Re-engineering nástroje pro podepisování, šifrování a dešifrování dokumentů

Bc. Jaroslav Sýkora

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jaroslav Sýkora**
Osobní číslo:	**A21636**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Re-engineering nástroje pro podepisování, šifrování a dešifrování dokumentů**
Téma práce anglicky:	**Re-engineering Tools for Signing, Encrypting and Decrypting Documents**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj multi-platformních aplikací.
2. Vhodně zvolte technologie pro tvorbu aplikace.
3. Analyzujte původní aplikaci.
4. Vypracujte návrh nové verze aplikace na základě modernější technologií.
5. Naprogramujte aplikaci dle původního vzoru.
6. Testujte výslednou aplikaci a odladťe případné nedostatky.
7. Demonstrujte výsledky a formulujte závěr včetně doporučení pro budoucí rozvoj aplikace.

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

Seznam doporučené literatury:

1. ROBINSON, Simon. C#: programujeme profesionálně. Přeložil Bogdan KISZKA. Brno: Computer Press, 2003. ISBN 80--251-0085-5.
2. HEJLSBERG, Anders. The C# programming language. 4th ed. Addison-Wesley Professional, 2004. ISBN 978-0-321--74176-9.
3. FILIPOVA, Olga a Rui VILÃO. Software Development From A to Z: A Deep Dive into all the Roles Involved in the Creation of Software. Imprint: Apress, 2018. ISBN 9781484239445.
4. MENEZES, A. J., Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. ISBN 9780849385230.
5. KATZ, Jonathan a Yehuda LINDELL. Introduction to modern cryptography. Second edition. Boca Raton, FL: CRC Press, [2015]. ISBN 1466570261id.

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D.
děkan

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

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové 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 diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové 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 diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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 diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 25.05.2023

Jaroslav Sýkora, v. r.
…….
podpis studenta

ABSTRAKT

Účelem práce je převést již existující nástroj Crypto Native Application (CNA) ze zastaralého jazyka Pascal do modernějšího C#. CNA je program sloužící pro podepisování, šifrování a dešifrování dokumentů, který mohou využívat zákazníci informačních systémů.

V teoretické části je popsán současný stav technologií pro vývoj multiplatformních aplikací. Dále je zde představena původní aplikace a technologie použité pro realizaci její náhrady. Praktická část se zabývá již samotným vývojem aplikace.

Klíčová slova: C#, .NET MAUI, reverzní inženýrství, multi-platformní aplikace

ABSTRACT

The purpose of this work is to convert the existing Crypto Native Application (CNA) tool from the obsolete Pascal language to the more modern C#. CNA is a program used for signing, encrypting and decrypting documents and might be used by customers of the information systems.

The theoretical section describes the current state of the art for developing cross-platform applications. It also presents the original application and the technologies used to implement its replacement. The practical part deals with the actual development of the application.

Keywords: .NET MAUI, C#, cross-platform application, reverse engineering

Touto cestou bych rád poděkoval svému vedoucímu práce Ing. Bc. Pavlu Vařachovi, Ph.D. za cenné rady, ochotu a trpělivost v průběhu zpracování této diplomové práce.

OBSAH

ÚVOD

Tato diplomová práce se zaměřuje na reverzní inženýrství nástroje pro podepisování, šifrovaní a dešifrování. Cílem této práce je prokázat možné převedení aplikace do moderního prostředí s využitím multi-platformních a bezpečnostních technologií, jako jsou .NET MAUI a SecureBlackbox. Dále analyzovat proces vývoje právě s využitím výše zmíněných prostředků a popsat případné výhody nebo problémy při jejich použití.

Práce se skládá z teoretické a praktické části. Teoretická část začíná přehledem vývoje multi-platformních aplikací a technologií používaných při jejich vývoji. Dále se zaměřuje již na konkrétní technologii, a to .NET MAUI, což je framework pro tvorbu multi-platformních aplikací. Jazyk C# je důkladně rozebrán z hlediska objektově orientovaného programování, které je klíčové pro vývoj aplikací v prostředí .NET. Další kapitola se věnuje samotnému reverznímu inženýrství a principům, které jsou během tohoto procesu využívány. Zde jsou popsány metody a nástroje používané pro analýzu existujícího softwaru. Dále se práce zaměřuje na zabezpečení dokumentů a typy jejich zabezpečení. Jsou zde popsána i obecná bezpečnostní doporučení a možné kybernetické hrozby, před kterými je nutné dokumenty chránit.

V praktické části práce je představena původní aplikace pro podepisování, šifrovaní a dešifrování. Následuje návrh nové aplikace, který vychází z poznatků získaných při reverzním inženýrství. Poté je popsán proces vývoje nové aplikace s využitím vhodných technologií a jazyků, jako je .NET MAUI a jazyk C#. Dále jsou zanalyzovány problémy, které se vyskytly během vývoje a jejich případná řešení. V závěrečné části práce je představena výsledná aplikace. Součástí této části bude také návrh doporučení pro další rozvoj a vylepšení aplikace na základě zjištěných poznatků z reverzního inženýrství.

I. TEORETICKÁ ČÁST

1 VÝVOJ MULTI-PLATFORMNÍCH APLIKACÍ

Vývoj multi-platformních aplikací je proces, při kterém se vyvíjí aplikace, které mohou být spuštěny na více platformách, jako jsou mobilní zařízení, webové prohlížeče nebo desktopová zařízení s různými operačními systémy. Tyto aplikace umožňují uživatelům používat stejné funkce a data na různých zařízeních, což jim poskytuje větší flexibilitu a pohodlí.

V minulosti se vývojáři museli učit programovat pro každou platformu zvlášť, což bylo náročné a zdlouhavé. Ale s příchodem webových technologií, jako je HTML, CSS a JavaScript, se stal vývoj multi-platformních aplikací mnohem jednodušší. Tyto technologie totiž umožňují vývojářům vyvíjet aplikace pro web, které mohou být spuštěny na libovolném zařízení s webovým prohlížečem. V průběhu let se pak jejich vývoj výrazně zlepšil, díky rozvoji nových technologií a nástrojů, jako jsou React Native, Flutter, Xamarin nebo právě .NET MAUI. Tyto nástroje poskytují vývojářům vysokou míru produktivity a umožňují jim vyvíjet aplikace pro více platforem současně, aniž by museli opakovaně psát kód pro každou platformu zvlášť. V budoucnosti se očekává, že vývoj multi-platformních aplikací bude pokračovat v růstu a rozvoji. Nové technologie a nástroje pravděpodobně zlepší produktivitu vývojářů a umožní jim vyvíjet aplikace pro ještě více platforem s ještě větší efektivitou. Navíc s rostoucí popularitou chytrých telefonů a dalších zařízení bude stále větší poptávka po multi-platformních aplikacích, což bude motivovat vývojáře k jejich vývoji. Kromě toho se očekává, že se vývoj multi-platformních aplikací bude stále více soustředit na uživatelské zážitky. Vývojáři budou chtít poskytovat uživatelům co nejlepší možnou zkušenost na všech platformách, a to prostřednictvím intuitivního a příjemného rozhraní, vysokého výkonu a dostupnosti v reálném čase. [1;2]

1.1 Výhody

- **Úspora času a nákladů**
    - Vývojáři mohou psát jednotný kód pro více platforem, což umožňuje snížit náklady a čas potřebný k vývoji aplikace pro každou platformu zvlášť.

- **Konzistentní uživatelská zkušenost**
    - Použitím jednotného zdrojového kódu mohou být uživatelské zkušenosti (UX) stejné na všech platformách.
- **Snadnější správa projektů**
    - Umožňují vývojářům používat stejné nástroje pro vývoj aplikace, jako jsou například integrované vývojové prostředí (IDE) a nástroje pro testování.

1.2 Nevýhody

- **Horší výkon**
    - Abstrakce nad platformovými specifikacemi a kompilace do nativního kódu mohou mít vliv na výkon aplikace, což může být zvláště důležité u aplikací s vysokými požadavky na výkon.
- **Omezené možnosti**
    - Některé funkce specifické pro určitou platformu mohou být omezené nebo nedostupné při použití těchto technologií a frameworků.
    - Design multi-platformních aplikací musí být též univerzální pro všechny platformy, což značně limituje jeho přizpůsobivost.
    - Některé hardwarové funkce na různých platformách mohou být pro tyto technologie omezeny nebo znepřístupněny. [3]

2 TECHNOLOGIE PRO VÝVOJ MULTI-PLATFORMNÍCH APLIKACÍ

V současné době se stává stále běžnějším požadavkem vytvořit aplikaci, která by fungovala na více platformách. To znamená, že ji bude možné používat jak na desktopových, tak na mobilních zařízeních, a to bez ohledu na operační systém, který je nainstalován. Multi-platformní aplikace tak zajišťují větší dostupnost aplikací pro uživatele a jsou pro vývojáře často efektivnější, protože nemusí vytvářet různé verze aplikací pro každou platformu zvlášť. Pro jejich vývoj existuje mnoho technologií a každá z nich má své silné a slabé stránky. Vývojář by tedy měl zvážit své potřeby a preference před výběrem správné technologie pro svůj projekt.

2.1 React Native

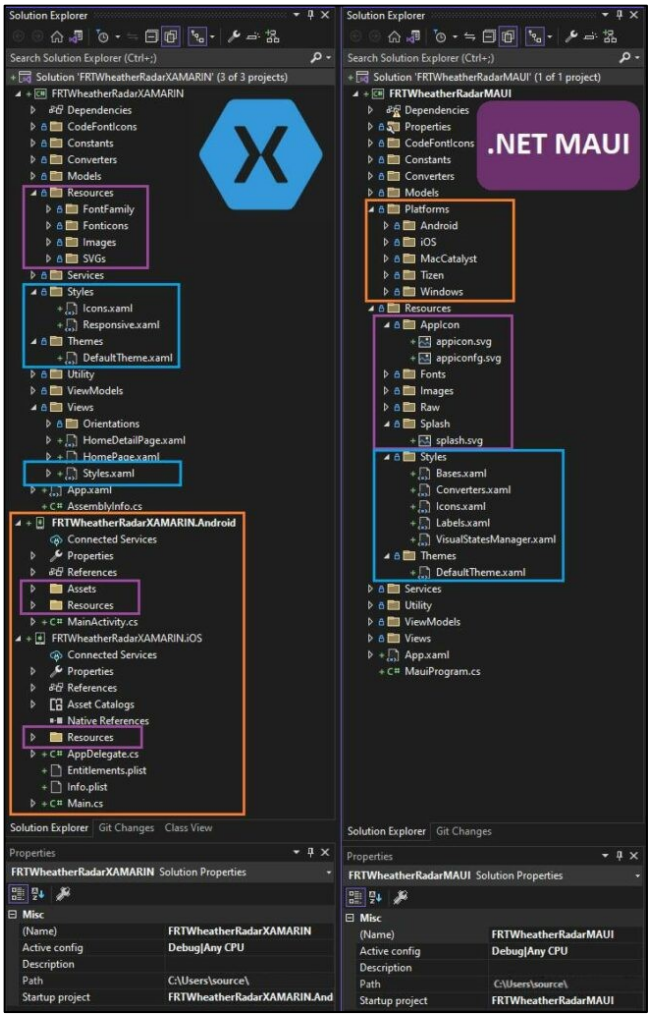
React Native je framework, díky kterému lze používat JavaScript a React pro vývoj aplikací na operačních systémech iOS a Android, které vypadají a fungují jako nativní aplikace. Umožňuje také vývojářům sdílet kód mezi platformami a snižuje tak čas potřebný na vývoj aplikace. React Native také poskytuje rozsáhlou knihovnu komponent, které jsou navrženy tak, aby fungovaly stejně na všech operačních systémech, což usnadňuje vytváření aplikací s konzistentním vzhledem a chováním. [4]

2.2 Xamarin

Xamarin je dalším frameworkem, který umožňuje vývojářům vytvářet multi-platformní aplikace, a to pro iOS, Android a Windows pomocí programovacího jazyka C#. Xamarin také umožňuje vývojářům sdílet kód mezi platformami. A jelikož se jedná o produkt od společnosti Microsoft poskytuje tak rozsáhlou sbírku knihoven a vývojových nástrojů, které umožňují přístup k nativním funkcím platformy. Pro každou platformu je však nutné vytvořit nový projekt a k němu i uživatelské rozhraní, ty však sdílí stejnou logiku aplikace. [5;6]

2.3 .NET MAUI

.NET MAUI (Multi-platform App UI) je novou technologií společnosti Microsoft, která umožňuje vývojářům vytvářet nativní aplikace pro různé platformy, jako jsou například Windows, iOS a Android, pomocí jednoho kódu. Jedná se o nástupce již existující technologie Xamarin. Na rozdíl od Xamarinu nově přidává podporu pro Windows, macOS a Linux. Další výhodou oproti předchozí technologii je jednotná tvorba uživatelského rozhraní, kdy není potřeba pro každou platformu vytvářet nový projekt, stačí vytvořit pouze jednou a o vše ostatní se postará .NET MAUI (viz Obrázek 1). [7]



Obrázek 1 Xamarin vs .NET MAUI [6]

Jednou z největších výhod .NET MAUI je, že umožňuje vývojářům vytvářet aplikace pro více platforem současně, a to včetně uživatelského rozhraní, tím tak mohou snížit čas i náklady potřebné na vývoj aplikací. Nemusí se totiž starat o to, aby vyvíjeli samostatnou aplikaci pro každou platformu, ale mohou využít jednoho kódu pro všechny. Dále také nabízí vynikající výkon a zároveň zajišťuje vysokou míru kompatibility s různými platformami.

