio 2017 a později ve verzi 2019.

6.1 .NET Framework

.NET Framework je technologie, která umožňuje sestavení a spouštění aplikací nebo webových služeb. .NET Framework se skládá ze dvou základních částí, CLR (Common Language Runtime) a knihovny tříd .NET Framework. .NET Framework lze použít k tvorbě aplikací, které poběží pouze v operačním systému Windows. [5]

6.1.1 CLR

Hlavními úkoly CLR jsou správa paměti, provádění vláken a kódu, ověřování a zabezpečení kódu, kompilaci a ostatní systémové služby. CLR umožňuje odchyťovat výjimky, které mohou nastat v průběhu provádění kódu a obsahuje tzv. Garbage Collector, což je forma automatické správy paměti, která automaticky prohledává paměť a uvolňuje části paměti, které již program nevyužívá. Programátor se díky tomu nemusí vlastnoručně starat o uvolňování paměti, jako např. v programovacím jazyku C/C++. CLR dále umožňuje vzájemnou spolupráci mezi spravovaným a nespravovaným kódem, což umožňuje vývojářům v používání komponent modelu COM a knihoven DLL. COM umožňuje vývojářům např. použití funkcí z knihoven, které byly napsány v programovacím jazyce C/C++. [5]

6.1.2 Knihovna tříd

Knihovna tříd obsahuje velkou kolekci typů, které se úzce integrují s CLR. Knihovna tříd je objektově orientovaná. Vývojářův kód si odvozuje funkcionalitu ze tříd, které poskytuje tato knihovna a díky tomu je usnadněno použití typů v .NET Framework a také to snižuje dobu naučení se nových funkcí. Různé třídy z knihovny .NET Frameworku umožňují např. připojení k databázi, vytváření a upravování souborů, manipulace s XML soubory atd.

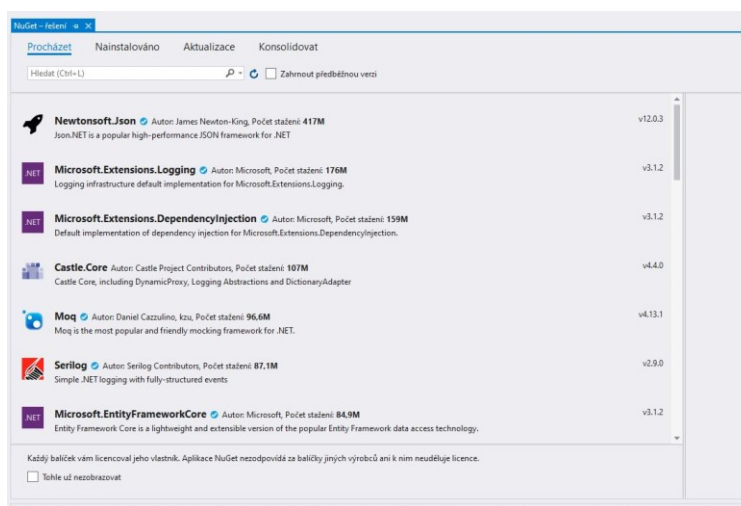
Pomocí knihoven tříd lze vyvíjet např. konzolové aplikace, aplikace s grafickým rozhraním (Windows Forms, WPF nebo UWP) a webové aplikace pomocí ASP.NET. [5]

6.1.3 NuGet balíčky

NuGet je mechanismus vytvořený společností Microsoft, pomocí kterého můžou vývojáři vytvářet, sdílet a využívat užitečný kód, který jim usnadní vývoj jejich aplikací, a díky těmto balíčkům nemusí vymýšlet a vytvářet kód, který již někdo vytvořil. Tento mechanismus specifikuje, jak se tyto balíčky kódu, které jsou typicky v podobě DLL knihoven, vytvářejí, hostují a následně používají v aplikacích vytvořených pomocí .NET Frameworku nebo novějšího a nově i multiplatformního .NET Core. [6]

NuGet balíčky vytvářejí buďto open source vývojáři, kteří své vytvořené balíčky poskytují zdarma. Existují také balíčky, které jsou zpoplatněny. Pro použití zpoplatněných balíčků je potřeba navštívit webové stránky tvůrců balíčků a zakoupit si licenci. Po zakoupení licence lze poté produkt aktivovat nejčastěji pomocí licenčního souboru, který se vloží do projektu nebo zadáním licenčního klíče (nejčastěji textový řetězec) při spuštění aplikace pomocí nějaké metody nebo vlastnosti třídy, která se stará o kontrolu, zda má použitý balíček platnou licenci či ne.

Vývojové prostředí Visual Studio má integrovanou funkci „Manage NuGet Packages“, kde lze vyhledávat, instalovat nebo odinstalovat dostupné NuGet balíčky, které se nacházejí v centrálním repozitáři, který je dostupný k prohlédnutí na webové stránce nuget.org. [6]



Obr. 6 Prohlížeč NuGet balíčků ve VS

NuGet balíčky jsou zde zmíněny, protože vytvořená aplikace pro správu časových razítek používá tyto balíčky. V aplikaci jsou použity dva NuGet balíčky a oba jsou zdarma:

1. Org.BouncyCastle – je knihovna, která obsahuje třídy, které vývojářům umožňují pracovat s certifikáty, poskytují generátory šifrovacích klíčů, třídy pro šifrování a dešifrování, ať už symetrických nebo asymetrických šifrovacích algoritmů a mnoho dalších funkcí. Balíček taky obsahuje třídy, které podle RFC-3161 umožňují vytvářet žádost o časové razítko (*TimeStampRequest*), přečíst odpověď časové autority (*TimeStampResponse*), která umožňuje z validovat odpověď časové autority i token časového razítka. Třída obsahuje i naparsované informace z tokenu časového razítka (*TimeStampToken*). Díky balíčku Org.BouncyCastle lze jednoduše vytvářet žádosti a poté zobrazit detailní informace o vydaných časových razítkách.
2. log4net – je balíček, který se stará o logování aplikace. Vývojář díky balíčku log4net nemusí vytvářet svou logovací třídu, ale vždy jen v každé třídě, ve které potřebuje logovat nějaké informace, inicializuje instanci třídy log4net.ILog. Log4net lze nakonfigurovat tak, aby logoval do souboru, nebo do konzole. Konfigurační soubor je ve formátu XML a může být v projektu vložen samostatně, nebo může být součástí globálního konfiguračního souboru app.config.

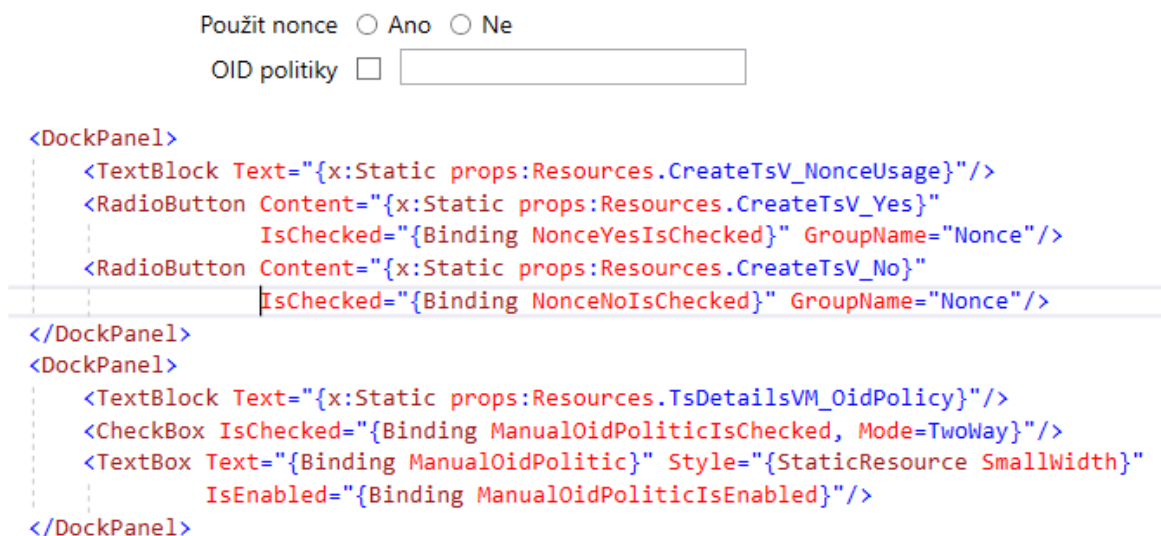
6.2 WPF

WPF (Windows Presentation Foundation) je framework k vytváření aplikací s grafickým uživatelským rozhraním na platformě .NET Framework od společnosti Microsoft. WPF obsahuje širokou škálu funkcí pro vývoj aplikací, včetně aplikačního modelu, ovládacích prvků, datových vazeb, zobrazení 2D a 3D grafiky. Umožňuje vytvářet animace a styly k jednotlivým prvkům. WPF podporuje Hardwarovou akceleraci, která výrazně zvyšuje výkon graficky náročnějších aplikací. WPF aplikace lze navrhovat pomocí jazyka XAML (Extensible Application Markup Language) a programovou logiku psát v tzv. kódu na pozadí.

6.2.1 XAML

XAML je značkovací jazyk odvozený od jazyka XML, který deklarativně implementuje vzhled aplikace. Pomocí jazyka XAML lze vytvořit grafické rozhraní celé aplikace. XAML obsahuje mnoho elementů, které definují různé prvky uživatelského rozhraní. Např. element Button pro tlačítko, TextBox pro textové pole, kde může uživatel vkládat obsah, kontejnery,

které jednoduše ukládají uživatelské prvky pod sebe (StackPanel) nebo vedle sebe (DockPanel) a mnoho dalších značek.



Obr. 7 Ukázka jazyka XAML

Uživatelské rozhraní vytvořené v jazyku XAML je sestaveno v hierarchii vnořených elementů, které ve výsledku tvoří stromovou strukturu. Díky tomu je docíleno jednoduché orientace v kódu. Hierarchie vnořených elementů lze vidět i na Obr. 7, kdy element DockPanel má v sobě vnořeny elementy TextBlock a dva elementy RadioButton. [7]

6.2.2 Kód na pozadí

Úkolem kódu na pozadí je implementovat funkcionalitu aplikace, např. reagovat na interakce uživatele, kdy např. po stisku tlačítka se provede nějaká operace (vytvoření časového razítka, ověření časového razítka atd.). Kód na pozadí je propojen s jednotlivými prvky, které jsou vytvořeny pomocí jazyka XAML a když se jednotlivým elementům nastaví atribut Name, tak podle nastaveného atributu Name lze k prvku přistupovat a manipulovat s ním v kódu na pozadí. Automatické propojení rozhraní vytvořeného v jazyce XAML a kódu na pozadí zajistí metoda InitializeComponent, která se vždy nachází v konstruktoru třídy kódu na pozadí (metoda lze vidět na Obr. 8). [7]

```
MainWindow.xaml
<Grid>
  <StackPanel>
    <TextBox x:Name="textBox" Text="Ahoj"/>
    <Button x:Name="button" Content="Změn text" Click="button_Click"/>
  </StackPanel>
</Grid>

MainWindow.xaml.cs
public partial class MainWindow : Window
{
    Počet odkazů: 0
    public MainWindow()
    {
        InitializeComponent();
    }

    Počet odkazů: 0
    private void button_Click(object sender, RoutedEventArgs e)
    {
        textBox.Text = "Změna textu";
    }
}
```

Obr. 8 Ukázka kódu na pozadí

Na obrázku lze vidět událost s názvem `button_Click`, která je přiřazena tlačítku se jménem `button`. Vždy když uživatel klikne na tlačítko `button`, provede se událost `button_Click`. V události `button_Click` je znázorněno, že v kódu na pozadí lze přistupovat k jednotlivým uživatelským prvkům pomocí hodnoty v atributu `x:Name`. Příklad na Obr. 8 změní text v uživatelském prvku `TextBox` s názvem `textBox`.

Pomocí tohoto postupu lze naprogramovat jakoukoliv aplikaci včetně aplikace pro správu časových razítek. Tento způsob programování se dnes však moc neuplatňuje, protože vznikl nový způsob vývoje aplikací ve WPF, který si klade za cíl oddělit programovou logiku od grafického rozhraní. K tomu slouží tzv. Data-Binding a návrhový vzor MVVM.

6.3 MVVM a Data-Binding

Návrhový vzor MVVM (Model-View-ViewModel) nabízí efektivní řešení, jak oddělit programovou logiku aplikace od grafického rozhraní a zpřehlednit díky tomu celý kód. Vzorek MVVM také dost urychluje vývoj aplikace, protože v jednom čase může designér vytvářet grafické rozhraní a vývojář implementovat programovou logiku. Dále ještě MVVM ulehčuje testování programové logiky, kdy lze naprogramovanou logiku jednoduše otestovat např. pomocí unit testů. [8]

6.3.1 Vrstvy MVVM

MVVM se skládá ze tří vrstev. View, Model a ViewModel:

1. View – vrstva grafického rozhraní aplikace. Tvoří se pomocí jazyka XAML. Na stránku se přidají všechny uživatelské prvky, které jsou potřeba. U MVVM už není nutno ovládacím prvkům, ze kterých např. potřebujeme získat jejich obsah, nastavovat argument `x:Name`, ale vstupy, se kterými aplikace potřebuje pracovat se napojí („nabindují“) na vybranou Property ve ViewModelu, viz kap. 6.3.2.
2. ViewModel – je klasická C# třída, která udržuje stav obrazovky aplikace. ViewModel obsahuje různé Properties (Vlastnosti), které budou sloužit k propojení s jednotlivými uživatelskými prvky. Properties mohou být napojeny např. na prvek `TextBox`, kdy si daná Property bude udržovat text, který je v daném `TextBoxu` uložen, ale Property si může uchovávat i událost, která má nastat po stisknutí tlačítka, tzv. `Command`. ViewModel musí implementovat rozhraní `INotifyPropertyChanged`. Rozhraní obsahuje událost `PropertyChanged`, která indikuje View, že se hodnota Property změnila a View se musí aktualizovat, aby zobrazovalo aktuální data. Dále ViewModel často obsahuje Properties typu `ObservableCollection<T>`, což je kolekce, podobně jako `List` nebo pole, ale navíc implementuje rozhraní `INotifyCollectionChanged`. Tato kolekce si automaticky upozorní View, že se její obsah změnil a View si aktualizuje obsah. Třída `Observable Collection` se často napojuje na tabulky, nebo uživatelské prvky, které umožňují výběr z více položek.
3. Model – popisuje data, se kterými aplikace pracuje. Model je také třída, ale neměla by obsahovat žádnou logiku, jen Properties, Modely často slouží jako typ obsahu dat, které obsahuje kolekce typu `ObservableCollection<T>`. Např. Ve View je tabulka, která má sloupce `Jméno`, `Příjmení` a `Datum narození`. Tabulce se nastaví pomocí propojení zdroj typu `ObservableCollection<T>`, která bude uchovávat data Modelové třídy, a ta bude mít Properties, které jsou totožné se sloupci v tabulce. Modely neobsahují žádnou jinou programovou logiku. [8]

6.3.2 Data-Binding

Data-Binding je název pro propojení jednotlivých vrstev návrhového vzoru MVVM. Pro použití Data-Bindingu je nutno nejprve propojit View s ViewModelem. Toho se docílí nastavením Property `DataContext`, kterou obsahuje každé View. Této Property se přiřadí právě instance třídy daného ViewModelu. Property `DataContext` lze nastavit dvěma

způsoby. Buď pomocí jazyka XAML, anebo pomocí kódu na pozadí. Způsob pomocí kódu na pozadí je častěji používán, protože konstruktory ViewModelů často mají parametrické konstruktory a ViewModel s parametrickým konstruktorem již nelze nijak nastavit jako DataContext pomocí jazyka XAML.