To znamená, že aplikace vytvořené s touto technologií budou fungovat stejně dobře na jakékoli platformě, na které budou spuštěny. Kromě toho .NET MAUI poskytuje jednoduchý a intuitivní vývojový proces. Vývojáři totiž mohou využívat své znalosti jazyka C# a vývojového prostředí Visual Studio. [6;7]

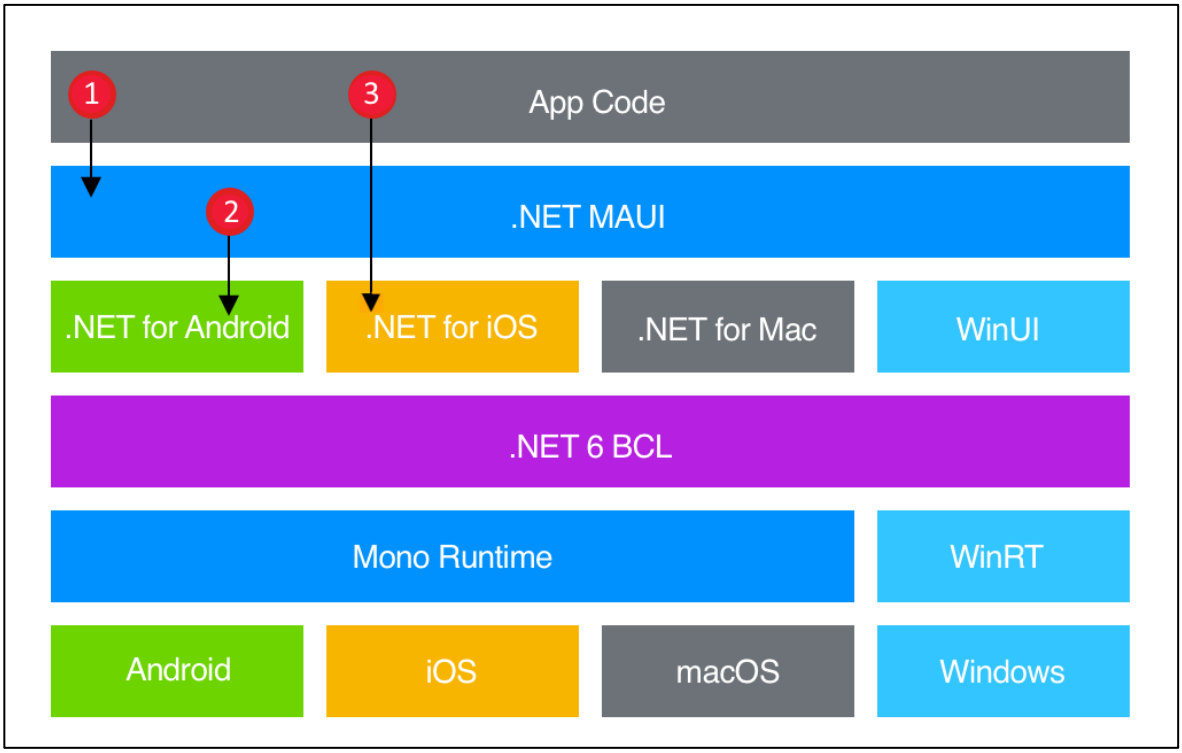
3 .NET MAUI

Hlavním cílem .NET MAUI je zjednodušit multi-platformní vývoj aplikací tak, že co nejvíc kódu, ať už z logiky aplikace nebo uživatelského rozhraní, chce implementovat v jedné code-base, tedy kódu projektu.

3.1 Sada technologií

.NET MAUI poskytuje řadu frameworků pro specifickou platformu, jako jsou například .NET pro Android, .NET pro iOS (a iPadOS), .NET pro Mac a WinUI 3 (s využitím Windows App SDK). Tyto frameworky mají přístup ke stejné knihovně základních tříd (BCL – Base Class Library) v rozhraní .NET 6, která umožňuje aplikacím fungovat na různých typech zařízení a sdílet stejnou logiku aplikace (business logiku). [8]

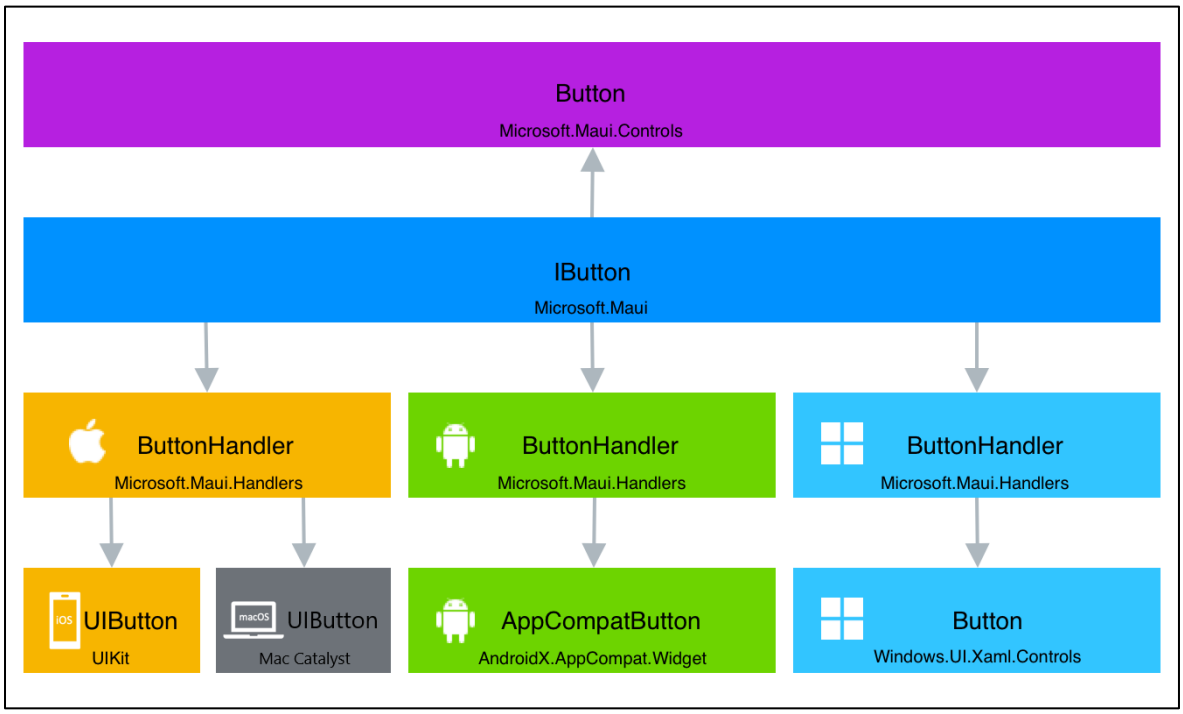
Uživatelské rozraní lze vytvořit pro každou platformu zvlášť pomocí výše zmíněných frameworků, to však vyžaduje udržovat kód pro každé zařízení zvlášť. .NET MAUI ale poskytuje jeden framework pro vytvoření jak mobilních, tak desktopových aplikací. Uživatelské rozhraní tedy může být napsáno jen na jednom místě za použití frameworku poskytnutého od .NET MAUI (znázorněného pod šipkou číslo 1) a ten se pak postará o převedení kódu pro příslušnou platformu (šipka číslo 2). Pokud by však bylo potřeba zavést funkci specifickou pouze pro určitou platformu, tak z daného frameworku lze provolat metody, které by tuto funkci zajišťovaly (šipka číslo 3). [8;9]



Obrázek 2 Struktura .NET MAUI [9]

3.2 Generování kódu

Generování nativního kódu pro cílové zařízení zastřešují takzvané „handlery“, které jsou specifikovány zvlášť pro každou platformu. Například pokud budeme chtít spustit mobilní aplikaci se systémem Android, tak daný handler namapuje kód na příslušný element, který by byl v tomto případě AppCompatButton (viz Obrázek 3). [9]



Obrázek 3 Popis generování kódu pro jednotlivé platformy [9]

4 JAZYK C#

V 90. letech minulého století se společnost Microsoft začala zajímat o vývoj jazyka, který by umožnil vývojářům vytvářet aplikace pro novou generaci operačního systému Windows. V roce 2002 byl jazyk C# následně představen spolu s platformou .NET Framework a od té doby se stal jedním z nejpopulárnějších jazyků pro vývoj aplikací na platformách od společnosti Microsoft. V průběhu let se však vyvíjel a rozšiřoval své možnosti, aby mohl lépe vyhovovat potřebám vývojářů a novým trendům v oboru vývoje softwaru. [10]

C# je vysokoúrovňový programovací jazyk, což znamená, že je pro vývojáře snadný na použití a umožňuje jim vytvářet složité aplikace bez nutnosti znát podrobnosti o tom, jak se aplikace fyzicky vykonávají na počítači. Takový jazyk je pro člověka lépe čitelný, oddaluje se totiž od strojového kódu svou vysokou mírou abstrakce. C# je vhodný pro tvoru různých typů aplikací, jako jsou desktopové aplikace, webové aplikace, mobilní aplikace nebo hry. C# se však v posledních letech stává stále populárnějším jazykem i pro cloudový vývoj. [11]

Tento programovací jazyk má také silnou a aktivní komunitu vývojářů, kteří se pravidelně zapojují na různých fórech do diskusí nebo se i potenciálně podílí na opravách či rozšířeních pro různé technologie. Tato rozsáhlá komunita je ideální z hlediska snadné dostupnosti k nespočtu zdrojů informací a podpory.

4.1 Syntaxe

Konstrukce jazyka C# zahrnuje několik základních stavebních prvků, které umožňují programátorům psát výkonné a efektivní kódy. Mezi takové se řadí například datové typy, proměnné, operátory, podmínky, cykly a funkce.

4.1.1 Datové typy

Datové typy jsou základní stavební jednotkou v jazyce C# a slouží k definici typu hodnoty, kterou může uchovávat proměnná. C# podporuje základní datové typy, jako jsou celočíselné hodnoty, reálná čísla, řetězce, booleany a další. [11]

4.1.2 Proměnné

Proměnné jsou v jazyce C# používány k uchování hodnot. Datový typ proměnné může být specificky určený nebo jej definuje až právě přiřazená hodnota. Tyto hodnoty mohou

nabývat různých datových typů a mohou být přiřazeny proměnným pomocí přiřazovacího operátoru "=". [12]

4.1.3 Operátory

Operátory jsou používány k manipulaci s hodnotami v jazyce C#. Tyto operátory zahrnují aritmetické operátory (jako jsou sčítání, odčítání, násobení a dělení), porovnávací operátory (např. rovná se, větší než, menší než) a logické operátory (např. AND, OR). Ty jsou pak v programovacím jazyce reprezentovány jako znaky například +, -, = a tak dále. [12]

4.1.4 Podmínky a cykly

Podmínky a cykly jsou konstrukce, které umožňují vytvářet větvení v kódu a vykonávat určité jeho části opakovaně. Podmínky se používají k vyhodnocení určitého výroku a provedení určitých akcí v závislosti na výsledku. Cykly umožňují opakovat určitý kód, dokud je splněna daná podmínka.

4.1.5 Funkce

Funkce jsou bloky kódu, které mohou být volány z jiných částí programu. Funkce mohou mít vstupní parametry, vracet výstupní hodnoty a mohou být definovány v rámci tříd nebo mimo ně.

4.1.6 Rozdíly syntaxe pro .NET 6

S každou novou verzí platformy .NET přicházejí novinky ovlivňující i syntaxi jazyka C# a s přechodem na .NET 6 přišly některé změny syntaxe, které mají zjednodušit vývoj a zlepšit čitelnost kódu.

Jednou z hlavních novinek jsou tzv. „*Top-level statements*“, což umožňuje programátorům psát kód bez nutnosti definovat třídy a metody na úplném začátku programu. Syntaxe těchto příkazů se odlišuje od běžné syntaxe C#, jelikož nedochází ke specifikaci názvu třídy nebo metody. Tato nová konstrukce umožňuje zjednodušit tvorbu menších aplikací, zatímco u větších projektů se stále doporučuje tradiční přístup s explicitní definicí tříd a metod. Další změnou syntaxe v .NET 6 je zavedení zkrácené syntaxe pro inicializaci položek kolekcí pomocí indexů (nalevo zkrácená, napravo původní syntaxe). [13]

```
string[] daysOfWeek =
{
    [0] = "Monday",
    [1] = "Tuesday",
    [2] = "Wednesday",
    [3] = "Thursday",
    [4] = "Friday",
    [5] = "Saturday",
    [6] = "Sunday"
};
```

```
string[] daysOfWeek = new string[7];
daysOfWeek[0] = "Monday";
daysOfWeek[1] = "Tuesday";
daysOfWeek[2] = "Wednesday";
daysOfWeek[3] = "Thursday";
daysOfWeek[4] = "Friday";
daysOfWeek[5] = "Saturday";
daysOfWeek[6] = "Sunday";
```