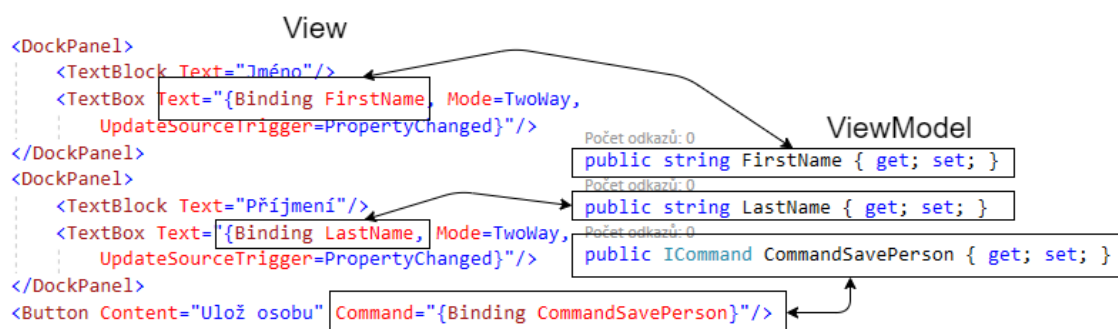
```
<Window.DataContext>
    <local:MainWindowViewModel />
</Window.DataContext>
```

Obr. 9 Nastavení DataContextu pomocí XAML

```
public MainWindow()
{
    this.DataContext = new MainWindowViewModel();
    InitializeComponent();
}
```

Obr. 10 Nastavení DataContextu pomocí kódu na pozadí

Po nastavení DataContextu je propojení View a ViewModelu dokončeno. Nyní lze již jednotlivé Properties ViewModelu napojovat (Binding) na jednotlivé prvky uživatelského rozhraní. [8]



Obr. 11 Ukázka propojení View a ViewModelu

Propojení jednotlivých prvků s Properties ViewModelu se provádí ve View. Položku uživatelského prvku, se kterou je potřeba manipulovat ve ViewModelu se propojí v argumentu elementu uživatelského prvku (např. Textbox a TextBlock mají argument Text, DataGrid a Combobox mají argument ItemSource pro zdrojovou kolekci prvků, které se mají zobrazit a argument SelectedItem, který má uloženou aktuálně zvolenou položku z kolekce) pomocí složených závorek a klíčového slova Binding za kterým následuje název Property. Na Obr. 11 lze vidět napojení Properties ViewModelu na jednotlivé uživatelské prvky uživatelského rozhraní. [8]

6.3.3 Způsob propojení

Na Obr. 11 lze vidět za klíčovým slovem Binding Jméno_Property také klíčové slovo Mode. Mode označuje způsob toku dat v propojení. K dispozici je pět možností výběru toku dat:

1. OneTime – data propojené s Property ViewModelu se aktualizují jen jedenkrát od spuštění aplikace nebo nastavení DataContextu daného View. Tento způsob propojení je vhodný pro statické texty, které není nutno po čas zobrazení View měnit.
2. OneWay – značí tok dat jedním směrem. Jedná se o směr z ViewModelu do View. Tzn. že pokud se data změní ve View, např. uživatel vloží nějaký text do textového pole, změna se neprojeví ve ViewModelu, ale pokud se data změní ve ViewModelu, změna se projeví ve View.
3. OneWayToSource – je také jednosměrný tok jako OneWay, jen opačným směrem, tedy změna dat může nastat jen ve View.
4. TwoWay – data můžou být měněna jak ve View, tak i ve ViewModelu.
5. Default (Výchozí) – použije se výchozí hodnota prvku, na který se používá propojení. Prvky, se kterými může manipulovat uživatel, např. TextBox, do kterého můžou uživatelé psát text, mají výchozí mód TwoWay a prvky, se kterými uživatel nemůže interagovat (TextBlock, který slouží jen pro zobrazení textu) mají výchozí mód OneWay. [9]

6.3.4 Příznak aktualizace dat

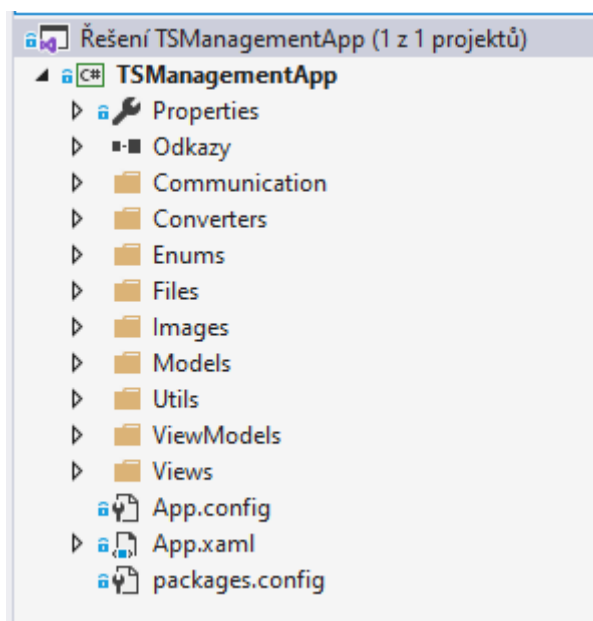
Na Obr. 11 lze vidět kromě klíčových slov Binding a Mode ještě klíčové slovo UpdateSourceTrigger. UpdateSourceTrigger určuje po jaké události se mají aktualizovat data v propojené Property. Existují čtyři možnosti nastavení, kdy se mají data aktualizovat:

1. PropertyChanged – aktualizace dat nastane ihned po změně vstupu, např. pokud se v textovém poli změní jen jeden znak, tak se ihned data aktualizují ve ViewModelu.
2. LostFocus – aktualizace dat nastane vždy, když editovatelný uživatelský prvek ztratí soustředění, tzn. uživatel se vyklikne z textového pole atp.
3. Explicit – aktualizuje data pouze po zavolání metody UpdateSource(). Tento způsob se dnes moc nepoužívá a je lepší jej nahradit např. LostFocus.

4. Default – použije se výchozí hodnota argumentu uživatelského prvku. Většina argumentů má výchozí mód PropertyChanged, ale prvky, kde se propojuje skrz argument Text, mají výchozí mód LostFocus [10]

7 IMPLEMENTACE APLIKACE

7.1 Struktura aplikace



Obr. 12 Struktura projektu

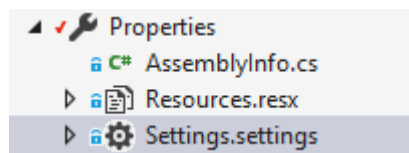
Na Obr. 12 lze vidět strukturu projektu. Vytvořeným adresářům se říká obory názvů a jejich cílem je shlukovat třídy, rozhraní, soubory s uživatelským rozhraním atd. pro zpřehlednění projektového adresáře. Pro jednoduché programy, které obsahují pár tříd, není nutné shlukovat jednotlivé třídy do různých oborů názvů. V případě rozsáhlejšího projektu, kdy by všechny soubory měly být v kořenovém adresáři projektu, bylo by velice obtížné pro autora i pro ostatní vývojáře se v projektu orientovat. Proto je dobré rozsáhlejší projekty rozdělovat do jednotlivých oborů názvů.

Projektový adresář projektu TSMangementApp (jméno projektu vytvářené aplikace) je rozdělen do těchto oborů názvů: Properties, Communication, Converters, Enums, Files, Images, Models, Utils, ViewModels, Views.

7.1.1 Properties

Tenhle obor názvů je vždy automaticky vygenerován při vytvoření libovolného projektu v jazyce C# a vývojovém prostředí Visual Studio. Obor názvů Properties si udržuje základní informace o vyvíjené aplikaci díky souboru AssemblyInfo, ve kterém se nacházejí důležité informace o projektu, jako jsou název vyvíjené aplikace, název společnosti, která vytváří aplikaci, aktuální verze aplikace a další. Dále je zde soubor Resources.resx, do kterého lze vkládat různé soubory, textové řetězce atd. Díky souboru Resources.resx lze všechny textové

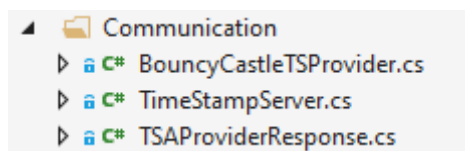
řetězce, které se nacházejí v aplikaci, sjednotit do jednoho souboru a poté lze tyto řetězce jednoduše volat kdekoliv v aplikaci pomocí názvu řetězce, který se zadá při vkládání do Resources.resx. Udržování textových zdrojů v jednom souboru usnadňuje práci např. při následné lokalizaci aplikace do cizího jazyka. Ještě se zde nachází soubor Settings.settings, do kterého lze ukládat nastavení aplikace a informace o aplikaci.



Obr. 13 Obsah oboru názvu Properties

7.1.2 Communication

Obor názvů Communication v projektu shlukuje třídy, jejichž cílová funkčnost je vytvořit žádost o časové razítko, za komunikovat s vybraným serverem časové autority, přijmout odpověď od serveru časové autority, tu rozbalit, z validovat ji a vrátit odpověď z časové autority do místa, odkud byly tyto funkce volány. V oboru názvů Communication je tedy naprogramováno znění normy RFC-3161.



Obr. 14 Obsah oboru názvu Communication

7.1.3 Converters

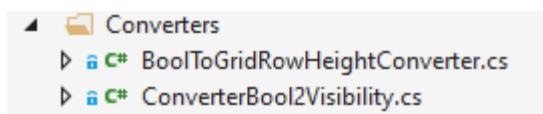
Converters je obor názvů, který uvnitř obsahuje tzv. konvertory. Konvertory jsou třídy, které mají velké využití hlavně při vytváření aplikací ve WPF. Konvertory umožňují převést klasickou proměnnou, která je dostupná v jazyku C#, do požadovaného typu např. z frameworku WPF. Tuhle klasickou proměnnou lze poté propojit s jakoukoliv vlastností uživatelského prvku, v propojení se zavolá požadovaný konvertor, který provede požadovanou konverzi.

```
<Grid Grid.Row="1" Visibility="{Binding ToolbarPanelIsVisible,  
    Converter={StaticResource ConverterBool2Visibility}}">
```

Obr. 15 Příklad použití konvertoru

Na obrázku lze vidět použití konvertoru, který je v projektu použit. Vlastnost Visibility, která je typu *Visibility*, je propojená s vlastností ve ViewModelu, která je typu *bool*. Takové

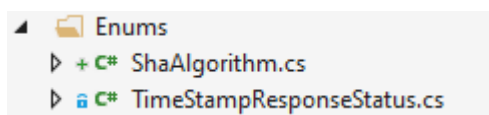
propojení by v době zobrazení View skončilo výjimkou, ale s použitím konvertoru se hodnota „pře konvertuje“ z typu bool na typ Visibility.



Obr. 16 Obsah oboru názvu Converters

7.1.4 Enums

Obor názvů Enums obsahuje výčtové typy, které se v aplikaci používají.



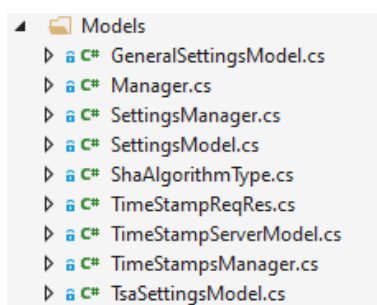
Obr. 17 Obsah oboru názvu Enums

7.1.5 Files a Images

Zde se nejedná přesně o obor názvů, ale spíše jen o adresář. V adresáři Files se nacházejí soubory, které se při sestavování aplikace přkopírují do adresáře, kde se aplikace sestaví (např. soubor s URL adresami časových autorit). Adresář Images obsahuje obrázky a ikony, které jsou v aplikaci použity.

7.1.6 Models

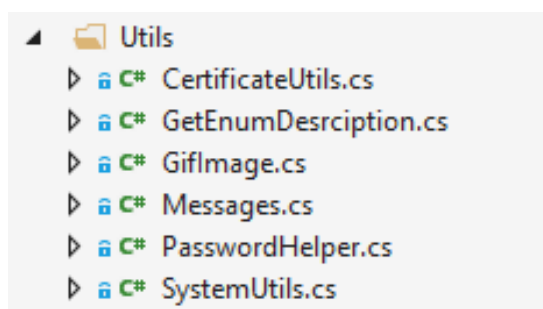
Obor názvů model zahrnuje dvě důležité části. První část jsou Modelové třídy, které jsou vytvářeny dle návrhového vzoru MVVM, část Models. Jsou to třídy, které jsou použity jako typ v kolekcích typu `ObservableCollection<T>`. Druhá část je způsob ukládání a zpřístupnění dat během běhu aplikace, ke kterým je potřeba často přistupovat ve všech částech programu. Tenhle problém je velmi efektivně vyřešen pomocí třídy Manager. Struktura třídy Manager je detailně rozebrána v kapitole 7.3.



Obr. 18 Obsah oboru názvu Models

7.1.7 Utils

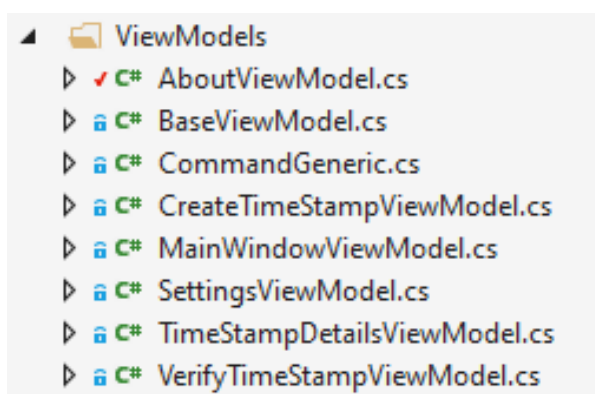
Obor názvu Utils obsahuje různé třídy, které „ulehčují“ a urychlují programování aplikace. Nachází se zde např. statická třída *Messages*, která obsahuje metody, které zobrazují různé systémové dialogy. Pro zobrazení dialogu je potřeba udělat pár kroků, které jsou vždy stejné. Proto je zbytečné psát tyhle kroky stále dokola, ale je lepší je dát do metody a poté jen volat danou metodu. Dále se zde nachází třída, která získává konkrétní informace z certifikátů, pomocná třída, která spustí animaci .gif obrázku, třída, která danému výčtovému typu vrátí slovní popis atd.



Obr. 19 Obsah oboru názvu Utils

7.1.8 ViewModels

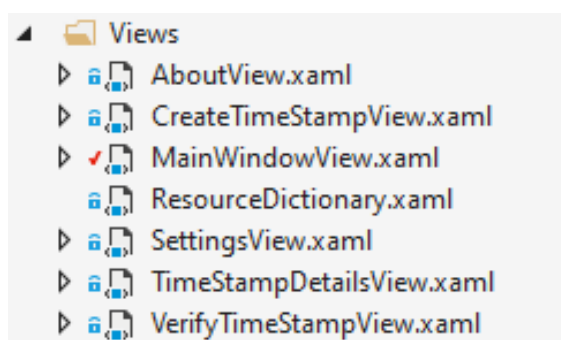
Obor názvu ViewModel obsahuje část návrhového vzoru MVVM, dle názvu oboru ViewModely. Nacházejí se zde ViewModely všech obrazovek, které se v aplikaci vyskytují, plus základní třída *BaseViewModel*, která implementuje rozhraní *INotifyPropertyChanged*, které musí ViewModely implementovat. Poté všechny ViewModely dědí od třídy *BaseViewModel*. Ještě je zde třída *CommandGeneric*, která implementuje rozhraní *ICommand* a umožňuje ve ViewModelech vytvářet instance rozhraní *ICommand*, které lze poté napojovat na tlačítka v uživatelském rozhraní.