Zdrojový kód 1 Ukázka rozdílné sytaxe

```
using System;

namespace MyApp // Note: actual namespace depends on the project name.
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Zdrojový kód 2 Příklad konzolové aplikace pro starší verze .NET

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

Zdrojový kód 3 Příklad konzolové aplikace pro .NET 5 a výše

4.2 Výhody a použití

Jedním z hlavních důvodů, proč zvolit C# oproti jiným jazykům, je jeho schopnost integrovat se s platformou .NET. Jedná se o velmi výkonnou platformu pro vývoj aplikací, která umožňuje programátorům vytvářet sofistikované aplikace pro Windows, webové aplikace a mnoho dalšího. V jazyce C# lze využívat všechny funkce této platformy a vytvářet tak aplikace, které jsou optimalizovány pro výkon, stabilitu a bezpečnost. [14]

Dalším důvodem pro zvolení C# je jeho jednoduchá syntaxe. Je velmi čitelná a intuitivní, usnadňuje tak práci programátorům a zvyšuje produktivitu. C# je také silně typovaný jazyk,

což znamená, že musíte přesně specifikovat typy dat, které chcete použít. To následně zvyšuje bezpečnost kódu tím, že minimalizuje možnosti chyb a zajišťuje, že daná aplikace funguje tak, jak by měla. C# také nabízí další různé funkce, jako je „*garbage collector*“, který automaticky „uklízí“ nepoužívané objekty, což snižuje riziko úniku paměti (*memory leak*) a zvyšuje stabilitu aplikace. Dále pak například asynchronní funkce, které se nejčastěji používají pro zlepšení odezvy a dosažením tak plynulejšího chodu aplikace. [15]

Díky velké popularitě toho jazyka vzniklo a nadále i vzniká obrovské množství nových rozšíření, například ve formě NuGet balíčků nebo knihoven, které značně usnadňují a zpříjemňují práci.

Klíčové vlastnosti by se tedy daly shrnout do následujících bodů:

- **Rozmanitost** – vývoj různých typů projektů (desktopové, konzolové nebo mobilní aplikace)
- **Multiplatformnost** – spustitelnost na různých operačních systémech
- **Flexibilita** – ideální pro vývoj jak malých, tak i velkých projektů

4.3 Vývojové prostředí a nástroje

4.3.1 Microsoft Visual Studio

Microsoft Visual Studio je nejrozšířenějším vývojovým prostředím pro vývoj aplikací v C#, které je k dispozici v několika verzích a variantách.

Visual Studio nabízí široké spektrum funkcí a nástrojů pro vývoj a ladění aplikací v C#, jako jsou například:

- IntelliSense – funkce, která pomáhá s rychlejším psaním kódu, snadnějším výběrem funkcí a metod a případně i s refaktoringem
- Debugger – umožňuje ladit kód v reálném čase a najít tak chyby v aplikaci
- Integrace s Git a dalšími verzovacími systémy
- Možnost využití různých projektových šablon pro různé typy aplikací [16]

4.3.2 Další nástroje a knihovny pro vývoj v C#

Kromě Visual Studia existuje i mnoho dalších nástrojů a knihoven, které lze při programování v C# využít a mezi ty se řadí například:

- **.NET Core** – multiplatformní framework pro vývoj různých typů aplikací
- **ASP.NET** – webový framework, který umožňuje vývoj webových aplikací
- **Entity Framework** – knihovna pro práci s databázemi

4.3.3 Možnosti vývoje aplikací s využitím cloudových technologií

V současné době se stále více aplikací vyvíjí s využitím cloudových technologií, které umožňují vytvářet, provozovat a spravovat aplikace na cloudových serverech, to přináší řadu výhod jako jsou například:

- **Škálovatelnost aplikací** – mohou být snadno rozšířeny podle aktuálních potřeb, bez nutnosti spravovat vlastní infrastrukturu
- **Vysoká dostupnost** – servery, zajišťují vysokou dostupnost a odolnost proti výpadkům
- **Bezpečnost dat** – data jsou ukládána na cloudových serverech s vysokou úrovní bezpečnosti a šifrování

5 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

Objektově orientované programování (OOP) je přístup k programování, který se spíše zaměřuje na objekty a jejich interakce než na logiku, nutnou pro jejich manipulaci. V posledních letech se tento přístup stal velmi populárním, protože umožňuje vývojářům efektivně vytvářet sofistikované aplikace. Tento způsob vývoje je vhodný i pro práci ve vícečlenných týmech, kdy na jednom projektu pracuje vícero lidí, díky jednotné úpravě je totiž kód lehce čitelný a pochopitelný. [17]

5.1 Základní prvky OOP

Základními kameny OOP jsou třídy a objekty. Třída je návrhový vzor pro objekt a obsahuje definici atributů a metod, které jsou společné pro všechny instance této třídy. Objekt je konkrétní instance třídy a obsahuje data (atributy) a funkce (metody) specifické pro tuto instanci. Nejčastějším příkladem bývají auta, kdy máme obecnou třídu „Auto“ a jednotlivé značky aut (Škoda, Renault…) pak představují objekty. [18]

5.1.1 Dědičnost

Dědičnost umožňuje vytvářet nové třídy na základě stávajících tříd. Nová třída (potomek) zdědí všechny atributy a metody nadřazené třídy (rodič) a může přidávat nebo přepisovat metody dle potřeby.

5.1.2 Polymorfismus

Polymorfismus umožňuje používat stejný název metody pro různé třídy a objekty a v závislosti na kontextu se vybírá správná implementace této metody. Takto mohou různé typy objektů procházet stejným rozhraním. [19]

5.1.3 Abstrakce a zapouzdření

Abstrakce umožňuje programátorům skrýt podrobnosti implementace a zaměřit se na základní vlastnosti tříd a objektů. Zapouzdření umožňuje skrýt interní stav objektu a poskytnout rozhraní pro použití třídy a objektu. [20]

5.2 Návrhové vzory

Návrhové vzory jsou osvědčené postupy a řešení, které se v průběhu let ukázaly jako efektivní pro řešení opakujících se problémů. Tyto vzory přináší řadu výhod, jako například zvýšení efektivity a rychlosti vývoje, zlepšení kvality kódu a zjednodušení údržby aplikace. Návrhové vzory se vyskytují v různých oblastech objektově orientovaného programování, jako jsou například návrh uživatelského rozhraní, architektura aplikací a vzory pro práci s databázemi. Každý vzor má své vlastní jméno, popis a použití, jako například Factory, Singleton nebo Observer. [21]

5.2.1 Factory

Factory umožňuje vytvářet objekty bez nutnosti specifikovat konkrétní třídu, která se použije pro vytvoření objektu. Díky tomu je snadné měnit konkrétní implementaci bez nutnosti upravovat kód na více místech.

Často se používá v situacích, kdy je třeba vytvářet objekty s podobným chováním, ale s odlišnými implementacemi. Například, pokud potřebujeme vytvářet různé typy aut, můžeme použít tento vzor, abychom zajistili, že se správná třída použije pro každý typ auta. Není tedy potřeba mít specifický kód pro každý typ auta, ale pouze jednu Factory třídu, která se stará o vytváření objektů. [21]

5.2.2 Singleton

Hlavním účelem Singletonu je zajistit, že existuje pouze jedna instance dané třídy v rámci celé aplikace a umožnit přístup k této instanci z jakéhokoli místa v aplikaci. To může být užitečné například v situacích, kdy máme třídu, která se stará o nějakou globální funkcionalitu, jako například nastavení aplikace, přístup k databázi nebo logování. Singleton však může být náchylný k problémům s vícevláknovostí a jeho špatné použití může vést k velmi komplikovanému a špatně udržovatelnému kódu. Je proto důležité vždy pečlivě zvažovat, zda je zrovna tento návrhový vzor je tou nejlepší volbou pro konkrétní aplikaci. [21;22]

5.2.3 Facade

Facade je jeden z nejčastěji používaných návrhových vzorů v moderním programování. V případě komplexnějších systémů poskytuje jednotné rozhraní pro všechny jeho podsystémy tak, aby bylo snadné s nimi pracovat.

Například, pokud máme aplikaci, která pracuje s databází, webovým rozhraním a různými API, můžeme použít Facade, abychom zajistili, že všechny tyto podsystémy budou mít rozhraní, ke kterému by se dalo jednoduše přistoupit v rámci celé aplikace. Může být také velmi užitečný pro zachování bezpečnosti aplikace, protože skrývá složitost a detaily podsystémů před vnějším světem. [22]

5.3 Výhody a nevýhody

OOP je často srovnáváno s jinými paradigmaty programování, jako jsou procedurální programování nebo funkcionální programování. Každé paradigma má své výhody či nevýhody, volba správného paradigmatu však závisí na konkrétní situaci. Používá se v široké škále aplikací, včetně desktopových, webových nebo mobilních aplikací a her. Dá se tedy říct, že je obzvláště užitečné pro velké projekty, kde je důležité udržovat kód organizovaný a snadno rozšiřitelný. Další výhodou OOP je znovupoužití kódu. Třídy a objekty lze snadno znovu použít v různých aplikacích a projektech, což snižuje množství kódu, který musí být napsán. [17;23]

OOP však má i své nevýhody, může být pomalejší než jiná paradigmata, jako je například procedurální programování. To je způsobeno tím, že objekty musí být vytvářeny a zpracovávány při každé operaci, což může být problém v aplikacích, které vyžadují vysokou rychlost zpracování dat. Další nevýhodou je, že správa paměti a životního cyklu objektů může být složitá. Programátoři musí být opatrní, aby nepoužili příliš mnoho paměti nebo nevytvářeli zbytečné objekty, které by mohly způsobit problémy s výkonem a stabilitou aplikace. [23]

6 REVERZNÍ INŽENÝRSTVÍ

Reverzní inženýrství je proces, který umožňuje analyzovat a dekonstruovat existující produkty, systémy nebo technologie a získat z nich informace a znalosti, které mohou být využity k vylepšení nebo nahrazení původního řešení. Reverzní inženýrství se často používá v různých odvětvích, jako jsou automobilový a letecký průmysl, ale také v oblasti informačních technologií, kde je nutné porozumět fungování součástek a komponent nebo aplikací, aby bylo možné vylepšit výkon nebo snížit náklady potřebné pro vývoj.

Důležitost reverzního inženýrství spočívá v tom, že umožňuje inovaci a vylepšení stávajících produktů a technologií, což může vést k větší konkurenceschopnosti. Také umožňuje lépe porozumět fungování těchto produktů a technologií, a umožnit tak rozvoj nových výrobků a služeb. Vyžaduje však speciální znalosti a nástroje, včetně hardwarových a softwarových nástrojů pro analýzu a dekompilaci kódu. V dnešní době je však stále více dostupných automatizovaných nástrojů, které umožňují rychlejší a efektivnější reverzní inženýrství. [24]

Reverzní inženýrství je také často vnímáno jako něco špatného. V některých případech se totiž může jednat o nelegální činnost, zejména pokud se týká ochrany autorských práv. Proto je důležité dodržovat platné zákony společně s příslušnými dohodami a licencemi.

6.1 Nástroje a technologie

Aby bylo možné efektivně provádět reverzní inženýrství, je dobré používat různé nástroje a technologie, které tento proces umožňují nebo jej do jisté míry usnadňují.

6.1.1 Dekompilátory

Dekompilátory umožňují převést strojový kód zpět na zdrojový kód. Tyto nástroje jsou velmi užitečné pro analýzu softwarových aplikací, zejména pro zjištění způsobu, jakým byly napsány a jak fungují. [24]

6.1.2 Emulátory

Emulátory jsou nástroje, které umožňují simulovat hardware nebo software na jiném systému. Využívají se hlavně pro testování a analýzu softwarových aplikací. Tímto způsobem je například možné si spustit takový emulátor pro mobilní zařízení na desktopovém zařízení.

6.1.3 Debugger

Díky debuggeru můžeme analyzovat chování programu a zjišťovat, jakým způsobem se program chová. Debugger umožňuje krokovat aplikaci, tedy spouštět jednotlivé části kódu krok za krokem a sledovat tak hodnoty proměnných a kontrolovat stav programu v různých bodech. Tento nástroj je velmi užitečný pro analýzu a nalezení chyb v programu nebo pro pochopení, jak jednotlivé metody fungují. [24]

6.2 Metody a postupy

Existují osvědčené metody a postupy uplatňované při procesu reverzního inženýrství, které lze využít k rychlejšímu dosažení výsledků.

6.2.1 Statická analýza

Statická analýza je technika, která umožňuje analýzu kódu bez jeho spuštění. Pomocí nástrojů nazývaných statické analyzátory se kód projde a zanalyzuje. Statická analýza identifikuje různé prvky kódu a hledá různé druhy chyb v kódu. Statický analyzátor také zjišťuje závislosti mezi jednotlivými prvky kódu. Výsledkem statické analýzy jsou reporty a grafy, které umožňují programátorům a inženýrům lépe porozumět kódu a najít různé problémy a chyby. [25]

6.2.2 Dynamická analýza

Dynamická analýza je metoda, která umožňuje analyzovat chování programu za běhu. Pomocí nástrojů pro dynamickou analýzu lze sledovat různé aspekty programu, jako jsou vstupy, výstupy, alokace paměti a další. Díky této technice lze identifikovat různé problémy v programu, jako jsou chyby přetečení paměti nebo chyby v režimu volání. [25]

6.2.3 Fuzzing

Fuzzing je metoda reverzního inženýrství, která se zaměřuje na generování náhodných vstupů do programu s cílem najít chyby. Provádí se tak, že se záměrně vytváří a odesílají nevalidní nebo neočekávané vstupy do testovacího softwaru s cílem prozkoumat jeho chování a případně vyvolat chybu. Takový proces může být i automatizován a opakován, aby se maximalizovala šance na odhalení chyb a zranitelností. [26]

7 ZABEZPEČENÍ DOKUMENTŮ

Zabezpečení dokumentů je dnes nezbytností v každé organizaci nebo podniku. Důvodem je skutečnost, že dokumenty obsahují informace, které jsou pro podnikání velmi důležité a často citlivé. Tyto informace mohou být finančního, obchodního, osobního nebo jinak tajného charakteru a pokud nejsou řádně zabezpečeny, mohou být zneužity nebo ukradeny.

7.1 Kybernetické hrozby

7.1.1 Malware

Nejčastější hrozby pro dokumenty jsou většinou spojeny s počítačovou technologií a jedním z nejčastějších způsobů, jakým jsou dokumenty ohroženy, je malware. Ten může být přenesen do počítače pomocí e-mailu, souborů ke stažení nebo dokonce z podvodných či infikovaných webových stránek. Může být navržen tak, aby z dokumentů ukradl citlivé informace nebo v horším případě data nenávratně smazal. [27]

7.1.2 Hackeři

Další hrozbou jsou hackeři. Ti mohou vniknout do počítače nebo serveru a získat tak přístup k citlivým dokumentům. Pokud tyto dokumenty obsahují právě ony citlivé informace, mohou být použity k vydírání nebo k podvodům.

7.1.3 Phishing

Phishingové útoky jsou dalším způsobem, jakým mohou být dokumenty ohroženy. Takové útoky jsou nejčastěji ve formě e-mailů, které vypadají jako legitimní e-maily od známého zdroje, ale ve skutečnosti jsou odesílány z falešné adresy. Takové e-maily většinou obsahují škodlivé odkazy nebo přílohy, které mohou být použity k odcizení citlivých informací. [27]

7.1.4 Lidský faktor

Je také důležité chránit dokumenty před interními hrozbami, jako jsou zaměstnanci nebo lidé, kteří mají přístup k citlivým dokumentům. Zaměstnanci totiž mohou nevědomky či úmyslně zveřejnit citlivé informace nebo integrovat škodlivý software, který citlivá data dokáže zneužít. Proto by měly být dokumenty nějakým způsobem chráněny a přístup k nim by měl být omezen pouze na osoby, které k nim mají oprávnění.

7.1.5 Obecná bezpečnostní doporučení

V závislosti na typu dokumentu a jeho citlivosti je třeba použít různé zabezpečovací opatření. Například dokumenty s citlivými osobními údaji by měly být chráněny šifrováním a měly by být uloženy na zabezpečeném serveru s omezeným přístupem. Důležité dokumenty obsahující obchodní informace by měly být chráněny prostřednictvím digitálních podpisů a přístupu pouze pro vybrané zaměstnance. Kromě toho je také důležité zabezpečit fyzickou kopii dokumentů, pokud jsou uloženy v kanceláři nebo jiném veřejném prostoru. Tyto dokumenty by měly být uzamčeny v trezoru nebo skříni a přístup k nim by měl být omezen jen na osoby, které mají oprávnění. [28]

Existuje řada opatření, která mohou organizace přijmout k ochraně svých dokumentů. Taková opatření například zahrnují:

1. Vytvoření zabezpečeného prostředí pro ukládání dokumentů. Takové prostředí by mělo být chráněno přístupovými kódy nebo hesly a mělo by být omezeno pouze na zaměstnance, kteří mají oprávnění k přístupu.
2. Použití silných hesel a šifrování pro chránění dokumentů před neoprávněným přístupem.
3. Omezení počtu lidí, kteří mají přístup k citlivým dokumentům.
4. Pravidelné zálohování dokumentů a ukládání záloh na zabezpečeném místě.
5. Monitorování přístupu k dokumentům a provádění pravidelných revizí zabezpečení dokumentů.

7.2 Typy zabezpečení dokumentů

Existuje mnoho různých typů zabezpečení, které jsou k dispozici. Při volbě, který typ by měl být zrovna použit, zohledňujeme hned několik faktorů. Pro klíčové dokumenty s velmi citlivými údaji volíme ta nejspolehlivější řešení, která zaručují největší bezpečnost. Naopak u méně důležitých dokumentů můžeme zvolit běžnější opatření, která i do jisté míry dodávají větší komfort při práci.

7.2.1 Hesla

Hesla jsou nejběžnějším typem zabezpečení dokumentů. Pomocí hesel lze omezit přístup k dokumentům pouze pro osoby, které znají správné heslo a pokud je dostatečně silné a složité, může poskytnout vysokou úroveň zabezpečení. Hesla jsou však často napadávána pomocí různých útoků, krádeží a tak dále, proto je důležité, aby bylo každé heslo dostatečně dlouhé, složité a často se měnilo.

7.2.2 Šifrování

Šifrování je proces převodu dat do nečitelné formy a přístup k datům lze získat pouze s použitím správného klíče. Šifrování může být použito pro celé dokumenty nebo pouze pro jeho určité části. Existuje mnoho různých šifrovacích algoritmů, které mají různé úrovně zabezpečení. Na druhou stranu může být šifrování náročné na implementaci a může zpomalit rychlost přístupu k datům. [29]

7.2.3 Digitální podpisy a certifikáty

Digitální podpisy a certifikáty jsou dalším způsobem, jak zabezpečit dokumenty. Digitální podpis je elektronický záznam, který je připojen k dokumentu, aby prokázal, že dokument pochází od určitého odesílatele a nebyl během přenosu upraven. Certifikáty jsou naopak vydávány příslušnou autoritou pro ověření totožnosti uživatele nebo organizace. Digitální podpisy a certifikáty mohou poskytnout velmi vysokou úroveň zabezpečení, ale je nutné zajistit, aby byly používány správně a aby byly vydávány důvěryhodnými autoritami. [30]

7.2.4 Přístupová práva

Díky přístupovým právům lze omezit přístup k dokumentům pouze na určité osoby. Pomocí přístupových práv může být stanoveno, kdo má právo na zobrazení nebo úpravu dokumentů, případně určitých částí dokumentu. Tato metoda zabezpečení je užitečná, pokud jsou například dokumenty uloženy ve veřejně přístupném adresáři, kde chceme zajistit, že pouze oprávněné osoby mohou provádět úpravy. [31]

7.2.5 Verzování

Zabezpečit dokumenty lze i verzováním, což je metoda, kterou lze uchovávat různé verze dokumentu a zaznamenávat změny provedené na něm provedené. Pokud dojde k chybě nebo úmyslné změně, je možné se vrátit k nějaké z jeho předchozích verzí. Díky tomu lze změny

snadno monitorovat a zajistit tak, že každá úprava dokumentu bude zaznamenána a v budoucnu lehce obnovitelná. [31]

7.3 Šifrovací algoritmy

7.3.1 Asymetrické algoritmy

Asymetrické algoritmy pracují, oproti symetrickým, s párem klíčů (veřejným a soukromým klíčem). Veřejný klíč je distribuován široké veřejnosti, zatímco soukromý klíč zůstává utajen a je pouze v držení příjemce zprávy. Při šifrování se veřejný klíč používá k zašifrování zprávy, kterou lze následně dešifrovat pouze soukromým klíčem příjemce. Mezi nejčastěji používané asymetrické algoritmy se řadí například RSA (Rivest-Shamir-Adleman), který je založen na principu náročnosti faktorizace, nebo ECC (Elliptic Curve Cryptography), jež na druhou stranu využívá vlastností eliptických křivek. [30]

Používání asymetrických algoritmů přináší určité výhody, jako je flexibilita, snadná distribuce veřejných klíčů a možnost komunikace s neznámými partnery. Nicméně, jsou však obecně výpočetně náročnější než symetrické algoritmy, což může ovlivnit jejich výkon a rychlost.

7.3.2 Symetrické algoritmy

Symetrické šifrovací algoritmy jsou obecně rychlejší a efektivnější než asymetrické algoritmy, což je důvod, proč jsou rozšířené a preferované pro většinu šifrovacích aplikací. Na rozdíl od asymetrických algoritmů využívají symetrické šifrovací algoritmy pro šifrování a dešifrování stejný klíč. Principem je aplikovat matematickou operaci na vstupní data pomocí soukromého klíče, čímž vzniká šifrovaná zpráva. Pro dešifrování se použije stejný klíč ale v opačném směru, čímž se původní data obnoví. [32]

Bezpečnost symetrických algoritmů spočívá v důvěrnosti klíče. Pokud by se tedy klíč dostal do rukou nepovolaným osobám, mohlo by to vést k ohrožení celého systému. Pokud jsou použity dostatečně dlouhé, náhodné klíče a jsou správně implementovány, tak tyto algoritmy dokážou poskytnout vysokou míru zabezpečení. [30]

AES (Advanced Encryption Standard)

Je založen na principu substituce a permutace dat, která jsou rozdělená do bloků o pevně dané velikosti na 128 bitů. Klíč poté může nabývat 128, 192 nebo 256 bitů. Právě od zvolené

velikosti klíče jsou pak odvozené i typy AES, které mohou být použity, a to tedy AES128, AES192 nebo AES256.

AES byl navržen tak, aby byl co nejefektivnější a zároveň zaručoval co největší možnou míru bezpečnosti. Používá pevný počet iterací, díky kterému je jeho výkon na různých zařízeních předvídatelný a konzistentní. Na modernějších vícejádrových procesorech může využít i paralelního zpracování dat na, které efektivně využívá výpočetního výkonu a zrychluje tak šifrovací operace. Dalším faktorem je i optimální velikost bloků, která je pevně nastavená na 128 bitů, což je ideální velikost pro rychlou manipulaci s daty. [30;32]

DES (Data Encryption Standard)

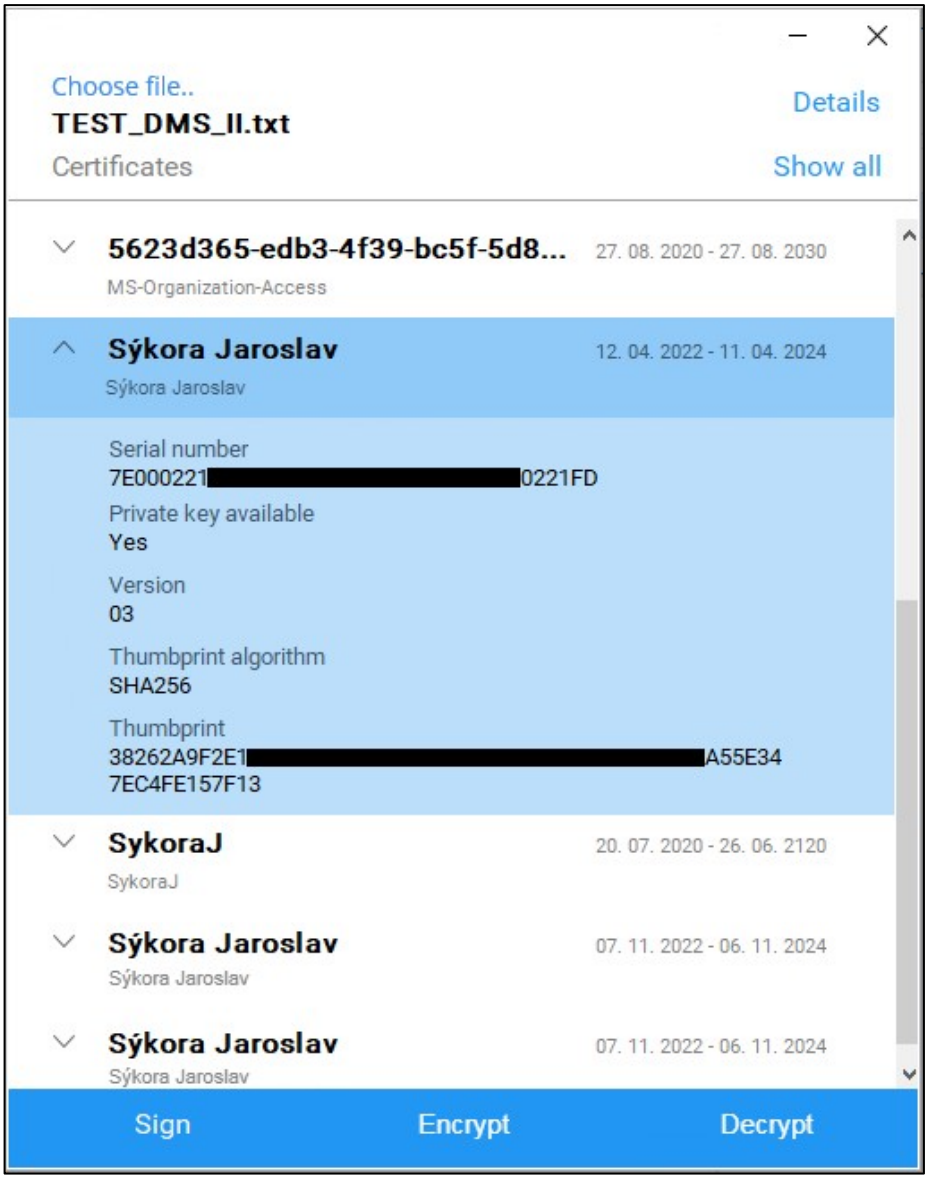
DES je starší symetrický algoritmus, který byl v minulosti standardem. Používá 56bitové klíče pro šifrování a dešifrování dat a je založen na struktuře Feistel Cipher, kde je prostý text rozdělen na dvě poloviny. Avšak v současné době se považuje za zastaralý a nedostačující z hlediska bezpečnosti. Hlavním důvodem je jeho krátká délka klíče, která je náchylná k prolomení pomocí brute-force útoků, při kterém útočník zkouší všechny možné kombinace klíče. Důvodem je rychlý výpočetní výkon moderních počítačů, který těmto útokům umožňuje prolomit DES v přijatelném čase.

Tam kde je nutné zachovat kompatibilitu se zastaralým algoritmem DES je využívá modernějšího řešení 3DES. Ten funguje tak, že aplikuje algoritmus DES třikrát za sebou a velikost klíče je tedy 168bitů a zajišťuje tak vyšší míru bezpečnosti než původní DES, ale je pomalejší a vyžaduje větší výpočetní výkon. [30;32]

II. PRAKTICKÁ ČÁST

8 PŘEDSTAVENÍ PŮVODNÍ APLIKACE

Crypto Native App (CNA) je nativní aplikace společnosti TESCO SW a.s., která umožňuje vytváření elektronických podpisů nad dokumenty a případně i jejich šifrování a dešifrování.



Obrázek 4 Původní aplikace

8.1 Motivace

Dosavadní stav aplikace, tak jak ji používají zaměstnanci nebo zákazníci, je pro běžnou potřebu dostačující, v důsledku neustálého vývoje je však aplikaci nutné často upravovat tak, aby vyhovovala novým požadavkům.

Z důvodů použití knihovny SecureBlackbox a jejím tehdejším problémům v prostředí mono, ale i dalším faktorům byl pro vývoj vybrán jazyk FreePascal s vývojovým prostředím Lazarus. Vzhledem k tomu se však jakákoliv práce na tomto projektu, obzvláště pro nové zaměstnance, kteří se pohybují spíše v modernějších prostředích, stala značně nekomfortní.

S příchodem aktualizace v podobě rozšíření knihovny SecureBlackbox i pro .NET 6 a s novou technologií od společnosti Microsoft, kterou je framework .NET MAUI, se naskytla ideální příležitost pro rekonstrukci tohoto nástroje a zasadit jej tak do modernějšího prostředí.

8.2 Funkce aplikace

Crypto Native App pracuje s certifikáty skrze knihovnu SecureBlackbox, která poskytuje rozhraní pro práci se systémovým uložištěm (Win, Mac). Uživatel má možnost nahrát soubor a na něm následně provést kryptografické operace (podpis, šifrování, dešifrování). Další možnosti aplikace jsou pak zobrazení hash hodnoty daného souboru nebo zobrazení validních certifikátů.

8.3 Principy

Při vývoji byl kladen důraz především na principy objektově orientovaného programování. Za zmínky pak stojí i využití dependency injection pro registraci a práci s poskytovateli podpisů a dešifrování (providery). Coding conventions jsou v aplikaci trochu specifické. Vzhledem k velkému využití knihovny SecureBlackbox byla snaha dodržet stejné konvence, jaké jsou použity v knihovně, což zahrnuje následující pravidla:

- Názvy rozhraní vždy začínají písmenem I (interface)
- Názvy tříd začínají písmenem T (type). Pole pak začínají vždy písmenem F (field)
- Proměnné, funkce, procedury aj. vždy začínají velkým písmenem

8.4 Architektura

Aplikace je v zjednodušeném pohledu složena ze dvou částí, kdy každá z nich má jasné rozdělení kompetencí.

1. Grafické uživatelské rozhraní (GUI) se stará o interakci s fyzickým uživatelem, umožňuje mu vybírat certifikáty, potvrzovat požadované operace (např. podepsání souboru) a případně zjistit detailní informace o datech (např. hash podepisovaného souboru).
2. Logická část aplikace obsahuje management certifikátů (např. načítání) a obsluhu daných kryptografických operací (např. podepisování).

9 NÁVRH NOVÉ APLIKACE

Při návrhu byl kladen důraz na podobnost s původní aplikací. Hlavní motivací pak je dokázat, že aplikaci lze převést do modernějšího prostředí a usnadnit tak budoucí práci na projektu.

9.1 Požadavky na aplikaci

1. Výběr vhodných technologií pro vývoj multiplatformních aplikací
2. Využít knihovny SecureBlackBox, která bude zastřešovat jednotlivé kryptografické operace
3. Umožnit uživateli nahrát soubor z podporované množiny souborů
4. Umožnit nahraný soubor podepsat, šifrovat nebo dešifrovat
5. Nahrát dostupné certifikáty ze systémového úložiště s možností zobrazit všechny nebo jen validní
6. Ošetřit jednotlivé operace, aby zohledňovali uvedené podmínky (viz níže)
7. Aplikace bude umožňovat nahrání pouze určitých typů souborů, konkrétně tedy:
   - PDF dokumenty
   - XML dokumenty
   - Vybrané dokumenty MS office (word, excel)

9.2 Podmínky

Pro podepsání dokumentů je nutné vybrat validní certifikát, který bude následně pro podpis použit. Pokud nebude vybrán, tak aplikace zobrazí uživateli chybovou hlášku o absenci certifikátu.

Při šifrování a dešifrování jsou podmínky mírně odlišné. Dokumenty totiž mohou být zašifrovány nebo dešifrovány i pouhým zadáním hesla. Podpora těchto operací za použití certifikátu je však pouze pro PDF dokumenty, ostatní je pak možné šifrovat či dešifrovat pouze pomocí hesla.

10 VÝVOJ APLIKACE

10.1 Vybrané technologie

Pro vývoj této aplikace byla zvolena moderní technologie .NET MAUI, která je podrobněji popsána v samostatné kapitole (viz Kapitola 3 .NET MAUI).

Důvod pro tuto volbu byl jednoznačný. Jelikož je MAUI produktem společnosti Microsoft je integrovaná do jejich aktuálního vývojové prostředí, kterým je Visual Studio, a dále využívá i programovacího jazyka C#. Ten má velké zastoupení na trhu a je využíván vývojáři po celém světě. Tím se velké množství vývojářů dokáže lehce zorientovat ve struktuře aplikace a porozumět psanému kódu.

Řešení zmíněných kryptografických operací není jednoduchou záležitostí a aby byla zajištěna co nejvyšší míra bezpečnosti byla zvolena knihovna SecureBlackbox jako určitá známka kvality. Ta nabízí mnoho komponent, které mohou být i v budoucnu použity pro rozšíření aplikace.

10.2 Implementace SecureBlackbox

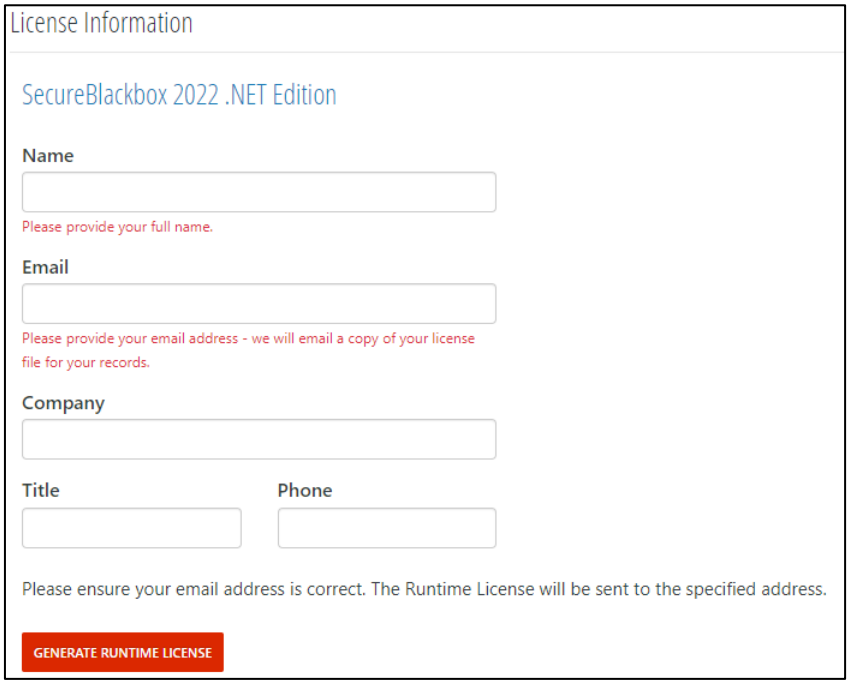
Jelikož je tato knihovna komerčním produktem, je potřeba k jejímu používání licence. Pro vývoj byla využita zkušební licence, o kterou si mohou vývojáři zažádat na stránkách společnosti nSoftware.

10.2.1 Žádost o licenci

Pro vygenerování zkušební licence je potřeba podat žádost na oficiálních stránkách nSoftware. K odkazu se jde proklikat skrze produktovou dokumentaci v části „*Introduction*“ pod kapitolou „*Trial licensing*“ nebo skrze tento odkaz:

https://www.nsoftware.com/lic/?prod=SBNHA&a=trial&grtk=true

Pro žádost je potřebné vyplnit pouze jméno a email a po zadání požadovaných údajů bude licenční klíč zaslán na zadanou adresu (viz Obrázek 5).



Obrázek 5 Formulář žádosti o licenci

10.2.2 Používání v aplikaci

Knihovnu lze stáhnout dvěma způsoby, a to buď skrze .EXE instalátor nebo stažením v podobě NuGet balíčku.

V případě instalátoru si uživatel může zvolit místo uložení a případně i možnost stáhnutí demo aplikací, které slouží jako ukázky použití určitých komponent. V samotné aplikaci je pak nutné odkazovat na tuto knihovnu skrze příslušné .dll soubory, jež slouží jako rozšíření k danému projektu. Zde nastává jeden z prvních problémů, který je blíže popsaný v kapitole o problémech při vývoji. Odkazovat na danou knihovnu je možné kliknutím pravého tlačítka myši na daný projekt, z nabídky kliknout na *„přidat referenci“* a pak už jen vybrat konkrétní .dll. V tomto případě se odkazuje na soubor pro prostředí .NET 6 v podsložce umístění instalace knihovny.

Pro pohodlnější práci s jednotlivými komponentami SecureBlackboxu byla vytvořena pomocná třída pro počáteční inicializaci každé využívané komponenty. Aby mohla být použita, je potřeba jí nastavit licenční klíč. Kód uvedený níže přijímá jako parametr instanci dané komponenty a do její vlastnosti *„RuntimeLicence“* nastaví daný klíč (viz Zdrojový kód 4).