Obr. 20 Obsah oboru názvu ViewModels

7.1.9 Views

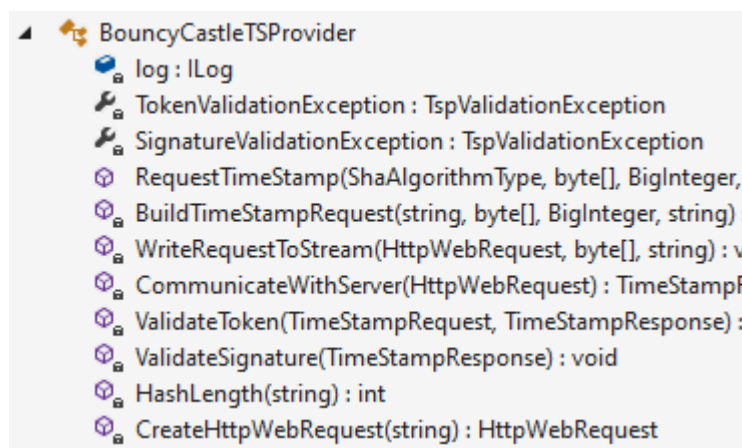
Obor názvů Views obsahuje poslední část návrhového vzoru MVVM, a to View. Nacházejí se zde soubory, ve kterých je vytvořeno uživatelské rozhraní jednotlivých obrazovek pomocí jazyka XAML a v kódu na pozadí každého .xaml souboru je vytvořeno propojení s příslušným ViewModelem, který se nachází v oboru názvu ViewModels. Ještě se zde nachází soubor ResourceDictionary, ve kterém jsou deklarovány barvy a styly uživatelských prvků a konvertory, které se nacházejí v oboru názvu Converters.



Obr. 21 Obsah oboru názvu Views

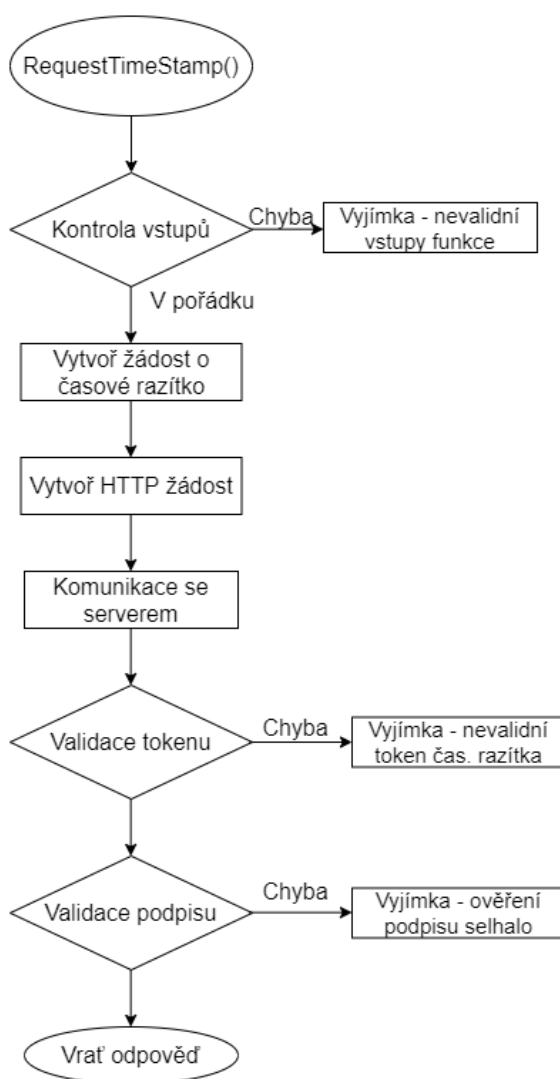
7.2 Zpracování normy RFC-3161 – tvorba časového razítka

Zpracování tvorby časového razítka dle normy RFC-3161 se nachází v oboru názvů Communication. Konkrétně se jedná o třídu *BouncyCastleTSPProvider*, která za pomoci NuGet balíčku Org.BouncyCastle vytvoří žádost o časové razítko, odešle žádost na server časové autority, vyčká na odpověď, kterou z validuje. Proces tvorby je ve třídě rozdělen do několika metod. Pro použití této třídy je potřeba vytvořit její instanci, viz kapitola 7.5.



Obr. 22 Struktura třídy BouncyCastleTSPProvider

Třída má jednu veřejnou metodu, `RequestTimeStamp`, která vykoná proces vytvoření časového razítka. Ve třídě je poté několik privátních metod, které slouží k zpřehlednění průběhu procesu v metodě `RequestTimeStamp`.



Obr. 23 Vývojový diagram funkce `RequestTimeStamp`

Privátní metody:

- `BuildTimeStampRequest` – metoda přijímá jako parametry identifikátor hashovacího algoritmu, otisk dokumentu, Nonce a politiku časové autority. Metoda vytvoří instanci třídy *TimeStampRequest*, a tu vrátí.
- `CreateHttpRequest` – vytvoří http požadavek, tedy instanci třídy *HttpRequest* a pokud je zvolena určitá metoda autentizace http požadavku (přihlašovací údaje, certifikát), vloží se do http požadavku.

- WriteRequestToStream – funkce vloží instanci třídy *TimeStampRequest* převedenou na pole *byte* do těla http požadavku.
- CommunicateWithServer – odešle http požadavek a počká na přijetí odpovědi;
- ValidateToken – funkce provede validaci tokenu časového razítka;
- ValidateSignature – zkontroluje digitální podpis časového razítka (zda je podepsán platným certifikátem časové autority, který slouží pro vytváření časových razítek);
- HashLength – vrací délku otisku použitého hashovacího algoritmu.

7.3 Přístup ke sdíleným datům v programu

Aplikace nabízí různé možnosti nastavení, co se týká způsobu připojení k internetu, nastavení pro službu tvorby časových razítek a možnostech autentizace http požadavku (více informací o možnostech nastavení se nachází v kapitole 7.9). Také je potřeba udržovat po celou dobu běhu aplikace seznam vydaných časových razítek.

K možnostem nastavení a k historii vydaných časových razítek musí jít přistoupit odkudkoli z programu, protože např. při vytváření časového razítka se musí použít zvolená volba autentizace http požadavku, musí se odeslat na nastavený server atd.

Existuje vícero možností, jak si požadovaná data uchovávat, např. ukládání do souborů (např. XML soubor) a v obrazovce, kde jsou potřeba by se ze souboru načetly, nebo hodnoty uchovávat v *MainWindowViewModel* a poté je předávat přes konstruktory jednotlivých tříd. Obě zmíněné metody lze použít, ale moc efektivní by nebyla a jejich implementace by požadovala více napsaného kódu.

V aplikaci pro správu časových razítek si aktuální hodnoty nastavení a seznam vydaných razítek udržuje třída *Manager*. Třída *Manager* obsahuje dvě *Properties*. První je typu *SettingsManager*, která ukládá veškeré nastavení aplikace. Druhá je typu *TimeStampsManager*, která obsahuje historii vydaných časových razítek. Dále obsahuje statickou *Property* sebe sama, tedy třídy *Manager*. Tahle *Property* je vytvořena podle návrhového vzoru *Singleton*. Pokud je nějaká třída nebo *Property* vytvořena dle návrhového vzoru *Singleton*, znamená to, že za celou dobu běhu programu existuje pouze jedna jediná instance této třídy. Poté když je potřeba přistoupit k datům, která jsou ve třídě *Manager* nevytváří se instance této třídy, ale přistupuje se k ní pomocí názvu třídy a *Singleton* *Property*, která se v aplikaci nazývá *Instance* (viz. Obr. 23).

```
private static volatile Manager _instance;
Počet odkazů: 27
public static Manager Instance
{
    get
    {
        if(_instance == null)
        {
            lock(_syncRoot)
            {
                _instance = new Manager();
            }
        }
        return _instance;
    }
}
```

Obr. 23 Ukázka vytvoření Singleton Property

```
SettingsModel settings = Manager.Instance.SettingsManager.Settings;
```

Obr. 24 Ukázka získání dat z nastavení pomocí Managera

Třída *Manager* umožňuje efektivní přístup k datům odkudkoli v programu, ale ještě je nutné, aby data byla dostupná při každém spuštění aplikace a uživatel si nemusel při každém spuštění volit nastavení a historie vydaných razítek by sahala do doby jednoho spuštění aplikace. Tomu lze zabránit pomocí serializace zvoleného nastavení a historie vydaných razítek na pevný disk.

O serializaci dat se také stará třída *Manager*. Serializace se realizuje pomocí ukládání dat do XML souborů. Aplikace vytváří dva serializační XML soubory. První obsahuje historii vydaných časových razítek a druhý obsahuje zvolené nastavení aplikace.

Vždy, když se vydá nové časové razítko, přidá se do historie vydaných razítek, a nakonec se zavolá metoda *SerializeTimeStamps*, která se nachází v *TimeStampsManager*, který je součástí třídy *Manager*, a ta vytvoří XML soubor, do kterého se celá historie uloží. Do XML souboru je potřeba uložit všechna data o časovém razítku ve formátu, aby bylo možno při dalším spuštění aplikace data ze souboru načíst, a korektně se zobrazil detail vydaného časového razítka. Z toho důvodu má XML soubor na serializaci vydaných časových razítek pevně danou strukturu, viz. Obr. 24.


```
<Timestamps>
  <Timestamp>
    <!-- Více násobná položka -->
    <SourceFilePath></SourceFilePath>
    <GenTime></GenTime>
    <TimeStampFilePath></TimeStampFilePath>
    <TimeStampFileName></TimeStampFileName>
    <TimeStampFileType></TimeStampFileType>
    <TimeStampRequest></TimeStampRequest>
    <TimeStampResponse></TimeStampResponse>
    <TsType></TsType>
  </Timestamp>
</Timestamps>
```

Obr. 24 Struktura serializačního souboru pro historii vydaných časových razítek

Element `Timestamps` je kořenový element XML souboru a v něm se může nacházet libovolný počet elementů `Timestamp`, který musí obsahovat tyto elementy:

- `SourceFilePath` – obsahuje cestu k souboru, pro které bylo vytvořeno časové razítko.
- `GenTime` – obsahuje čas vytvoření časového razítka.
- `TimeStampFilePath` – obsahuje cestu k časovému razítku.
- `TimeStampFileType` – časové razítko lze uložit dvěma způsoby. Lze uložit celá odpověď ze serveru časové autority (*TimeStampResponse*, přípona *.tsr*), nebo jen token časového razítka (*TimeStampToken*, *.tst*). Typ uložení je kritický pro správné načtení vydaného razítka z historie.
- `TimeStampRequest` – obsahuje žádost o časové razítko ve formátu Base64 string.
- `TimeStampResponse` – obsahuje odpověď časového razítka, nebo token časového razítka ve formátu Base64 string.

Podobně probíhá i serializace nastavení. Vždy, když uživatel klikne v obrazovce nastavení na tlačítko „Uložit“ (viz. kapitola 7.9), nové hodnoty nastavení se uloží do `Properties` třídy *SettingsManager* a následně se serializují do XML souboru. Struktura XML serializačního souboru nastavení lze vidět na Obr. 25.

```
<TSManagementAppSettings>
  <GeneralSettings>
    <IsSettingsIsChecked></IsSettingsIsChecked>
    <ProxyIsChecked></ProxyIsChecked>
    <ProxyServer></ProxyServer>
    <UserName></UserName>
    <Password />
  </GeneralSettings>
  <TsaSettings>
    <NoAuthIsChecked></NoAuthIsChecked>
    <TsaAuthCertIsChecked></TsaAuthCertIsChecked>
    <TsaAuthIpIsChecked></TsaAuthIpIsChecked>
    <TsaAuthLoginDataIsChecked></TsaAuthLoginDataIsChecked>
    <TsaAuthCert />
    <TsaLoginName></TsaLoginName>
    <TsaLoginPassword />
    <TsaPolitics></TsaPolitics>
    <TsaMessageImprintType></TsaMessageImprintType>
    <TsaSelectedTimeStampServer></TsaSelectedTimeStampServer>
  </TsaSettings>
</TSManagementAppSettings>
```

Obr. 25 Struktura serializačního souboru nastavení

Kořenový element serializačního XML souboru nastavení je TSManagementAppSettings. Kořenový element má vnořené dva elementy, kdy první element, GeneralSettings, obsahuje obecné nastavení a druhý element, TsaSettings, obsahuje nastavení, které se týká vytvoření časového razítka.

Element GeneralSettings obsahuje následující elementy:

- IsSettingsIsChecked – možnost nastavení připojení k internetu pomocí výchozího nastavení systému Windows. Obsahuje hodnoty True nebo False.
- ProxyIsChecked – možnost připojení pomocí Proxy. Obsahuje hodnoty True nebo False.
- ProxyServer – obsahuje adresu Proxy serveru ve formátu Proxy_server:Port.
- UserName – Přihlašovací jméno k Proxy serveru.
- Password – Heslo k Proxy serveru.

Element TsaSettings obsahuje následující elementy:

- NoAuthIsChecked – udává, zda je zvolena možnost odeslání http požadavku bez autentizace. Může obsahovat hodnoty True nebo False.

- TsaAuthCertIsChecked – udává, zda se má použít autentizace pomocí certifikátu. Může obsahovat hodnoty True nebo False.
- TsaLoginDataIsChecked – udává, zda se má použít autentizace pomocí přihlašovacích údajů. Může obsahovat hodnoty True nebo False.
- TsaAuthCert – obsahuje RawData certifikátu, které se konvertují na Base64 string.
- TsaLoginName – obsahuje přihlašovací jméno pro autentizaci pomocí přihlašovacích údajů.
- TsaLoginPassword – obsahuje heslo pro autentizaci pomocí přihlašovacích údajů. Heslo je konvertováno do Base64 string.
- TsaPolitics – obsahuje hodnotu požadované politiky časové autority.
- TsaSelectedTimeStampServer – obsahuje zvolenou adresu serveru časové autority.
- TsaMessageImprintType – obsahuje zvolený hash algoritmus pro vytvoření otisku souboru, pro který se má vytvořit časové razítko.

Vytvoření XML souboru a kořenového elementu

```
XmlDocument xml = new XmlDocument();  
xml.AppendChild(xml.CreateXmlDeclaration("1.0", "utf-8", null));  
  
XmlNode timeStamps = xml.CreateElement("TimeStamps");
```

Přidávání elementů do kořenového elementu TimeStamps

```
node = xml.CreateElement("SourceFilePath");  
node.InnerText = a.SourceFilePath;  
timeStamp.AppendChild(node);  
node = xml.CreateElement("GenTime");  
node.InnerText = a.GenTime;  
timeStamp.AppendChild(node);  
  
...  
node = xml.CreateElement("TimeStampResponse");  
if (a.TimeStampResponse != null)  
    node.InnerText = Convert.ToBase64String(a.TimeStampResponse);  
timeStamp.AppendChild(node);  
  
timeStamps.AppendChild(timeStamp);
```

Přidání elementu TimeStamps do XML dokumentu a uložení XML souboru

```
xml.AppendChild(timeStamps);  
xml.Save($"{SystemUtils.UserLocalAppDirectory}\\TimeStampsHistory.xml");
```

Obr. 26 Ukázka vytváření XML souboru v C# .NetFramework

Deserializace, neboli načtení serializovaných dat, se provádí v momentě inicializace Property Instance ve třídě *Manager*. Deserializace proběhne pouze v případě, že jsou nalezeny serializační XML soubory.