```
public class SBBUtils
{
  public static string RuntimeLicense => "Zde_uveďte_licenční_klíč";
  public static T CreateSBBComponent<T>(T instance) where T : class
  {
    typeof(T).GetProperty("RuntimeLicense").SetValue(instance,
RuntimeLicense);

    return instance;
  }
}
```

Zdrojový kód 4 Pomocná třída SBBUtils

Za účelem otestování aplikace si tedy bude muset daný uživatel nastavit vlastní vygenerovaný licenční klíč (viz Zdrojový kód 4).

10.3 Práce s certifikáty

Pro načtení certifikátů ze systémového úložiště bylo využito jedné z komponent knihovny SecureBlackbox. Pro lepší vizuální reprezentaci byl vytvořen samostatný model, do kterého se načtou relevantní informace o daném certifikátu, jež se následně propojí s uživatelským rozhraním (viz Zdrojový kód 5).

```
public class CertificateModel
    {
        public string Subject { get; set; }
        public string Issuer { get; set; }
        public string NotBefore { get; set; }
        public string NotAfter { get; set; }
        public Color TextColor { get; set; }
        public string SerialNumber { get; set; }
        public string ThumbPrint { get; set; }
        public bool PrivateKeyAvailable { get; set; }
        public string ThumbPrintAlgorithm { get; set; }
    }
```

Zdrojový kód 5 Ukázka modelu certifikátu

Na samotném začátku je potřeba zaregistrovat Runtime licenci ke komponentě buď pomocí metody *CreateSBBComponent,* nebo přímo na vlastnosti dané komponenty, a následně otevřít systémové úložiště, kde se nachází certifikáty. Pro každý certifikát zvlášť se pak zaznamenávají informace v upraveném nebo stejném tvaru. Informace o certifikátu, tak jak je poskytuje komponenta SecureBlackboxu, nejsou uživatelsky dobře čitelné, proto pro jejich zobrazení byla vytvořena samostatná kolekce využívající výše zmíněného modelu. Jedinou informací, která je v rámci modelu pro uživatelské rozhraní navíc, je *TextColor*.

Ta v aplikaci vizuálně odlišuje validní certifikáty od nevalidních v závislosti na její době expirace (viz Zdrojový kód 6).

```
public List<CertificateModel> LoadCertificates(List<Certificate> storedCertificates)
{
    var certstorage = SBBUtils.CreateSBBComponent(new Certificatestorage());
    certstorage.Open("system://?store=My");
    if (certstorage.Certificates.Count > 0)
    {
        foreach (Certificate cert in certstorage.Certificates)
        {
            storedCertificates.Add(cert);
            CertificateModel certificate = new()
            {
               // Subjekt/vlastník certifikátu
               Subject = cert.Subject,
               // Název CA (Certificate Authority)
               Issuer = cert.Issuer,
               // Certifikát vydán OD
               NotBefore = DateTime.Parse(cert.ValidFrom).ToString("dd. MM. yyyy"),
               // Certifikát vydán DO
               NotAfter = DateTime.Parse(cert.ValidTo).ToString("dd. MM. yyyy"),
               // Barevné odlišení propadlých certifikátů od validních
               TextColor = DateTime.Compare(DateTime.Parse(cert.ValidTo), DateTime.Now)
< 0 ? Color.FromArgb("#ff0000") : Color.FromArgb("#808080"),
               // Sériové číslo certifikátu
               SerialNumber = Convert.ToHexString(cert.SerialNumber),
               // TumbPrint == Fingerprint, SBB to má jinak pojmenované
               ThumbPrint = Convert.ToHexString(cert.Fingerprint),
               // Informace o přítomnosti privátního klíče
               PrivateKeyAvailable = cert.PrivateKeyExists,
               // Algoritmus použitý CA k podepsání certifikátu
               ThumbPrintAlgorithm = cert.SigAlgorithm
            };
            AllCertificates.Add(certificate);
        }
    }
    return AllCertificates;
}
```

Zdrojový kód 6 Metoda starající se o načtení certifikátů

Metoda vrací List certifikátů (*AllCertificates*) s vlastním definovaným typem, tato kolekce certifikátů je pak zobrazena v uživatelském rozhraní. Dále metoda přijímá v parametru druhý List certifikátů (*storedCertificates*), který je na rozdíl od prvně zmíněného listu v nezměněném tvaru, tak jak jej poskytuje knihovna SecureBlackbox. Důvodem existence

druhého listu certifikátů je ten, že komponenty této knihovny mohou pracovat jen s certifikáty poskytnutými z její vlastní komponenty (*Certificatestorage*).

10.4 Služba pro práci s dialogy

Byla vytvořena služba, která zajišťuje stabilní volání metod pro zobrazování dialogů v .NET MAUI. Ačkoli lze tyto metody volat napřímo bez služby, může nastat problém s voláním z jiného než hlavního vlákna. Potom by aplikace mohla havarovat na *„Wrong thread exception“* tedy výjimku vyvolanou voláním metody ze špatného vlákna. To je zde ošetřeno definováním takzvaných *Fire-and-Forget* metod. Ty fungují na principu, že odesílatel odešle zprávu a dále se na ni neohlíží, tedy nevyžaduje odpověď nebo informaci o přijetí. Příjemce následně zprávu přijme a zpracuje ji bez interakce s odesílatelem (viz Zdrojový kód 7).

```
public interface IAlertService
{
    // asynchronní volání (použití s "await" - MUSÍ BÝT NA HLAVNÍM VLÁKNU)
    Task ShowAlertAsync(string title, string message, string cancel = "OK");

    // volání "Fire and forget"
    void ShowAlert(string title, string message, string cancel = "OK");
}

internal class AlertService : IAlertService
{
    // asynchronní volání (použití s "await" - MUSÍ BÝT NA HLAVNÍM VLÁKNU)
    public Task ShowAlertAsync(string title, string message, string cancel = "OK")
    {
    return Application.Current.MainPage.DisplayAlert(title, message, cancel);
    }

    // ----- volání "Fire and forget" -----
    public void ShowAlert(string title, string message, string cancel = "OK")
    {
        Application.Current.MainPage.Dispatcher.Dispatch(async () =>
            await ShowAlertAsync(title, message, cancel)
        );
    }
}
```

Zdrojový kód 7 Metoda obsluhující dialogová okna

10.5 Testování

Aplikace byla manuálně testována, aby ověřila správnost a funkčnost jednotlivých operací. Byl kladen důraz na správnou interakci mezi aplikací a uživatelem při vyvolání chybné či správné akce.

Testována byla také vizuální stránka aplikace. Zde bylo oproti původnímu návrhu pozměněno zobrazení načtených certifikátů, kdy dle mého názoru nedošlo k uživatelskému diskomfortu a uživatel má díky této změně k dispozici veškeré doplňující informace o certifikátu ihned zobrazené.

10.6 Publikace

Posledním krokem při vývoji aplikace je její publikování. V následujících podkapitolách je popsán tento proces pro platformu Windows a macOS, tak jak ji definuje .NET MAUI.

10.6.1 Pro platformu macOS

Distribuce instalačního balíčku pro tuto platformu je možná pouze ze zařízení s macOS. Dále je k tomuto však nutné splnit určité prerekvizity.

Je nutné mít na tomto zařízení nainstalované Visual Studio, které zpřístupňuje vyvolání příslušného příkazu. Dále je nutné mít nainstalované vlastní IDE od společnosti Apple, kterým je Xcode. Ten je v době psaní této práce dostupný v App Store pro zařízení se systémem macOS verze 13 a vyšší. Pro jeho instalaci je však nutné mít vytvořené vlastní Apple ID. Důležitá je také prvotní konfigurace nástroje Xcode. Abychom jej mohli používat je nutné odsouhlasit licenční podmínky a provést prvotní inicializaci. Obě akce lze provést z terminálu pomocí příkazů „*sudo xcodebuild -licence*“ pro odsouhlasení licenčních podmínek a „*xcodebuild -runFirstLaunch*“ pro prvotní inicializaci.

Na závěr už stačí jen v umístění projektu spustit novou instanci terminálu a vyvolat příkaz „*dotnet publish -f net6.0-maccatalyst -c Release*“ a instalační balíček bude následně publikován do příslušné složky (viz Obrázek 6).

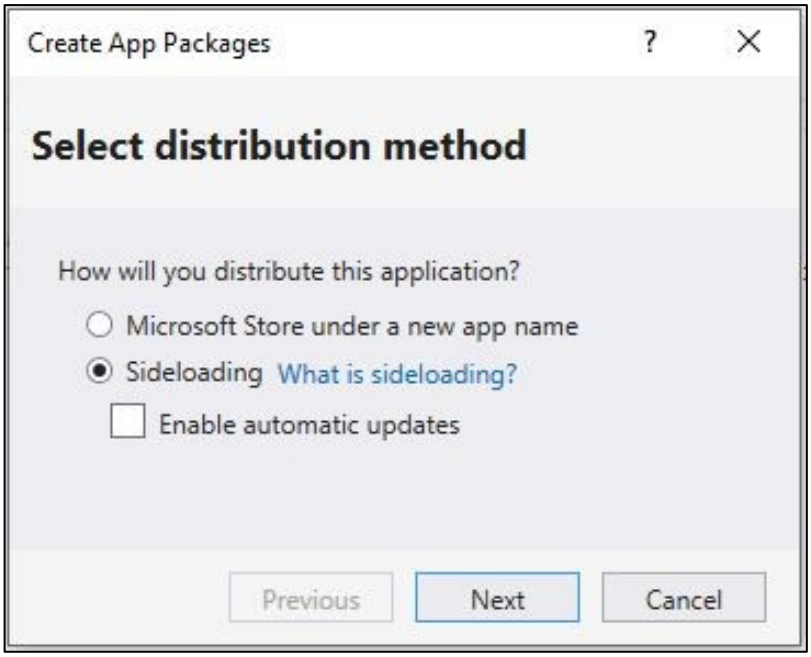
```
jaroslavsykora@mac-mini013 CryptoNativeApp % dotnet publish -f net6.0-maccatalyst -c Release
MSBuild version 17.6.1+8ffc3fe3d for .NET
  Zjišťují se projekty, které se mají obnovit...
  Všechny projekty jsou v aktuálním stavu pro obnovení.
  Detected signing identity:

    Bundle Id: com.companyname.cna
    App Id: com.companyname.cna
  CryptoNativeApp -> /Users/jaroslavsykora/Desktop/CryptoNativeApp/CryptoNativeApp/bin/Release/
net6.0-maccatalyst/maccatalyst-x64/CryptoNativeApp.dll
  Optimalizace velikosti sestavení může změnit chování aplikace. Po publikování nezapomeňte pro
vést test. Viz: https://aka.ms/dotnet-illink
  Optimalizace velikosti sestavení Tento proces může chvíli trvat.
  Created the package: /Users/jaroslavsykora/Desktop/CryptoNativeApp/CryptoNativeApp/bin/Releas
e/net6.0-maccatalyst/maccatalyst-x64/publish/CryptoNativeApp-1.0.pkg
jaroslavsykora@mac-mini013 CryptoNativeApp %
```

Obrázek 6 Publikování pro platformu macOS

10.6.2 Pro platformu Windows

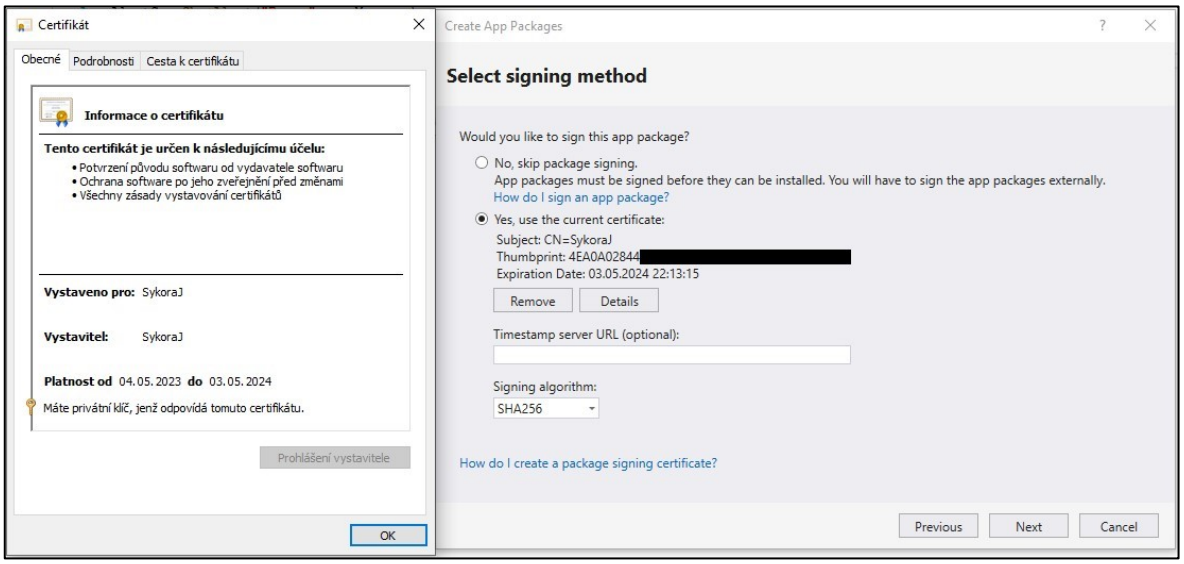
Publikování je zde možné po kliknutí pravého tlačítka myši na projekt aplikace a zvolením možnosti „*Publish“*. Následně jsou dostupné dvě možnosti, a to buď nahrání aplikace na oficiální Microsoft Store s novým názvem aplikace nebo pomocí takzvaného *Sideloadingu*.

Sideloading je využíván pro instalaci aplikací z neoficiálních zdrojů. Pro tuto variantu lze navíc vybrat možnost automatické aktualizace aplikace. Ta funguje tak, že je vybrána instalační složka a interval kontroly této složky o dostupnosti nové verze. Aplikace pak v zadaném intervalu složku kontroluje a pokud je dostupný nový instalační balíček, vyzve uživatele k aktualizaci (viz Obrázek 7).



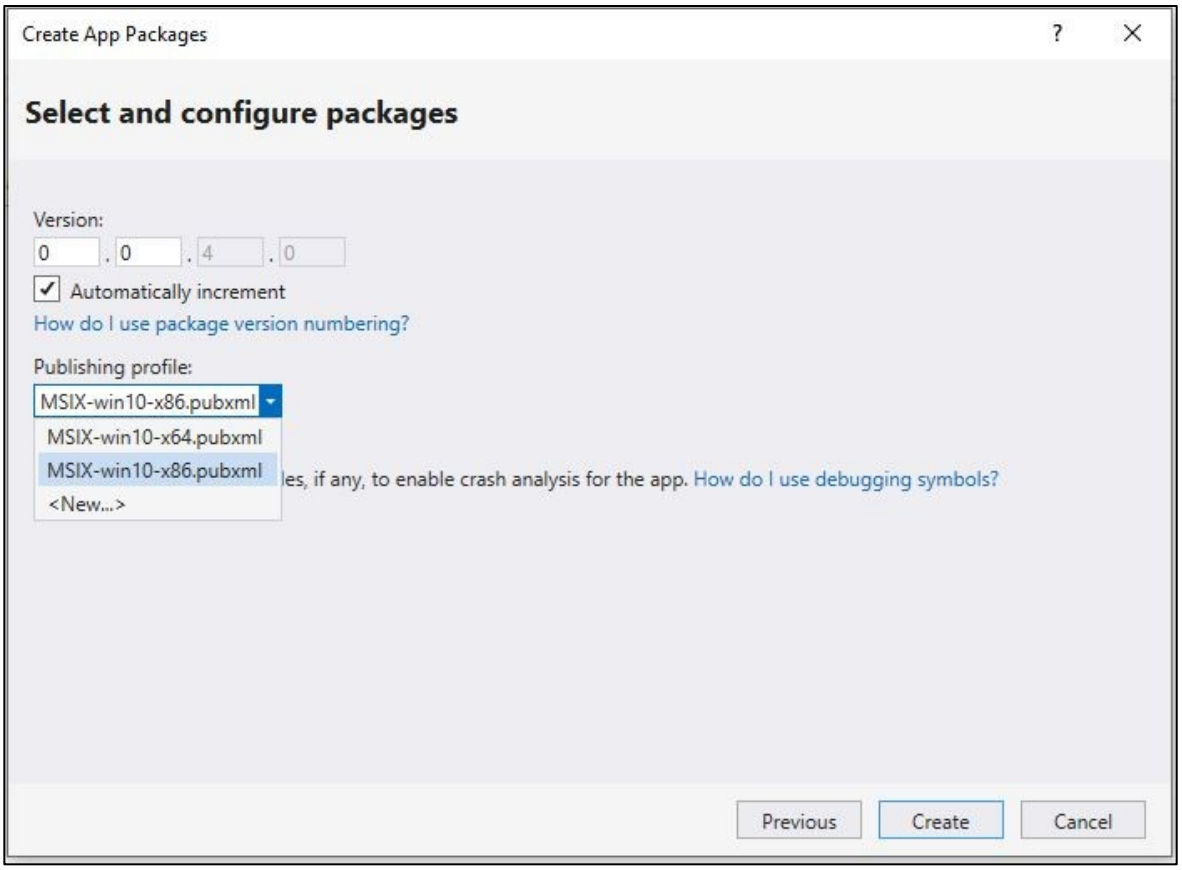
Obrázek 7 Volba distribuční metody

Pro tuto metodu je však nutné použít certifikát k podepsání výsledného balíčku. V následujícím kroku můžeme buď vytvořit nový nebo vybrat již existující certifikát. Tento krok lze prozatím přeskočit ale před instalací aplikace je nutné balíček externě podepsat (viz Obrázek 8).



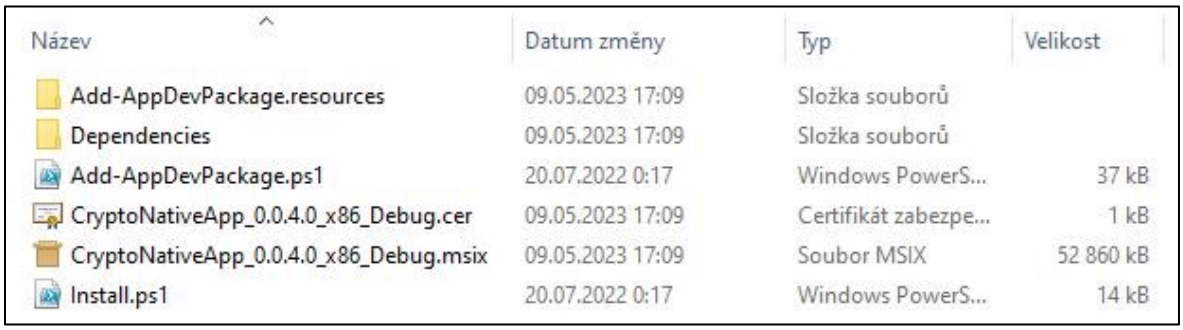
Obrázek 8 Volba certifikátu

V neposlední řadě se definují detaily potřebné pro publikování balíčku a tou je verze a profil. Nejběžnější a nejbezpečnější volbou profilu balíčku je x86, protože tímto máme jistotu, že půjde zprovoznit téměř na každém zařízení. Pokud bychom zvolili verzi x64, tak bychom se omezili pouze na systém Windows 10 s šedesáti čtyř bitovou verzí operačního systému (viz Obrázek 9).



Obrázek 9 Výběr verze a profilu

Nakonec se zobrazí okno s umístěním publikovaného balíčku, který obsahuje všechny potřebné soubory. Pro instalaci však stačí pouze spustit .msix instalátor (viz Obrázek 10).



Obrázek 10 Výsledný balíček

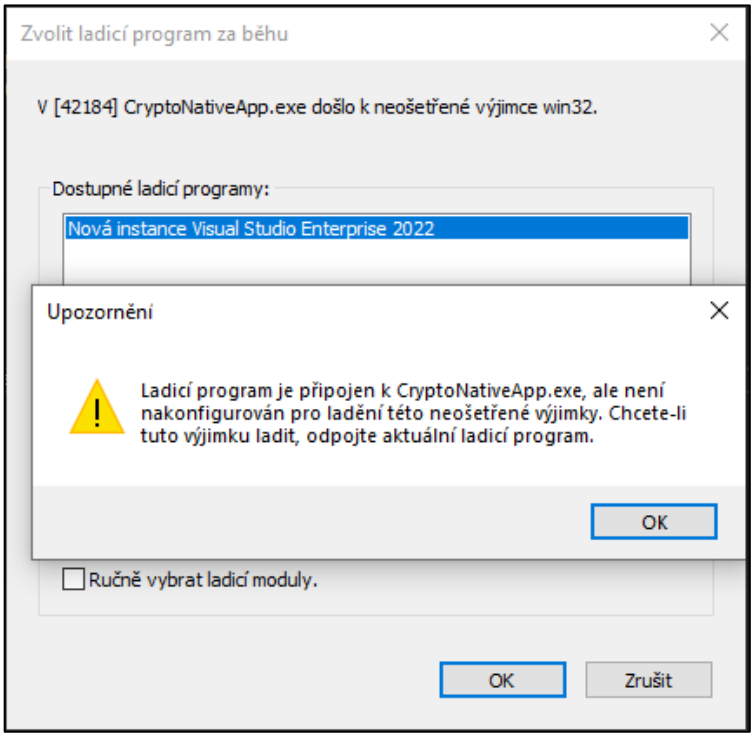
11 PROBLÉMY PŘI VÝVOJI

Vyvíjet aplikace nad novými technologiemi je vždy určitým způsobem hazard, protože mohou obsahovat spoustu nedostatků či chyb. Důvodů pro publikaci produktů, které nejsou v perfektním stavu, je však několik. Mezi nejčastější se řadí asi časový tlak, kdy například firmě docházejí peníze určené na vývoj daného produktu nebo se publikování oddalovalo příliš dlouho a pro udržení potenciálních zákazníků musel být vydán i nedokonalý produkt, který však nese příslib aktualizací, které by právě ony nedokonalosti eliminovaly. Nemusí to však vždy nutně být náročnost vývoje, může se stát, že není poskytnuta detailní dokumentace nebo že produkt není dostatečně otestován a vývojářům tak uniknou některé speciální případy, které vyplynou na povrch až při ostrém testování u zákazníků.

11.1 .NET MAUI

11.1.1 Neošetřené výjimky

Vývoj zde často ztěžovaly neošetřené výjimky bez detailních informací o konkrétním problému. Místo toho, aby aplikace zobrazila chybovou hlášku, dostávalo se mi ve většině případů pouze hlášky o neošetřené výjimce, na kterou není debugger nakonfigurován (viz Obrázek 11).



Obrázek 11 Ukázka neošetřené výjimky

Často se pak tato výjimka zobrazovala v případě, kdy chyba souvisela s knihovnou SecureBlackbox a to ať už při špatné implementaci knihovny, absence runtime licence nebo jakýmkoli jiným problémům.

Řešením k tomuto problému bylo obalovat všechno do takzvaných „*try/catch*“ bloků, tedy do bloků, zachytávajících výjimky. Do bloku „*try*“ se vloží kód (například tělo metody), který se spustí a pokud v průběhu jeho vykonání dojde k chybě, přejde se do bloku „*catch*“, kde se již s danou chybou/výjimkou pracuje.

11.1.2 Hot reload

Další drobností bylo neoptimální fungování funkce *„Hot reload“*, která v případě změny uživatelského rozhraní za běhu aplikace zajišťuje jeho aktualizaci bez nutnosti aplikaci restartovat. Tato funkce je užitečná pro lepší představu vzhledu aplikace s již načtenými daty. Uživatelské rozhraní pak můžeme snadno upravit tak, aby co nejlépe reprezentovala zobrazovaná data.

Tato chyba se pak projevovala, když jsem nějakým způsobem upravoval zobrazení kolekce načtených certifikátů. V momentě, kdy byly provedené změny zásadní jako například změna struktury zobrazení z mřížky *(Grid)* na *StackLayout*, přestala funkce *„Hot reload“* fungovat a sekla se na jednom zobrazení. Jakékoli další úpravy pak nereflektovala a bylo aplikaci nutné restartovat.

11.1.3 Responzivní design

Jelikož aplikace sdílí stejné uživatelské rozhraní je designovým cílem, aby aplikace mohly být jednoduše přizpůsobeny různým zařízením a velikostem obrazovky. Ty se totiž od sebe mohou lišit výškou, šířkou nebo hustotou pixelů. Proto je v .NET MAUI využito konceptu relativního rozvržení a responzivního designu, které znemožňují použití pevného nastavení výšky a šířky. Tohle byl však problém, protože z mého pohledu nebylo žádoucí, aby si uživatel mohl libovolně zvětšovat či zmenšovat aplikaci, která pak v určitých rozměrech nevypadala uživatelsky přívětivě. V takové situaci bych ocenil aspoň možnost definování minimálních a maximálních rozměrů aplikace klidně i pro každou platformu zvlášť. .NET MAUI sice umožňuje nastavení těchto vlastností, ale ty se bohužel vztahují pouze na obsah stránky, nikoli na rozměry samotné aplikace.

Nakonec jsem se rozhodl aspoň nastavit počáteční rozměry aplikace, tak jak bylo původně zamýšleno. Vytvoření responzivních komponent, které se přizpůsobují aktuálním

rozměrům, pak byly nedílnou součástí tohoto řešení. Dosáhl jsem tohoto efektu pomocí kódu, který cílí na konkrétní platformu. Na obrázku níže lze vidět takový kód pro platformu Windows (viz Zdrojový kód 8).