Načtení XML souboru

```
XmlDocument xml = new XmlDocument();
xml.Load($"{SystemUtils.UserLocalAppDirectory}\\TimeStampsHistory.xml");
```

Získání kořenového elementu

```
XmlNode timeStamps = xml.SelectSingleNode("/TimeStamps");
```

Procházení položek v kořenovém elementu TimeStamps a jeho vnořených elementů

```
foreach (XmlNode timeStamp in timeStamps.ChildNodes)
{
    TimeStampReqRes stamp = new TimeStampReqRes();
    foreach (XmlNode node in timeStamp.ChildNodes)
    {
        switch (node.Name)
        {
            case "SourceFilePath":
                stamp.SourceFilePath = node.InnerText; break;
            case "GenTime":
                stamp.GenTime = node.InnerText; break;
            case "TimeStampFilePath":
                stamp.TimeStampFilePath = node.InnerText; break;
            case "TimeStampFileName":
                stamp.TimeStampFileName = node.InnerText; break;
            case "TimeStampFileType":
                stamp.TimeStampFileType = node.InnerText; break;
        }
    }
}
```

Obr. 27 Ukázka deserializace XML dokumentu

7.4 Hlavní okno aplikace

Hlavní okno aplikace je vytvořeno ze souboru *MainWindowView.xaml*, které tvoří View hlavního okna a poté ViewModel *MainWindowViewModel.cs*

Hlavní okno aplikace je rozděleno do tří hlavních částí:

1. První část se nachází na horní straně hlavního okna a je zde dostupné hlavní menu aplikace. Z hlavního menu aplikace si uživatel může vybrat požadovanou akci (např. vytvořit časové razítko, ověřit časové razítko, nastavení atd.) K hlavním funkcím aplikace lze přistoupit dvěma způsoby. První způsob je lišta záložek, která je typická pro aplikace ve Windows. Lišta záložek lze vytvořit ve WPF pomocí značky *Menu*, do kterého se vnořuje značka *MenuItem*.

```

<Menu Grid.Row="0" VerticalAlignment="Center">
    <MenuItem Header="Razítko" >
        <MenuItem Header="Vytvořit" Command="{Binding CommandCreateNewTimestamp}"
            Tooltip="Vytvořit nové časové razítko">
        </MenuItem>
    </MenuItem>
</Menu>

```

Obr. 28 Ukázka tvorby lišty záložek

Druhý způsob volby jednotlivých funkcí aplikace je panel nástrojů (anglicky Toolbar), který lze více stylovat. Panel nástrojů lze vytvořit pomocí značky Toolbar, do kterého lze vnořovat libovolné uživatelské prvky. Nejčastěji se zde vkládá prvek Button, který má nějaký speciální styl. Ve vytvářené aplikaci se tlačítko skládá z obrázku a stručného názvu.

```

<ToolBarTray Background="Transparent" VerticalAlignment="Stretch">
    <ToolBar Loaded="ToolBar_Loaded" ToolBarTray.IsLocked="True" Margin="0,5">
        <Button Tooltip="Vytvořit časové razítko" Command="{Binding CommandCreateNewTimestamp}">
            <StackPanel>
                <Image Source="../../Images/TimeStampCreate.png" Height="30" Width="50"/>
                <TextBlock Text="Vytvořit" TextAlignment="Center"/>
            </StackPanel>
        </Button>
    </ToolBar>
</ToolBarTray>

```

Obr. 29 Ukázka tvorby panelu nástrojů

2. V druhé části hlavního okna se nachází historie časových razítek. Historie časových razítek obsahuje seznam vydaných razítek. V historii lze vybírat jednotlivé položky a po zvolení položky se vždy zobrazí detail zvoleného časového razítka. Seznam historie časových razítek je vytvořen pomocí tabulky. Tabulka se ve WPF tvoří pomocí značky DataGrid. Pro korektní zobrazení dat v tabulce je potřeba ještě nadefinovat sloupcečky tabulky. Příklad definování sloupců tabulky lze vidět na Obr. 30.

```

<DataGrid ItemsSource="{Binding Users}">
    <DataGrid.Columns>
        <DataGridTextColumn Binding="{Binding FirstName, Mode=OneWay}"
            Header="Křestní jméno" Width="*"/>
        <DataGridTextColumn Binding="{Binding LastName, Mode=OneWay}"
            Header="Příjmení" Width="*"/>
    </DataGrid.Columns>
</DataGrid>

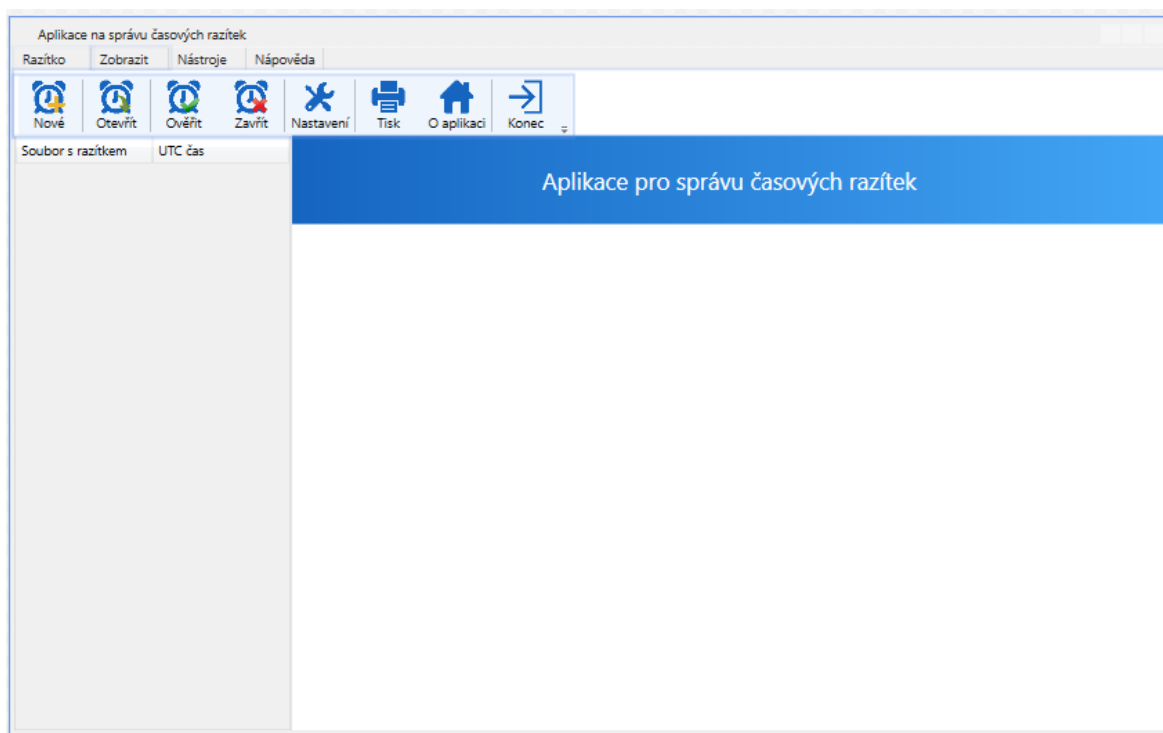
```

Obr. 30 Ukázka definice sloupců v DataGrid

3. Poslední a největší část hlavního okna zabírá prostor pro zobrazení dílčích obrazovek aplikace. Pokud si např. uživatel zvolí, že chce přejít do nastavení, právě na této ploše se dílčí obrazovka zobrazí. Celou plochu vyplní prvek UserControl, kdy se vlastnost Content propojí s Property ve ViewModelu, která je typu UserControl.

```
<UserControl HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
    Grid.Row="1" Content="{Binding CurrentUserControl}" Margin="5,0"/>
```

Obr. 31 Ukázka použití UserControl v aplikaci



Obr. 32 Hlavní okno aplikace

ViewModel hlavního okna (MainWindowViewModel.cs) je první ViewModel, který se po spuštění aplikace vytváří. Proto se ve ViewModelu musí načíst globální nastavení aplikace, a historie vydaných razítek, viz kapitola 7.3. K historií časových razítek se přistupuje z hlavního okna a nastavení aplikace se používá ve více částech aplikace, a kdyby nebylo načteno hned při startu aplikace, mohlo by způsobit špatný chod nebo v horším případě i pád aplikace. Dále ViewModel hlavního okna obsahuje obsluhy kliknutí tlačítek, které zobrazují dílčí obrazovky aplikace (nastavení, vytvoření časového razítka atd.)

Z hlavní obrazovky aplikace lze přistoupit k následujícím funkcím:

1. Nové časové razítko – zahájí proces vytváření nového časového razítka.
2. Otevřít časové razítko – umožní přidat do historie vydané časové razítko, které je uloženo na disku zařízení.

3. Ověřit časové razítko – zahájí proces ověření časového razítka.
4. Odstranit časové razítko z historie – odstraní zvolené časové razítko z historie vydaných časových razítek.
5. Nastavení – zobrazí obrazovku nastavení, kde lze nastavovat atributy připojení k síti, výběr serveru časové autority atd.
6. Tisk – funkce která umožní vytisknout právě zvolenou obrazovku. Např uživatel potřebuje vytisknout informace, které jsou obsaženy v detailu časového razítka. Uživatel tedy stiskne tlačítko Tisk, které mu umožní vytisknout aktuální obrazovku, nebo jí uložit do .PDF souboru.
7. Informace o aplikaci – zobrazí základní informace o aplikaci. Tedy název, základní informace a verzi aplikace.
8. Ukončit aplikaci – možnost ukončení aplikace, ukončení proběhne totožně jako kdyby uživatel klikl na „křížek“ v pravém horním rohu aplikace.

7.5 Vytvoření časového razítka

Proces zahájení tvorby časového razítka lze spustit pomocí tlačítka „Nové“ umístěného v hlavním panelu nebo po stisku tlačítka Razítko a Nové v menu aplikace. Po zahájení tvorby procesu se uživateli zobrazí systémový dialog pro vybrání souboru, ke kterému se má vytvořit časové razítko.

Systémový dialog pro volbu souboru lze v C# .NET Frameworku jednoduše pomocí třídy OpenFileDialog, která se nachází v oboru názvů System.Windows.Forms.

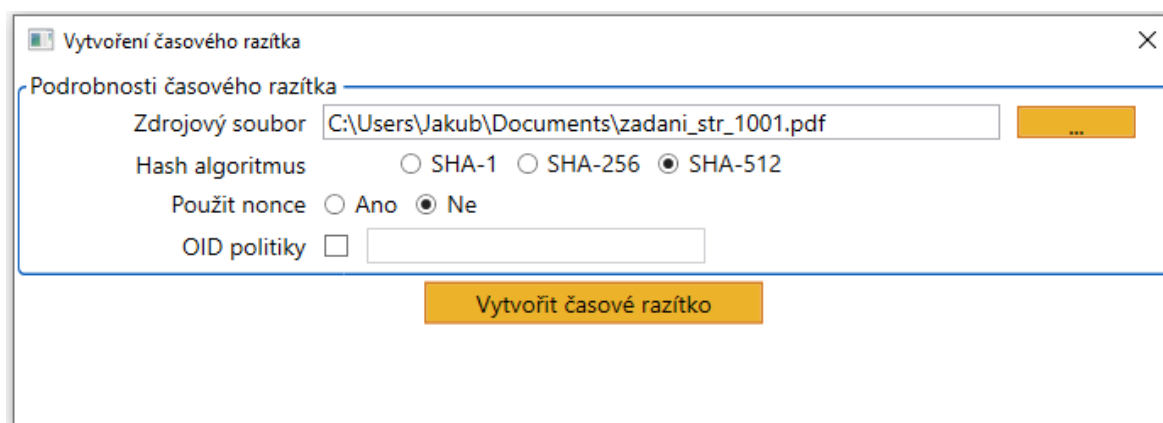
```
using (OpenFileDialog dialog = new OpenFileDialog())
{
    dialog.Title = title;
    dialog.Filter = filter;
    dialog.Multiselect = false;
    if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        return dialog.FileName;
    else
        return null;
}
```

Obr. 33 Ukázka vytvoření dialogu pro získání cesty souboru

Na Obr.33 lze vidět ukázka kódu, která uživateli zobrazí dialog pro výběr souboru. Před zobrazením dialogu pro výběr souboru lze nastavit několik Properties, které nastavují chování a vzhled zobrazeného dialogu:

- Title – nastaví titulek okna dialogu.
- Filter – umožní nastavit filtr preferovaných souborů, které se mají zobrazovat v dialogu. Nastavení filtru má následující syntaxi: „požadovaná přípona|Název filtru“. Např. „.docx|Soubor Microsoft Word“. Pokud je potřeba uvést vícero filtrů, uvedená syntaxe se oddělí středníkem a lze uvést další filtry.
- Multiselect – povolí, nebo zakáže výběr více souborů najednou. Aplikace umožňuje vytvořit v jeden moment jen jedno časové razítko, proto má volba MultiSelect hodnotu False.

Poté co si uživatel zvolí soubor, zobrazí se mu nové dialogové okno, kde si lze vybrat doplňující položky ohledně vytvoření časového razítka. V dialogu je zobrazena cesta ke zvolenému souboru, volba Hash algoritmu pro vytvoření otisku souboru, volba, zda do žádosti o časové razítko zakomponovat Nonce a možnost vložit požadovanou hodnotu politiky časové autority. Všechny zmíněné doplňující možnosti je možné přednastavit v obrazovce nastavení, a před zobrazením dialogového okna pro vytvoření časového razítka se tyto možnosti vyplní přednastavenými hodnotami.



Obr. 34 Dialogové okno pro vytvoření časového razítka

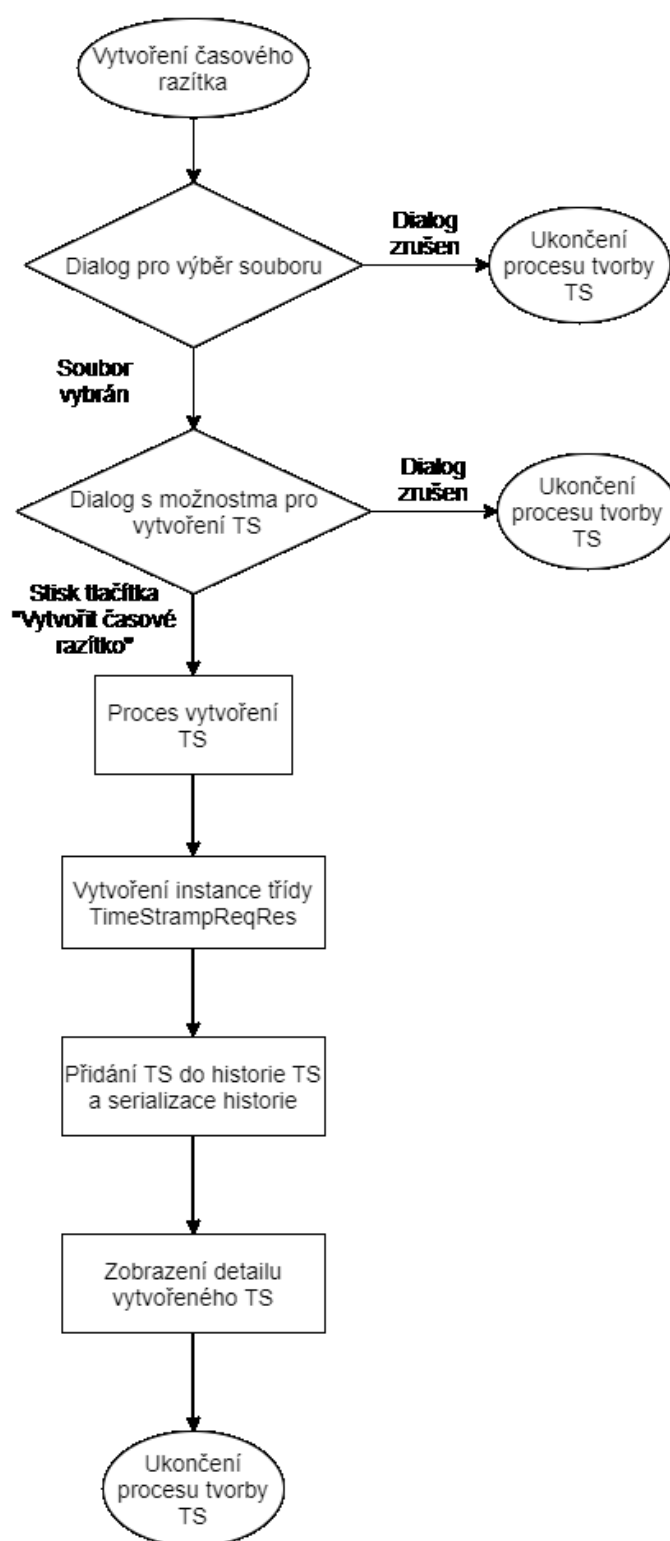
Po stisku tlačítka „Vytvořit časové razítko“ se zahájí proces vytvoření časového razítka. Nejprve se vytvoří instance třídy *BouncyCastleTSPProvider* (viz. kapitola 7.2), a poté se zavolá metoda *RequestTimeStamp* pomocí instance třídy *BouncyCastleTSPProvider*. Metoda *RequestTimeStamp* implementuje celý proces vytvoření časového razítka a vrátí odpověď

ze serveru časové autority. Odpověď z časového razítka je uložena v instanci třídy, která se nazývá *TimeStampReqRes* a obsahuje následující položky:

- *TimeStampRequest* – žádost o časové razítko převedené na pole Bytů.
- *TimeStampResponse* – odpověď navracená ze serveru časové autority.
- *SourceFilePath* – soubor, ke kterému bylo časové razítko vytvářeno.
- *GenTime* – čas vytvoření časového razítka. Property má hodnotu pouze v případě, pokud bylo časové razítko vydáno.
- *TimeStampFilePath* – cesta, kde je uloženo vytvořené časové razítko.
- *TimeStampFileName* – jméno souboru vytvořeného časového razítka.
- *TimeStampFileType* – typ, jak je časové razítko uloženo. K dispozici jsou dvě možnosti. Lze uložit celá odpověď časového razítka, nebo jen token časového razítka.

Třída *TimeStampReqRes* obsahuje totožné položky jako serializační soubor pro časová razítka. Je to z důvodu, že třída *TimeStampReqRes* je použita jako zdrojový typ pro kolekci, která obsahuje historii vydaných razítek, tudíž je všechny Properties potřeba uložit do serializačního souboru, aby se při dalším spuštění aplikace načetly (viz kapitola 7.3).

Po vložení jednotlivých hodnot do instance třídy *TimeStampReqRes* se instance této třídy přidá do historie vydaných razítek a ta se následně serializuje. Po dokončení procesu tvorby časového razítka se zobrazí detail právě vydaného časového razítka, viz kapitola 7.8.



Obr. 35 Proces vytvoření časového razítka v aplikaci

7.6 Ověření a otevření existujícího časového razítka

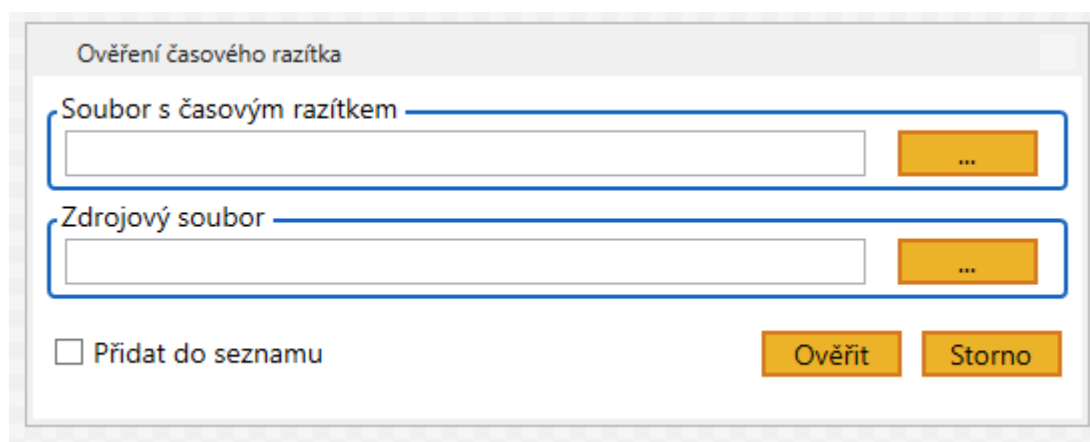
Procesy ověření a otevření existujícího časového razítka mají stejný základ, používají stejné dialogové okno, ale proces otevření existujícího časového razítka je rozšířen o doplňující

funkci, která umožňuje ověřené časového razítka uložit do historie vydaných časových razítek a po ověření se zobrazí detail ověřovaného časového razítka.

7.6.1 Ověření časového razítka

Proces ověření existujícího časového razítka lze spustit pomocí tlačítka „Ověřit“, který je umístěn na panelu nástrojů, který se nachází v hlavním okně aplikace, nebo pomocí tlačítka Razítka a poté Ověřit umístěné z hlavního menu aplikace.

Po stisku tlačítka Ověřit se uživateli zobrazí nové dialogové okno:



Obr. 36 Dialogové okno funkce Ověření časového razítka

Obrazovka obsahuje pole pro výběr souboru časového razítka. Umístění časového razítka lze vepsat manuálně do textového pole, nebo lze použít systémový dialog pro vybrání existujícího souboru, který je podrobněji popsán v kapitole 7.5 s upraveným filtrem, který zobrazuje soubory s příponou .tsr (časové razítka ve formátu TimeStampResponse), soubory s příponou .tst (časové razítka ve formátu TimeStampToken) a možnost pro zobrazení souborů se všemi příponami pro případ, že by uživatel použil svou vlastní příponu při uložení časového razítka (možno udělat v obrazovce Detail časového razítka, viz kapitola 7.8).

Když uživatel použije systémový dialog pro výběr souboru časového razítka, aplikace se pokusí najít zdrojový soubor, ke kterému bylo časové razítka vytvořeno. Zdrojový soubor se aplikaci povede najít jedině v případě, pokud se časové razítka jmenuje stejně jako zdrojový soubor a nachází se ve stejném adresáři. Zdrojový kód funkce, který se pokouší najít zdrojový soubor časového razítka, lze vidět na obr. 37.

```
private string FindSourceFile()
{
    FileInfo tsFileInfo = new FileInfo(TimeStampFilePath);
    string[] files = Directory.GetFiles(tsFileInfo.DirectoryName);
    foreach (string file in files)
    {
        FileInfo info = new FileInfo(file);
        if (info.Name == tsFileInfo.Name.Replace(tsFileInfo.Extension, null) && info.Extension != ".tsr")
            return info.FullName;
    }
    return null;
}
```

Obr. 37 Funkce pro nalezení zdrojového souboru

Pokud funkce nedokáže najít zdrojový soubor, musí jej uživatel vybrat sám. Cestu k souboru může opět vložit ručně do textového pole, nebo využít systémový dialog Windows pro výběr souboru.

Po výběru souboru s časovým razítkem a zdrojového souboru, ke kterému bylo časové razítko vytvářeno, lze přejít k ověření časového razítka. Proces ověření se spustí po stisknutí tlačítka Ověřit.

Po spuštění procesu ověření je nejprve potřeba zjistit v jakém formátu je soubor uložen. První indikátor, zda je soubor časového razítka uložen ve formátu *TimeStampResponse*, nebo *TimeStampToken*, je přípona souboru. O příponu souboru jako přesný indikátor typu souboru se opřít nelze, protože příponu každého souboru lze v systému Windows jednoduše změnit. Proto probíhá zjištění formátu časového razítka programově. Jednoduše se ze vstupního souboru zkusí vytvořit instance třídy *TimeStampToken* nebo *TimeStampResponse* a podle toho, která instance se povede vytvořit, se určí formát, v jakém je soubor časového razítka uložen. Pokud se nepovede vytvořit instance ani jedné ze dvou výše zmíněných tříd, proces ověření je ukončen, protože nebyl zvolen soubor časového razítka, který by odpovídal normě RFC-3161.

```
private bool IsTimeStampResponse(byte[] data)
{
    try
    {
        TimeStampResponse response = new TimeStampResponse(data);
        return true;
    }
    catch
    {
        return false;
    }
}
```

Obr. 38 –Metoda zjišťující, zda je ověřovaný soubor ve formátu *TimeStampResponse*

Zjištění formátu ověřovaného souboru časového razítka je důležité pro proces ověření. Pokud je ověřovaný soubor časového razítka ve formátu `TimeStampResponse`, lze ověřit i digitální podpis časového razítka a status uložený v `TimeStampResponse`, pokud je ve formátu `TimeStampToken` ověření podpisu a statusu se přeskočí a proces ověření začíná až v kroku č. 3.

Proces ověření časového razítka se skládá z těchto kroků:

1. Ověření statusu `TimeStampResponse` – Status musí mít hodnotu 0 (Granted), nebo 1 (GrantedWithMods).
2. Ověření digitálního podpisu časového razítka – pokud se podpis nepodaří ověřit, časové razítko není validní.
3. Ověření délky otisku orazítkovaného dokumentu – délka otisku musí souhlasit s délkou otisku použitého hashovacího algoritmu, který je uveden v časovém razítku, že byl použit pro vytvoření otisku.
4. Ověření otisku obsaženého v časovém razítku s otiskem zdrojového souboru – při ověření je potřeba znovu vypočítat otisk zdrojového souboru pomocí stejného hashovacího algoritmu, který je uložen v časovém razítku a vypočtený otisk porovnat s otiskem obsaženým v časovém razítku. Pokud se otisky neshodují, znamená to, že se zdrojovým souborem bylo manipulováno od doby vzniku časového razítka.

Pokud jednotlivé kroky ověření proběhnou bez chyby, je časové razítko v pořádku a uživateli je oznámeno, že ověření proběhlo úspěšně. Pokud se naskytnou při validaci chyby, zobrazí se uživateli chybová hláška s textovým popisem chyb.

```
private string VerifyTSResponse(TimeStampResponse tsResponse)
{
    StringBuilder errMsg = new StringBuilder();
    if (tsResponse.Status != 0 && tsResponse.Status != 1)
        errMsg.AppendLine($"{tsResponse.Status} - {tsResponse.GetStatusString()}");
    try { BouncyCastleTSProvider.ValidateSignature(tsResponse); }
    catch (Exception e) { errMsg.AppendLine(Resources.VerifyTsVM_InvalidSignature); }
    if (tsResponse.TimeStampToken != null)
        errMsg.AppendLine(VerifyTSToken(tsResponse.TimeStampToken));

    return errMsg.ToString();
}
```

Obr. 39 Proces ověření `TimeStampResponse`

```

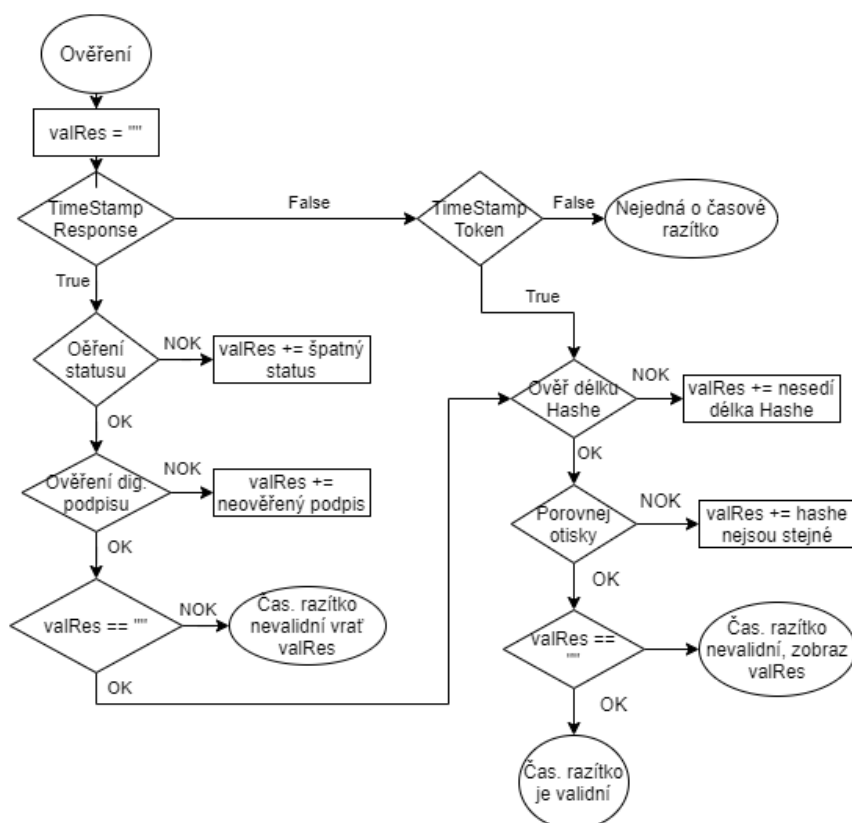
private string VerifyTSToken(TimeStampToken tsToken)
{
    StringBuilder errMsg = new StringBuilder();
    string hashOid = tsToken.TimeStampInfo.MessageImprintAlgOid;
    HashAlgorithm sourceFileImprint = null;
    FileStream sourceFS = new FileStream(SourceFilePath, FileMode.Open);
    switch (hashOid)
    {
        case "1.3.14.3.2.26":
            sourceFileImprint = SHA1.Create();
            sourceFileImprint.ComputeHash(sourceFS);
            break;
        case "2.16.840.1.101.3.4.2.1":
            sourceFileImprint = SHA256.Create();
            sourceFileImprint.ComputeHash(sourceFS);
            break;
        case "2.16.840.1.101.3.4.2.3":
            sourceFileImprint = SHA512.Create();
            sourceFileImprint.ComputeHash(sourceFS);
            break;
        default:
            errMsg.AppendLine(Resources.TsDetailsVM_UnsupportedHash); break;
    }

    sourceFS.Dispose();

    if (sourceFileImprint.Hash.Length != tsToken.TimeStampInfo.GetMessageImprintDigest().Length)
        errMsg.AppendLine(Resources.TsDetailsVM_HashLengthNotEqual);
    else if (!CompareTwoHashImprints(sourceFileImprint.Hash, tsToken.TimeStampInfo.GetMessageImprintDigest()))
        errMsg.AppendLine(Resources.TsDetailsVM_MessageImprintNonEqual);
    return errMsg.ToString();
}

```

Obr. 40 Proces ověření TimeStampToken



Obr. 41 Vývojový diagram procesu ověření časového razítka

7.6.2 Otevření časového razítka

Proces otevření časového razítka lze spustit pomocí tlačítek „Otevřít“. Jedno tlačítko Otevřít je k dispozici v panelu nástrojů na umístěném na hlavním okně aplikace a druhé tlačítko Otevřít se nachází v hlavním menu aplikace.

Po spuštění procesu otevření časového razítka se uživateli zobrazí systémový dialog pro výběr souborů s filtrem definovaným pro soubory s časovými razítky, tedy soubory s příponou .tsr nebo .tst a filtr pro zobrazení všech souborů.

Po dokončení výběru souboru se uživateli zobrazí stejné dialogové okno jako při ověření časového razítka. Jen je v dialogovém okně zvolena volba „Přidat do seznamu“. Tuhle volbu lze vybrat i při procesu ověření.

Po stisknutí tlačítka „Ověřit“ proběhne proces ověření časového razítka, jako je popsáno v kapitole 7.6.1 v bodech 1-5. V procesu Otevření se ale po dokončení ověření, přičemž nezáleží na výsledku ověření, ověřené časové razítko vloží do historie vydaných razítek a po dokončení ověření je zobrazen detail otevíraného časového razítka (viz kapitola 7.8).

7.7 Manipulace s historií časových razítek

Do historie vydaných časových razítek se přidávají časová razítka automaticky po vytvoření časového razítka (viz kapitola 7.5). Dále lze přidat časové razítko i pomocí funkce Otevřít časové razítko (viz kapitola 7.6.2). Vydaná časová razítka lze z historie i odstranit. Proces odstranění se skládá ze dvou kroků. Nejprve je potřeba zvolit časové razítko, které se má odstranit z historie vydaných razítek. Po zvolení časového razítka z historie vydaných razítek je potřeba kliknout na tlačítko „Zavřít“, které se nachází v panelu nástrojů nebo v hlavním menu aplikace pomocí kombinace Razítko a poté „Zavřít“. Po stisku tlačítka „Zavřít“ se zvolené časové razítko vyhledá v historii časových razítek, odstraní se, a nakonec se historie serializuje. Časové razítko se pouze smaže z historie, soubor časového razítka je v počítači nadále uložen.

```
public void DeleteOneTimeStamp(TimeStampReqRes timeStamp)
{
    TimeStamps.Remove(timeStamp);
    log.Info($"Time stamp {timeStamp.TimeStampFilePath} was deleted from history list");
    SerializeTimeStamps();
}
```

Obr. 42 Funkce pro odstranění časového razítka z historie vydaných časových razítek

7.8 Detail časového razítka

Obrazovka detailu časového razítka se uživateli zobrazí po zvolení časového razítka z historie vydaných razítek nebo po dokončení procesu vytvoření nebo otevření časového razítka. V detailu časového razítka je možné si prohlédnout informace o časovém razítku a poté provést dodatečné akce s časovým razítkem (např. dodatečné uložení časového razítka).

7.8.1 Informace o časovém razítku

Informace zobrazené v obrazovce „Detail časového razítka“ se dělí do dvou kategorií, informace obecné a podrobnosti časového razítka.

Obecné informace obsahují tyto položky:

- Umístění časového razítka – obsahuje cestu, kde je v počítači uložen soubor s časovým razítkem.
- Umístění zdrojového souboru – obsahuje cestu, kde je v počítači uložen soubor, ke kterému bylo časové razítko vytvořeno.
- Výsledek ověření – obsahuje výsledek ověření časového razítka, které proběhlo při zobrazování obrazovky detailu časového razítka.
- Detail chyby – položka se v detailu časového razítka nachází pouze pokud se nepodařilo ověřit časové razítko a zobrazuje informace o chybě při ověřování.

Podrobnosti časového razítka obsahují:

- Předmět certifikátu časové autority – tato informace je k dispozici pouze pokud je v žádosti vyžádáno zaslání certifikátu časové autority. V aplikaci je vyžádání certifikátu časové autority programově tak, aby se certifikát vždy vložil do odpovědi časové autority (TimeStampResponse). Předmět certifikátu obsahuje např. jméno, příjmení, zemi vydání, sériové číslo certifikátu atd.
- UTC čas – obsahuje čas vytvoření časového razítka ve formátu UTC.
- OID politika – obsahuje OID politiku časové autority, podle které bylo vytvořeno časové razítko.
- Nejistota časového údaje – obsahuje přesnost času vytvoření časového razítka.

- Verze – obsahuje verzi protokolu TSP, pomocí kterého bylo vytvořeno časové razítko.
- Sériové číslo – obsahuje sériové číslo časového razítka zobrazené dekadicky a hexadecimálně.
- Algoritmus miniatury – obsahuje použitý hashovací algoritmus, pomocí kterého byl vytvořen otisk zdrojového souboru.

Obecné informace	
Soubor obsahující kvalifikované časové razítko	C:\Users\Jakub\Documents\zadani_str_1001.pdf.tsr
Zdrojový soubor	C:\Users\Jakub\Documents\zadani_str_1001.pdf
Výsledek ověření	Kvalifikované časové razítko je v pořádku
Podrobnosti časového razítka	
Kvalifikovaný systémový certifikát časového serveru	CN=Apple Timestamp Certification Authority,OU=Apple Certification Authority,O=Apple Inc.,C=US
UTC čas	2020.03.23 17:16:23
OID politiky	1.2.3
Nejistota časového údaje ±	1 s
Verze	1
Sériové číslo razítka	4177902637724572717 (39FAE45043BD9C2D)
Algoritmus miniatury	SHA-512

Obr. 43 Příklad zobrazení informací o časovém razítku v aplikaci

Při vytváření obrazovky s informacemi o časovém razítku do ViewModelu vstoupí přes konstruktor zvolené časové razítko ve formátu, v jakém je uloženo v historii časových razítek, viz Obr. 24 Struktura serializačního souboru pro historii vydaných časových razítek. Obecné informace o razítku (umístění souboru s časovým razítkem a umístění zdrojového souboru) jsou k dispozici ihned. Informace o časovém razítku lze získat vytvořením instance třídy *TimeStampResponse* nebo *TimeStampToken* z NuGet balíčku Org.BouncyCastle.

Pro vytvoření instance třídy *TimeStampResponse* nebo *TimeStampToken* je potřeba mít k dispozici soubor časového razítka ve formátu pole bytů. Data časového razítka jsou v historii uložena jako pole bytů. Poslední věc, která zbývá určit, je, zda jsou data ve formátu *TimeStampResponse*, nebo *TimeStampToken*. Typ formátu je v historii také uložen a podle typu se vytvoří buď instance *TimeStampResponse*, nebo *TimeStampToken*.

```

if (SelectedTimeStamp.TimeStampFileType == TimeStampReqRes.FileTypeResponse)
{
    TimeStampResponse response = new TimeStampResponse(SelectedTimeStamp.TimeStampResponse);
    ParseTimeStampResponse(response);
}
else if (SelectedTimeStamp.TimeStampFileType == TimeStampReqRes.FileTypeToken)
{
    TimeStampToken token = new TimeStampToken(new CmsSignedData(SelectedTimeStamp.TimeStampResponse));
    ParseTimeStampToken(token);
}

```

Obr. 44 Vytvoření instance *TimeStampResponse* nebo *TimeStampToken* z historie vydaných časových razítek

Po vytvoření instance je možno získat všechny potřebné informace, které jsou v časovém razítku uloženy a také lze časové razítko ověřit. Proces ověření je totožný s procesem, který je uveden v kapitole 7.6.

Informace o časovém razítku jsou uloženy ve dvou kolekcích typu `ObservableCollection<Tuple<string, string>>`. `Tuple` je třída, která je vhodná použít, pokud programátor nechce vytvářet modelovou třídu pro kolekci, nebo by její vytvoření bylo zbytečné. V detailu časového razítka se použití třídy `Tuple` nabízí, protože zobrazuje jen název položky a její hodnotu. První kolekce slouží pro uložení obecných dat o časovém razítku a do druhé kolekce se ukládají podrobnosti o časovém razítku.

```
GeneralDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TsFilePath, tsFileInfo.FullName));
GeneralDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TsSourceFilePath, SelectedTimeStamp.SourceFilePath));
string errorMessage = VerifyTSResponse(response);
if (string.IsNullOrEmpty(errorMessage))
    GeneralDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_VerifyResult, Resources.TsDetailsVM_VerifyOk));
else
{
    GeneralDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_VerifyResult, Resources.TsDetailsVM_VerifyError));
    errorMessage = errorMessage.Replace("\r\n", " ");
    GeneralDetails.Add(new Tuple<string, string>("Detail chyby", errorMessage));
}
```

Obr. 45 Získání obecných informací o časovém razítku

```
TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TsaCertIssuer,
    response.TimeStampToken.SignerID.Issuer.ToString()));
TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_GenTime, SelectedTimeStamp.GenTime));
TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_OidPolicy,
    response.TimeStampToken.TimeStampInfo.Policy));
if (response.TimeStampToken.TimeStampInfo.Accuracy.Millis != null)
    TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TimeAccuracy,
        $"{response.TimeStampToken.TimeStampInfo.Accuracy.Millis.PositiveValue.ToString()} ms"));
else if (response.TimeStampToken.TimeStampInfo.Accuracy.Micros != null)
    TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TimeAccuracy,
        $"{response.TimeStampToken.TimeStampInfo.Accuracy.Micros.PositiveValue.ToString()} μs"));
else if (response.TimeStampToken.TimeStampInfo.Accuracy.Seconds != null)
    TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_TimeAccuracy,
        $"{response.TimeStampToken.TimeStampInfo.Accuracy.Seconds.PositiveValue.ToString()} s"));
TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_Version,
    response.TimeStampToken.TimeStampInfo.TstInfo.Version.ToString()));
TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_SerialNumber,
    $"{response.TimeStampToken.TimeStampInfo.SerialNumber.ToString()}" +
    $" ({response.TimeStampToken.TimeStampInfo.SerialNumber.ToString(16).ToUpper()})"));
foreach (ShaAlgorithmType sha in Manager.Instance.SettingsManager.TsaShaAlgorithms)
{
    if (sha.Oid == response.TimeStampToken.TimeStampInfo.MessageImprintAlgOid)
    {
        TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_ShaAlgorithm, sha.Name));
        break;
    }
}
if (response.TimeStampToken.TimeStampInfo.Nonce != null)
    TimeStampDetails.Add(new Tuple<string, string>(Resources.TsDetailsVM_Nonce,
        $"{response.TimeStampToken.TimeStampInfo.Nonce.ToString()}" +
        $" ({response.TimeStampToken.TimeStampInfo.Nonce.ToString(16).ToUpper()})"));
```

Obr. 46 Získání podrobností o časovém razítku

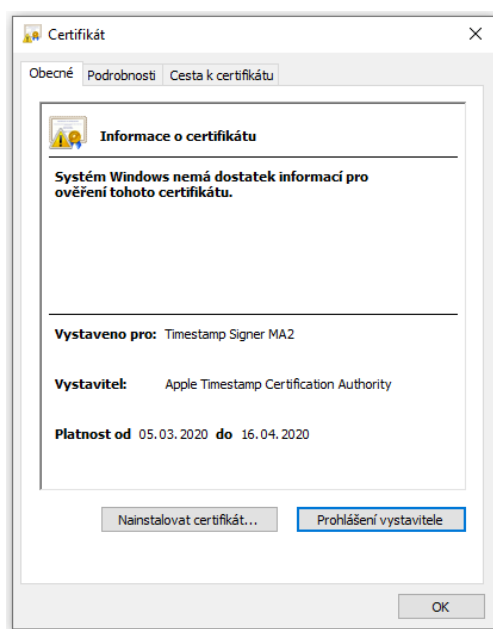
7.8.2 Dodatečné akce dostupné v detailu časového razítka

V obrazovce detailu časového lze provést následující akce:

- Zobrazit certifikát časové autority – podrobnosti o certifikátu časové autority se zobrazí pomocí systémového dialogu, který lze zobrazit pomocí třídy *X509Certificate2UI* a její metody *DisplayCertificate*, která jako vstupní parametry přijímá certifikát, který se má zobrazit a poté tzv. Handle okna, který slouží k tomu, aby se systémový dialog zobrazil v místě na displeji, kde je umístěna i aplikace.

```
IX509Store store = token.GetCertificates("collection");
ArrayList certs = new ArrayList(store.GetMatches(null));
foreach (Org.BouncyCastle.X509.X509Certificate cert in certs)
{
    if (cert.IssuerDN.ToString(true, new Hashtable()) == token.SignerID.Issuer
        .ToString(true, new Hashtable()))
    {
        X509Certificate2 cert2 = new X509Certificate2(cert.GetEncoded());
        IntPtr hwnd = new WindowInteropHelper(_mainWinView).Handle;
        X509Certificate2UI.DisplayCertificate(cert2, hwnd);
    }
}
```

Obr. 47 Zdrojový kód pro zobrazení podrobností o certifikátu



Obr. 48 Systémový dialog pro zobrazení podrobností o certifikátu

- Uložit žádost o časové razítko – do historie vydaných razítek se v aplikaci přidává také žádost o časové razítko, pro případ, že by jí uživatel chtěl uložit. Při ukládání žádosti se nejprve musí zvolit adresář kam chce uživatel soubor uložit. Pro výběr adresáře lze použít opět systémový dialog, který lze vytvořit v .NET Frameworku

pomocí třídy *SaveFileDialog* z oboru názvů *System.Windows.Forms*. Na obr. 49 lze vidět, jak se dialog vytváří. Dialog umožňuje různé nastavení, jako např. výchozí použitá přípona, výchozí název ukládaného souboru, nadpis dialogu, filtr atd. Po potvrzení dialogu se vezme cesta, kterou uživatel zvolil a soubor se uloží pomocí třídy *File* a její metody *WriteAllBytes*, která přijímá parametry cestu, kde se má soubor vytvořit a data, která se mají uložit.

```
using (SaveFileDialog dialog = new SaveFileDialog())
{
    dialog.AddExtension = false;
    dialog.DefaultExt = ".tsreq";
    dialog.Title = Resources.TsDetailsVM_SaveTsreqTitle;
    dialog.FileName = SelectedTimeStamp.SourceFilePath + ".tsreq";
    dialog.Filter = $"{Resources.TsDetailsVM_Tsreq}|*.tsreq|{Resources.CreateTsVM_Allfiles}|*.*";
    var result = dialog.ShowDialog();
    if(result == DialogResult.OK)
        File.WriteAllBytes(dialog.FileName, SelectedTimeStamp.TimeStampRequest);
}
```

Obr. 49 Vytvoření dialogu pro uložení souboru)

- Uložit časové razítko ve formátu *TimeStampResponse* – provede se stejným způsobem jako ukládání žádosti, jen se uloží časové razítko. Možnost uložit časové razítko ve formátu *TimeStampResponse* lze pouze pokud je i v historii razítko uloženo ve formě *TimeStampResponse*. Nelze uložit časové razítko, které je formátu *TimeStampToken* jako *TimeStampResponse*, protože *TimeStampResponse* obsahuje celou odpověď přijatou ze serveru a *TimeStampToken* obsahuje pouze token časového razítka.
- Uložit časové razítko ve formátu *TimeStampToken* – uložení se provede stejným způsobem jako uložení žádosti o časové razítko.

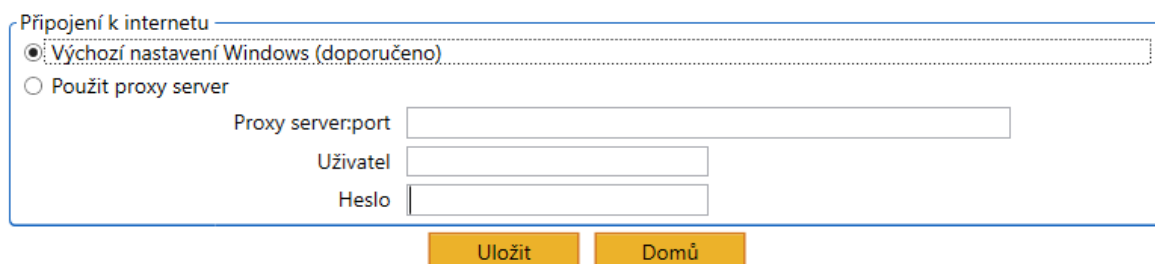
7.9 Nastavení aplikace

Nastavení aplikace se skládá ze dvou pod obrazovek. První obrazovka se týká možnosti připojení k internetu a druhá obrazovka obsahuje různé možnosti nastavení služby tsa. Nastavení aplikace je kritické pro správnou funkčnost aplikace, protože pokud si uživatel nejdříve nenastaví jednotlivé položky nastavení, nebude moci vytvářet časová razítka.

7.9.1 Nastavení připojení k internetu

V nastavení připojení si lze nastavit, jak bude probíhat komunikace s časovou autoritou při vytváření časového razítka. Lze nastavit klasické systémové nastavení, nebo pomocí Proxy

serveru. Nastavení připojení k internetu se poté použije při vytváření http požadavku v procesu vytvoření časového razítka (viz kapitola 7.5).



Připojení k internetu

☒ Výchozí nastavení Windows (doporučeno)

☐ Použít proxy server

Proxy server:port

Uživatel

Heslo

Uložit Domů

Obr. 50 Obrazovka Nastavení připojení k internetu

7.9.2 Nastavení služby TSA

V obrazovce Nastavení služby TSA si lze přednastavit všechny hodnoty, které se zobrazují v dialogu Vytvoření časového razítka (viz kapitola 7.5) a jsou poté využity při procesu vytvoření časového razítka. Je možno si přednastavit typ hashovacího algoritmu, který se použije při vypočtení otisku zdrojových dat a politiku časové autority.

Nejdůležitější volbou je poté server časové autority, na kterou se bude posílat žádost o vytvoření časového razítka. Servery časových autorit, které jsou dostupné pro tvorbu časových razítek nejsou v aplikaci hard kódovány, ale přebírají se ze souboru TSA_urls.ini, který se nachází v adresáři aplikace. Není tedy potřeba pracovat jen dostupnými časovými autoritami, ale lze si také přidat další servery časových autorit.

Dále se v obrazovce nacházejí možnosti autentizace k službě vydání časového razítka, pokud je zvolena časová autorita, která má své služby zpoplatněny. Aplikace umožňuje nastavit čtyři druhy autentizace:

- Žádná autentizace – nutno použít tuto formu autentizace, pokud je zvolen poskytovatel časových razítek, který své služby poskytuje zdarma.
- Autentizace pomocí certifikátu – po zakoupení služby dostane uživatel od poskytovatele časových razítek autentizační certifikát, který se poté importuje do používaného zařízení. Autentizační certifikát volí pomocí tlačítka „Změnit“. Následně se autentizační certifikát bude vkládat do hlavičky http požadavku s žádostí o časové razítko.
- Autentizační IP adresa – uživatel při zakoupení služby vytváření časových razítek nahlásí poskytovateli svou IP adresu, která pote bude sloužit k autentizaci.

- Autentizace pomocí přihlašovacích údajů – uživatel dostane po zakoupení služby na vytváření časových razítek přihlašovací údaje, které vloží do příslušných políček (Přihlašovací jméno a Heslo) v aplikaci. Poté se tyto údaje vloží do hlavičky http požadavku, kterým se zasílá žádost o vytvoření časového razítka.

Poté co se nastaví všechny požadované položky, je nutno nastavení uložit a serializovat, aby jej bylo možno použít v aplikaci. Proto je na obrazovce dostupné tlačítko „Uložit“. To uloží nastavené hodnoty do instance třídy *SettingsManager*, která je přístupná ve třídě *Manager*, a nakonec ještě zavolá serializaci hodnot nastavení do XML souboru, aby jej bylo možno načíst i při dalších spuštění aplikace (viz kapitola 7.3).

Autentizace služby TSA

Časová autorita

Hash Algoritmus ☐ SHA-1 ☐ SHA-256 ☒ SHA-512

Politika časové autority
Nepovinné. Pokud není zadáno je použita výchozí politika časové autority

Server časové autority
* Adresu zadejte ručně nebo vyberte jednu možnost *

☒ Žádná autentizace

Žádnou autentizaci lze provést pouze u serverů časových autorit, které nejsou zpoplatněny.

☐ Autentizační certifikát

Zobrazit Změnit

Autentizační certifikát je vyžadován pouze v případě neanonymní HTTPS komunikace

☐ Autentizační IP adresa

Požadavek na změnu IP adresy zašlete na tsa@ica.cz nebo svému obchodnímu zástupci. Vždy uvádějte kontaktní údaje ze smlouvy.

Vaši veřejnou adresu lze zjistit např. pomocí následující internetové stránky <http://whatismyipaddress.com/>

☐ Autentizační jméno

Uživatelské jméno

Uživatelské heslo

Požadavek na změnu jména a hesla zašlete na tsa@ica.cz nebo svému obchodnímu zástupci. Vždy uvádějte kontaktní údaje ze smlouvy. Změnu hesla můžete provést sami, případně o ní též požádat.

Uložit Domů

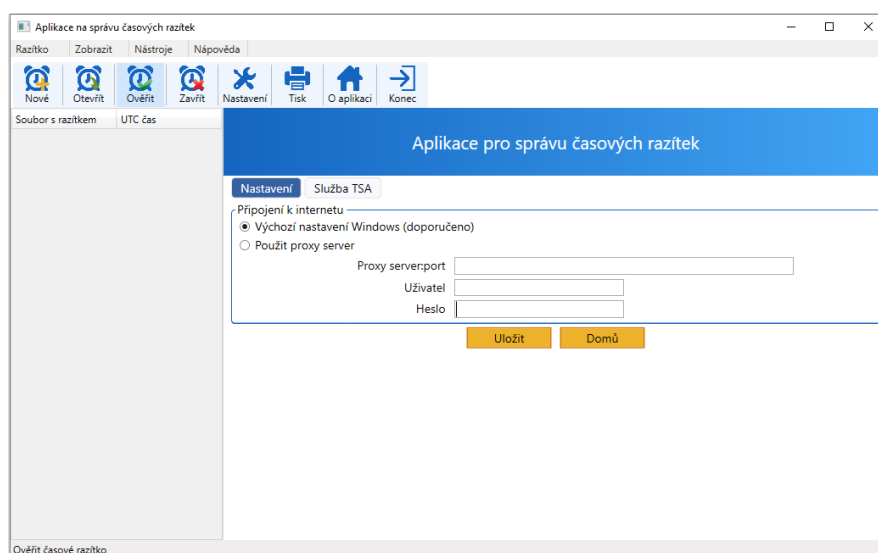
Obr. 51 Obrazovka Nastavení služby TSA

8 PREZENTACE VÝSLEDNÉ APLIKACE

Po dokončení programování aplikace je možno se podívat na výslednou funkcionalitu aplikace.

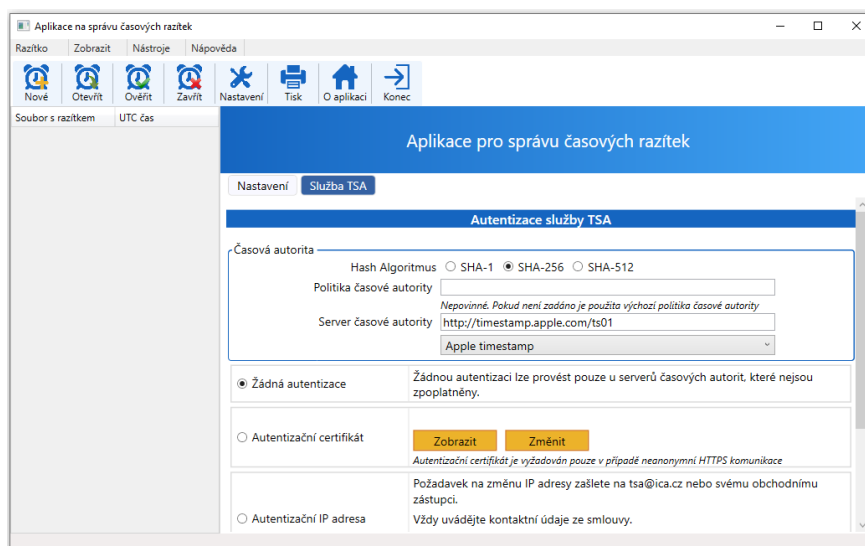
8.1 První spuštění aplikace

Při úplně prvním spuštění aplikace v počítači je potřeba nejprve provést nastavení aplikace, aby bylo možno úspěšně vytvořit časové razítko. Je potřeba nastavit způsob připojení (pokud je požadováno, může se nastavit Proxy server). Při ukázce je použito nastavení výchozího nastavení Windows.



Obr. 52 První nastavení aplikace, sekce Obecné nastavení

Nejdůležitější je nastavení v sekci Služba TSA, kde se nastavují výchozí možnosti při tvorbě žádosti o časové razítko. Výchozí hashovací algoritmus, použitý server časové autority, požadovaná politika časové autority a autentizace žádosti o časové razítko. V ukázce je použit server časové autority od společnosti Apple, výchozí hashovací algoritmus je SHA-256 a není zvolena žádná autentizace.

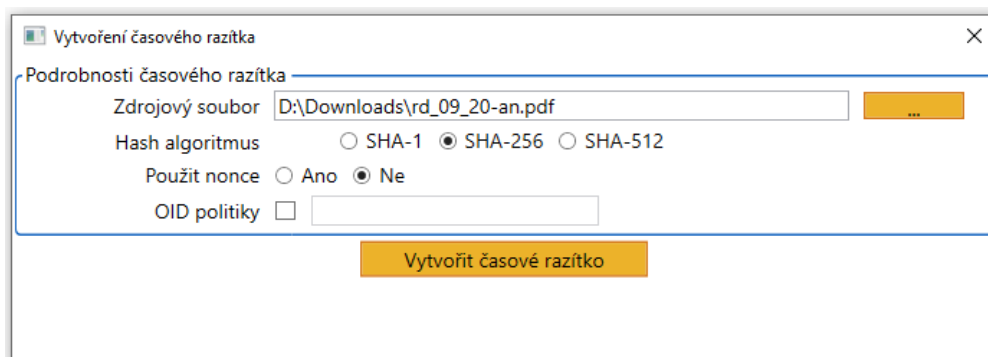


Obr. 53 První nastavení aplikace, sekce Služba TSA

Po nastavení aplikace se provede uložení nastavení pomocí tlačítka „Uložit“, které se na obrazovce nachází. Po dokončení nastavení aplikace lze přejít k vytvoření časového razítka.

8.2 Vytvoření časového razítka

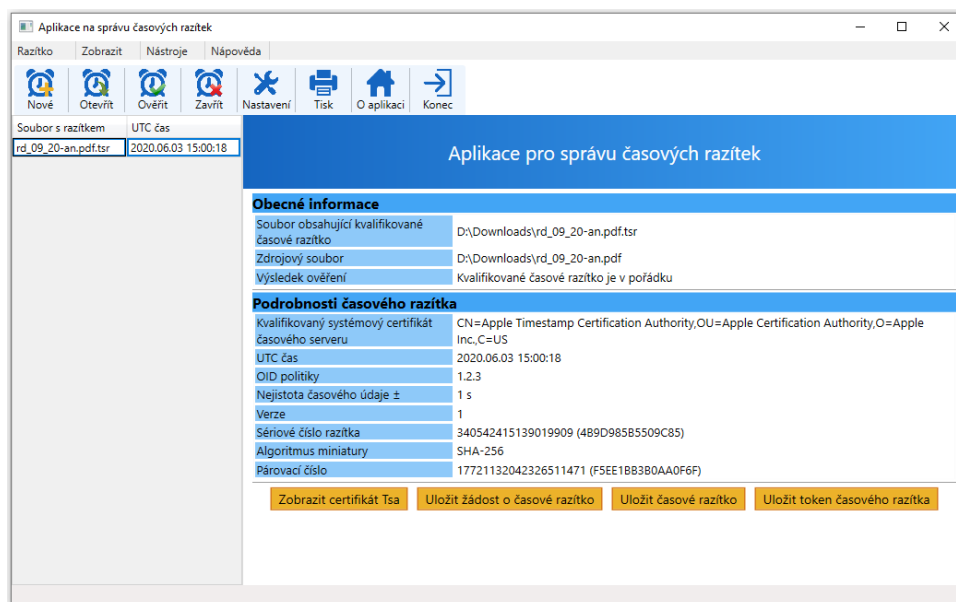
Spuštění procesu vytvoření časového razítka se provede stisknutím tlačítka „Nové“, umístěné na hlavní obrazovce aplikace. Nejprve se zvolí zdrojový soubor, ke kterému se vytvoří časové razítko. V ukázce se bude vytvářet časové razítko k souboru typu .PDF. Po zvolení souboru si lze dodatečně nastavit typ hashovacího algoritmu a použití Nonce. V ukázce se použije algoritmus SHA-256 a nechá se vložit Nonce do žádosti o časové razítko.



Obr. 54 Nastavený dialog při vytvoření časového razítka

Po nastavení požadovaných hodnot se může vytvořit žádost o časové razítko a odeslat na server časové autority. Tento proces spustí tlačítko „Vytvořit časové razítko“. Po stisknutí je potřeba počkat na odpověď časové autority. Po dokončení procesu tvorby časového

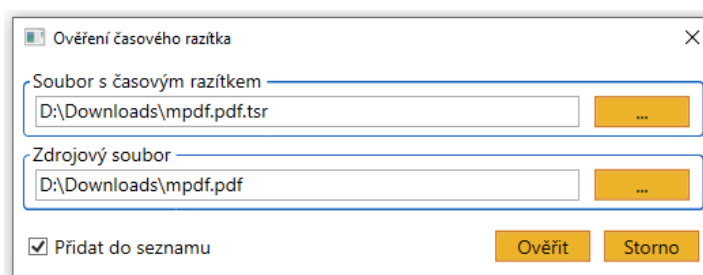
razítka se razítko přidá do historie vydaných razítek a zobrazí se detail právě vytvořeného časového razítka.



Obr. 55 Detail vydaného časového razítka

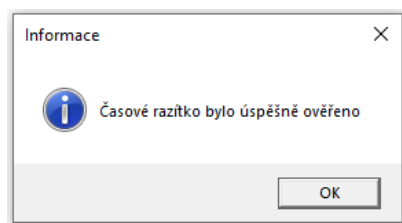
8.3 Ověření / Otevření časového razítka

Aplikace umožňuje ověření / otevření již existujícího časového razítka. Proces lze zahájit pomocí tlačítka „Otevřít“ nebo „Ověřit“. Po stisku tlačítka se zobrazí dialog, ve kterém je možno vybrat soubor s časovým razítkem. Po výběru časového razítka se pokusí aplikace najít zdrojový soubor. Pokud se to aplikaci nepodaří, tak jej vybere sám uživatel.



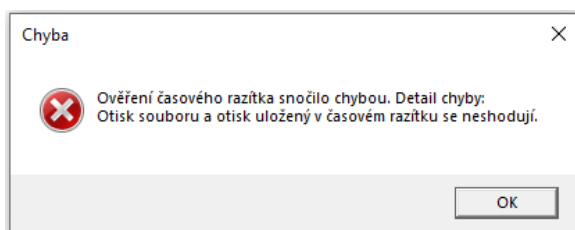
Obr. 56 Dialog ověření / otevření časového razítka

Po zvolení souboru s časovým razítkem a zdrojového souboru lze spustit proces samotného ověření pomocí tlačítka „Ověřit“. Výsledek ověření je uživateli sdělen pomocí informačního dialogu.



Obr. 57 Výsledek ověření – úspěšné ověření

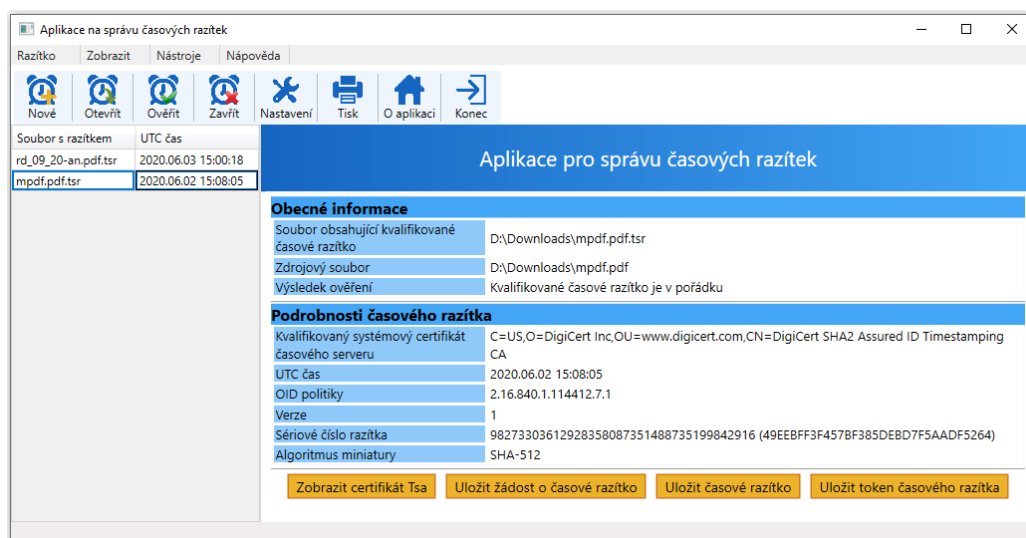
Pro ukázkou lze také nasimulovat selhání ověření časového razítka. V ukázce se chyby při ověření docílí pomocí zvolení jiného souboru, ke kterému dané časové razítko vůbec nebylo vytvořeno. Poté se uživateli zobrazí chybová hláška s informací, proč ověření razítka selhalo.



Obr. 58 Výsledek ověření – chyba

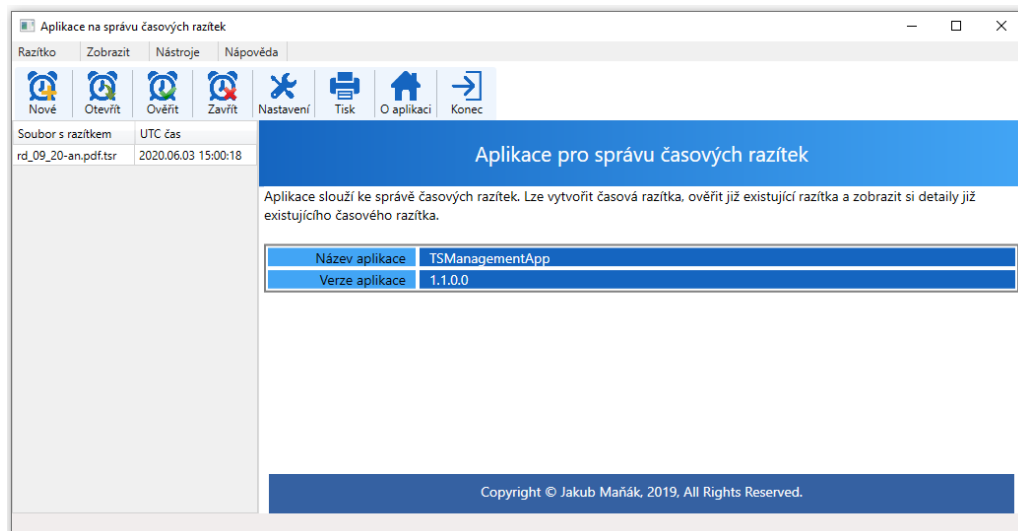
8.4 Zobrazení detailu časového razítka z historie

Všechna časová razítka, která byla uživatelem vytvořena nebo dodatečně otevřena se ukládají do historie vydaných razítek. Historie se nachází v levé části hlavního okna. Pro zobrazení časového razítka lze jednoduše kliknout na požadované razítko umístěné v historii a uživateli je zobrazen detail daného časového razítka.



Obr. 59 Otevření detailu časového razítka z historie vydaných razítek

Pokud uživatel bude mít dané časové razítko uchované v historii, může ho z ní odstranit pomocí tlačítka „Zavřít“. V ukázce je odebráno časové razítko, které bylo do historie přidáno pomocí funkce Otevření časového razítka (viz kapitola 8.3).



Obr. 60 Ukázka odstranění časového razítka

Rozdíl mezi obr. 60 a obr. 59 je ten, že na obr. 60 se v historii vydaných razítek již nenachází časové razítko „mpdf.pdf.tsr“.

ZÁVĚR

Tato bakalářská práce měla několik hlavních cílů. Prvním cílem bylo seznámit čtenáře s pojmem časové razítko, popsat jeho strukturu a proces vytvoření a ověření. Druhým cílem bylo rozvést problematiku spojenou s aplikacemi pro správu časových razítek a popsat funkce, které by aplikace pro správu časových razítek měly obsahovat. Dalším cílem bylo čtenáře informovat o možnostech, kterými by se aplikace mohla vytvořit. Z popsaných možností si jednu možnost zvolit a pomocí ní aplikaci naprogramovat. Posledním cílem bylo ukázat čtenáři výslednou aplikaci a prezentovat její nejdůležitější funkce.

V teoretické části bakalářské práce byly popsány hashovací algoritmy, které jsou nedílnou součástí pro vytvoření časového razítka a pojem „hashovací algoritmus“ je v práci nespočetněkrát použit. Dále jsou v teoretické části detailně rozebrány pojmy časové razítko, autority vydávající časová razítka, průběh vytvoření a ověření časového razítka. Posledním tématem teoretické části byl popis jednotlivých funkcí, které by aplikace pro správu časových razítek měly a mohly obsahovat.

Praktická část bakalářské práce se nejprve zabývala rozebráním metod a prostředků, pomocí kterých by bylo možno aplikaci vytvořit. Byly zmíněny platformy, na jaké aplikace mohla cílit, tedy desktopové systémy (aplikace v příkazové řádce nebo s grafickým rozhraním), mobilní aplikace (Android, iOS, hybridní vývoj). Dále programovací jazyky, pomocí kterých by bylo možno aplikaci naprogramovat – Java, C#, C++, Python, SWIFT.

Z uvedených platforem byly vybrány desktopy s operačním systémem Windows a podle zvolené platformy byl vybrán programovací jazyk C#, .NET Framework a grafická knihovna WPF. Poté byly zvolené prostředky a metody popsány a také byl popsán návrhový vzor MVVM, který byl při tvorbě aplikace použit.

Největší část praktické části se věnuje programování aplikace, přičemž k implementaci bylo použito vývojové prostředí Visual Studio 17 a později Visual Studio 19. Byla popsána základní struktura projektu a jednotlivé části rozebrány. Bylo popsáno programové zpracování normy RFC-3161, uložení sdílených dat potřebných k běhu aplikace, serializace dat, popis programování jednotlivých obrazovek a jejich funkcí s ukázkami důležitých částí kódu a výslednou obrazovkou.

Následně v další fázi byla výsledná aplikace otestována a vyzkoušena. Byl ukázán proces správného nastavení aplikace, aby ostatní funkce správně fungovaly. Poté byl ukázán postup

při vytvoření, ověření a otevření časového razítka. Bylo ukázáno, jak manipulovat s historií vydaných časových razítek.

Výsledná funkční aplikace obsahuje všechny důležité a také většinu doplňujících funkcí, které byly uvedeny v teoretické části. Pro testování aplikace bylo vybráno šest poskytovatelů časových razítek, které službu poskytují zdarma. Aplikace by měla být schopna vytvářet časová razítka i u placených poskytovatelů časových razítek.

Do budoucna by aplikace mohla být vylepšena o práci ve více vláknech. Aktuálně aplikace běží synchronně a může nastat situace, kdy např. při tvorbě časového razítka může aplikace zatuhnout z důvodu dlouhého čekání na odpověď serveru časové autority.

Aplikaci lze rozšířit o průvodce přidání nových poskytovatelů časových razítek. Nyní lze přidat nové poskytovatele manuálně do konfiguračního souboru. Po přidání poskytovatele do konfiguračního souboru je ještě nutno vyzkoušet u nového poskytovatele vytvořit časové razítko, protože může nastat situace, kdy poskytovatel např. již nemusí existovat. Při hledání poskytovatelů často nastala situace, že server poskytovatele vracel chybu nebo již neexistoval. Proto by aplikace mohla tímto procesem uživatele provést, dostupnost poskytovatele ověřit a podle výsledku nového poskytovatele přidat nebo informovat uživatele o chybě.

Závěrem lze říci, že aplikace byla vytvořena se všemi náležitostmi a s nejdůležitějšími funkcemi pro správu časových razítek, tedy jejich vytvoření a ověření.

SEZNAM POUŽITÉ LITERATURY

- [1] DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. *Velký průvodce infrastrukturou PKI*. 2. aktualizované vydání. Brno: Computer Press, 2009. ISBN 9788025145142.
- [2] RFC 3161. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. U.S.A: IETF, 2001.
- [3] Secure Hash Algorithms: Comparison of SHA functions. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020, 16 January 2020 [cit. 2020-03-09]. Dostupné z: https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
- [4] Co je Xamarin.Forms? In: *Microsoft Docs: Co je Xamarin.Forms?* [online]. Redmond, 980 52, USA: Microsoft, 2019, 18. 09. 2019 [cit. 2020-03-13]. Dostupné z: <https://docs.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin-forms>
- [5] Přehled rozhraní .NET Framework. In: *Microsoft Docs* [online]. USA: Microsoft, 2017, 30. 03. 2017 [cit. 2020-03-16]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/framework/get-started/overview>
- [6] Úvod do NuGetu. In: *Microsoft Docs* [online]. USA: Microsoft, 2019, 24. 05. 2019 [cit. 2020-03-16]. Dostupné z: <https://docs.microsoft.com/cs-cz/nuget/what-is-nuget>
- [7] Přehled grafického subsystému WPF (Windows Presentation Foundation). *Microsoft Docs* [online]. USA: Microsoft, 2016, 4.11.2016 [cit. 2020-03-17]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/framework/wpf/introduction-to-wpf>
- [8] DAJBYCH, Václav. MVVM: Model-View-ViewModel. Dotnetportal [online]. ČR, 2009, 21.04.2009 [cit. 2020-03-19]. Dostupné z: <https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel>
- [9] BindingMode Enum. *Microsoft Docs* [online]. USA: Microsoft, c2020 [cit. 2020-03-19]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/system.windows.data.bindingmode?redirectedfrom=MSDN&view=netframework-4.8>
- [10] UpdateSourceTrigger Enum. *Microsoft Docs* [online]. USA: Microsoft, c2020 [cit. 2020-03-19]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/system.windows.data.updatesourcetrigger?view=netframework-4.8>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

TSP	Time Spamping Protocol (protokol pro vydávání časových razítek)
TSA	Time Stamp Authority (autorita pro vydávání časových razítek)
TSU	Time Stamp Unit
OID	Object IDentifier (jedinečný identifikátor objektu)
UTC	Coordinated Universal Time (koordinovaný světový čas)
CLI	Command Line Interface
CLR	Common Language Runtime
VS	Vývojové prostředí Visual Studio
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
MVVM	Model-View-ViewModel
IP	Internet Protocol

SEZNAM OBRÁZKŮ

Obr. 1 Schéma žádosti o časové razítko	14
Obr. 2 Schéma odpovědi z časové autority	16
Obr. 3 Schéma statusu	17
Obr. 4 Datová struktura objektu CMS SignedData (vlastní tvorba)	19
Obr. 5 Schéma TSA (vlastní tvorba)	23
Obr. 6 Prohlížeč NuGet balíčků ve VS	33
Obr. 7 Ukázka jazyka XAML	35
Obr. 8 Ukázka kódu na pozadí	36
Obr. 9 Nastavení DataContextu pomocí XAML	38
Obr. 10 Nastavení DataContextu pomocí kódu na pozadí	38
Obr. 11 Ukázka propojení View a ViewModelu	38
Obr. 12 Struktura projektu	41
Obr. 13 Obsah oboru názvu Properties	42
Obr. 14 Obsah oboru názvu Communication	42
Obr. 15 Příklad použití konvertoru	42
Obr. 16 Obsah oboru názvu Converters	43
Obr. 17 Obsah oboru názvu Enums	43
Obr. 18 Obsah oboru názvu Models	43
Obr. 19 Obsah oboru názvu Utils	44
Obr. 20 Obsah oboru názvu ViewModels	44
Obr. 21 Obsah oboru názvu Views	45
Obr. 22 Struktura třídy BouncyCastleTSPProvider	45
Obr. 23 Vývojový diagram funkce RequestTimeStamp	46
Obr. 23 Ukázka vytvoření Singleton Property	48
Obr. 24 Ukázka získání dat z nastavení pomocí Managera	48
Obr. 24 Struktura serializačního souboru pro historii vydaných časových razítek	49
Obr. 25 Struktura serializačního souboru nastavení	50
Obr. 26 Ukázka vytváření XML souboru v C# .NetFramework	51
Obr. 27 Ukázka deserializace XML dokumentu	52
Obr. 28 Ukázka tvorby lišty záložek	53
Obr. 29 Ukázka tvorby panelu nástrojů	53
Obr. 30 Ukázka definice sloupců v DataGrid	53
Obr. 31 Ukázka použití UserControl v aplikaci	54
Obr. 32 Hlavní okno aplikace	54

Obr. 33 Ukázka vytvoření dialogu pro získání cesty souboru	55
Obr. 34 Dialogové okno pro vytvoření časového razítka	56
Obr. 35 Proces vytvoření časového razítka v aplikaci.....	58
Obr. 36 Dialogové okno funkce Ověření časového razítka	59
Obr. 37 Funkce pro nalezení zdrojového souboru	60
Obr. 38 –Metoda zjišťující, zda je ověřovaný soubor ve formátu TimeStampResponse	60
Obr. 39 Proces ověření TimeStampResponse.....	61
Obr. 40 Proces ověření TimeStampToken.....	62
Obr. 41 Vývojový diagram procesu ověření časového razítka	62
Obr. 42 Funkce pro odstranění časového razítka z historie vydaných časových razítek.....	63
Obr. 43 Příklad zobrazení informací o časovém razítku v aplikaci.....	65
Obr. 44 Vytvoření instance TimeStampResponse nebo TimeStampToken z historie vydaných časových razítek	65
Obr. 45 Získání obecných informací o časovém razítku	66
Obr. 46 Získání podrobností o časovém razítku	66
Obr. 47 Zdrojový kód pro zobrazení podrobností o certifikátu.....	67
Obr. 48 Systémový dialog pro zobrazení podrobností o certifikátu	67
Obr. 49 Vytvoření dialogu pro uložení souboru).....	68
Obr. 50 Obrazovka Nastavení připojení k internetu	69
Obr. 51 Obrazovka Nastavení služby TSA.....	70
Obr. 52 První nastavení aplikace, sekce Obecné nastavení	71
Obr. 53 První nastavení aplikace, sekce Služba TSA.....	72
Obr. 54 Nastavený dialog při vytvoření časového razítka.....	72
Obr. 55 Detail vydaného časového razítka	73
Obr. 56 Dialog ověření / otevření časového razítka	73
Obr. 57 Výsledek ověření – úspěšné ověření	74
Obr. 58 Výsledek ověření – chyba.....	74
Obr. 59 Otevření detailu časového razítka z historie vydaných razítek	74
Obr. 60 Ukázka odstranění časového razítka.....	75

SEZNAM PŘÍLOH

Příloha P I: Obsah CD

Příloha P II: Aplikace TSManagementApp

Příloha P III: Návod k aplikaci

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář **Text bakalářské práce** – obsahuje text bakalářské práce ve formátu PDF;
- Adresář **Zdrojové kódy** – obsahuje projekt spustitelný ve vývojovém prostředí Visual Studio 19. Součástí projektu jsou všechny zdrojové soubory.
- Adresář **Aplikace** – obsahuje zkompilovanou verzi aplikace, která je spustitelná na operačních systémech Windows. Aplikaci lze spustit pomocí souboru TsManagementApp.exe. V adresáři se nachází také soubor **TsManagementApp – nápověda.pdf**, která slouží jako návod pro použití aplikace.

PŘÍLOHA P II: APLIKACE TSMANAGEMENTAPP

Zkompilovaná verze aplikace zabalena do zip archivu. Pro spuštění aplikace stačí rozbalit zip soubor do libovolného adresáře a pomocí souboru TsManagementApp.exe aplikaci spustit.

PŘÍLOHA P III: NÁPOVĚDA K APLIKACI

Nápověda k aplikaci je sepsána v souboru **TsManagementApp – nápověda.pdf**. V nápovědě je popsán postup nastavení aplikace, aby bylo možno dosáhnout správné funkce aplikace. V nápovědě jsou dále popsány všechny důležité funkce aplikace.