```
#if WINDOWS
        Microsoft.UI.Xaml.Window window = (Microsoft.UI.Xaml.Window)App.Current
.Windows.First<Window>().Handler.PlatformView;

        IntPtr windowHandle = WinRT.Interop.WindowNative.GetWindowHandle(window);

        Microsoft.UI.WindowId windowId = Microsoft.UI.Win32Interop
.GetWindowIdFromWindow(windowHandle);

        Microsoft.UI.Windowing.AppWindow appWindow = Microsoft.UI.Windowing
.AppWindow.GetFromWindowId(windowId);

       appWindow.Resize(new Windows.Graphics.SizeInt32(550, 850));
#endif
```

Zdrojový kód 8 Nastavení rozměrů aplikace

Byla vytvořena instance třídy „*Microsoft.UI.WindwoId*“, která získává referenci na aktivní okno aplikace. Následně je pomocí rozhraní „*WinRT“* získán ukazatel na okno (*windowHandle*). Dále byl vytvořen identifikátor okna (*windowId*), který se získá pomocí tohoto ukazatele získán. Tento identifikátor je použit k vytvoření instance třídy „*Microsoft.UI.Windowing.AppWindow*“, jenž umožňuje správu aplikace. Nakonec je pomocí metody „*Resize*“ nastavena nová velikost okna aplikace (550 pixelů šířky a 850 pixelů výšky).

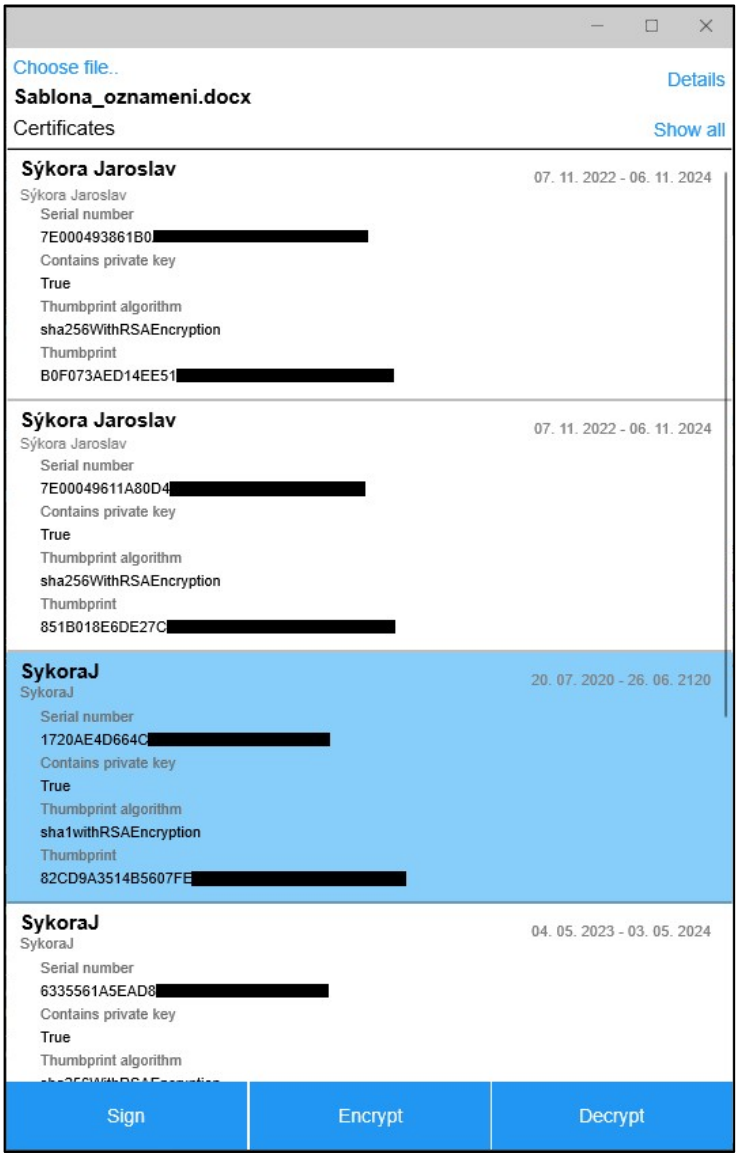
11.2 SecureBlackbox

Problém s knihovnou SecureBlackbox, na který jsem během vývoje narazil byla jednoznačně nepřehledná a velmi obecná dokumentace. Například hned při implementaci knihovny do projektu, konkrétně tedy u odkazování na .dll soubory knihovny. Dokumentace uvádí soubor „*nsoftware.SecureBlackbox.dll*“, umístěný v hlavním adresáři instalace, jako základní knihovnu podporující runtime edice .NET Framework 4.0, .NET Core 3.0, .NET 5 i jejich vyšší verze. To však z nějakého důvodu v mém případě nefungovalo. Bylo tedy nutné odkazovat na konkrétní knihovnu pro .NET 6.0, která je v příslušném podadresáři.

12 PŘEDSTAVENÍ APLIKACE

Aplikace byla vypracována dle původního vzoru, tak aby se od něj zásadně nelišila a zůstala tak uživatelsky stejně přívětivá.

Hlavním rozdílem v uživatelském rozhraní mezi původní a nově vytvořenou aplikací je v reprezentaci zobrazovaných certifikátů, kdy v původní aplikaci byl zvolen přístup rozbalovacích „*záložek*“ a byly zobrazeny pouze informace o držiteli a poskytovateli certifikátu společně s časovým rozmezím jeho platnosti. Zbylé informace byly přístupné až po kliknutí na danou záložku. Nyní jsou však jednotlivé certifikáty rovnou zobrazovány společně s podrobnějšími informacemi. Pro jejich lepší vizuální odlišení byla přidána tenká čára, kdy bylo tímto způsobem zamezeno případnému vizuálnímu splývání dat.



Obrázek 12 Nová aplikace

12.1 Popis jednotlivých částí

12.1.1 Výběr souboru

Uživatel má v tomto kroku možnost zadání souboru, pro který chce následně vykonat jednu z nabízených akcí. Výběr je však omezený pouze na určité typy, jelikož aplikace umožňuje pouze práci se soubory MS Office (konkrétně pak word či excel) nebo PDF a XML dokumenty.

Akce pro výběr se vyvolá po kliknutí na tlačítko „*Choose file…*“ a pokud byl soubor vybrán, je následně zobrazen pod tímto tlačítkem (viz Obrázek 11). Po vybrání souboru si uživatel může zobrazit detaily daného souboru, které obsahují jeho hash, jenž je určený k ověření pravosti souboru.

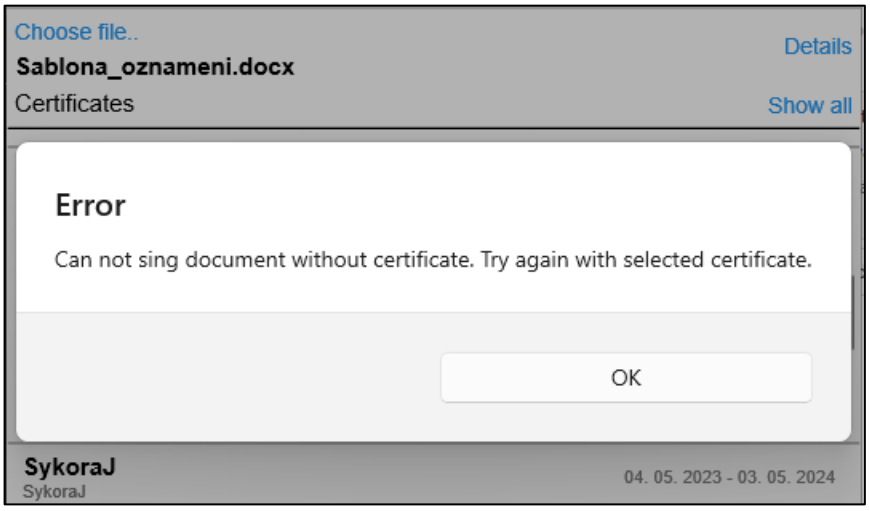
12.1.2 Zobrazení certifikátů

Certifikáty jsou načteny ze systémového úložiště a zobrazeny pod sebou společně s jejich základními informacemi, jako je například vlastník (tučně zvýrazněné) a poskytovatel (menší šedé písmo pod vlastníkem) certifikátu, jeho otisk (*Thumbprint*) nebo platnost.

Vybrat konkrétní certifikát lze pomocí kliknutí na libovolné buňku, obsahující informace o certifkátu. Ta se následně barevně zvýrazní. Zrušit aktuální výběr lze dvěma způsoby, a to vybráním nového certifikátu nebo pomocí klávesové zkratky *CTRL* + klik levým tlačítkem myši (viz Obrázek 12).

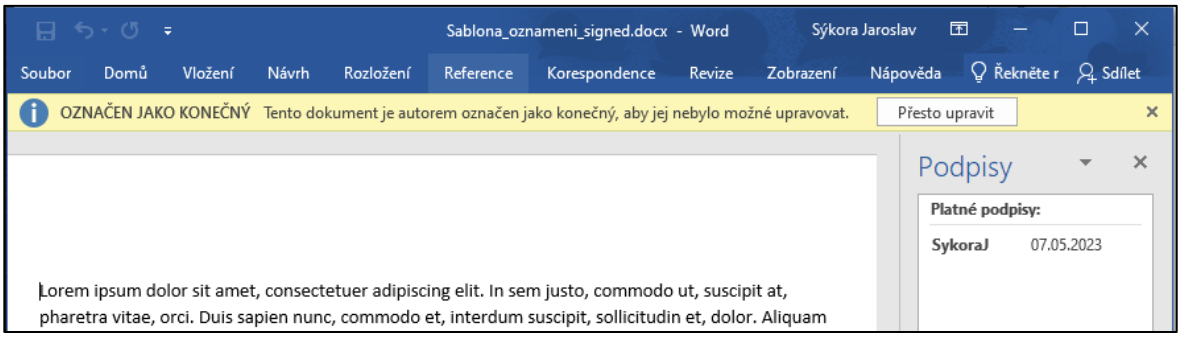
12.1.3 Podepisování

Pro podepsání nahraného dokumentu musí být vybrán certifikát, jinak bude uživateli zobrazena chybová hláška o absenci certifikátu (viz Obrázek 13).



Obrázek 13 Chybová hláška při podpisu bez certifikátu

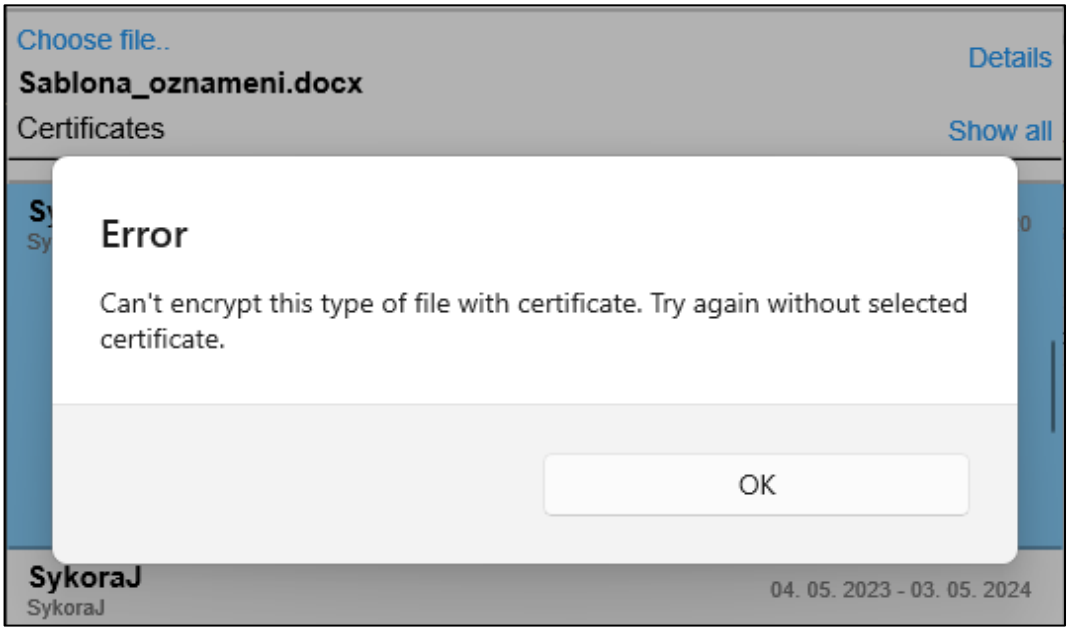
Po kliknutí na tlačítko „*Sign*“ se provede podepsaní nahraného souboru, kdy se do stejného adresáře, odkud byl nahrán původní soubor, uloží nově vygenerovaný s názvem prováděné operace. Pro případ z výše uvedeného obrázku by tedy byl v následujícím tvaru: „*Sablona_oznameni_signed.docx“*.



Obrázek 14 Ukázka podepsaného dokumentu

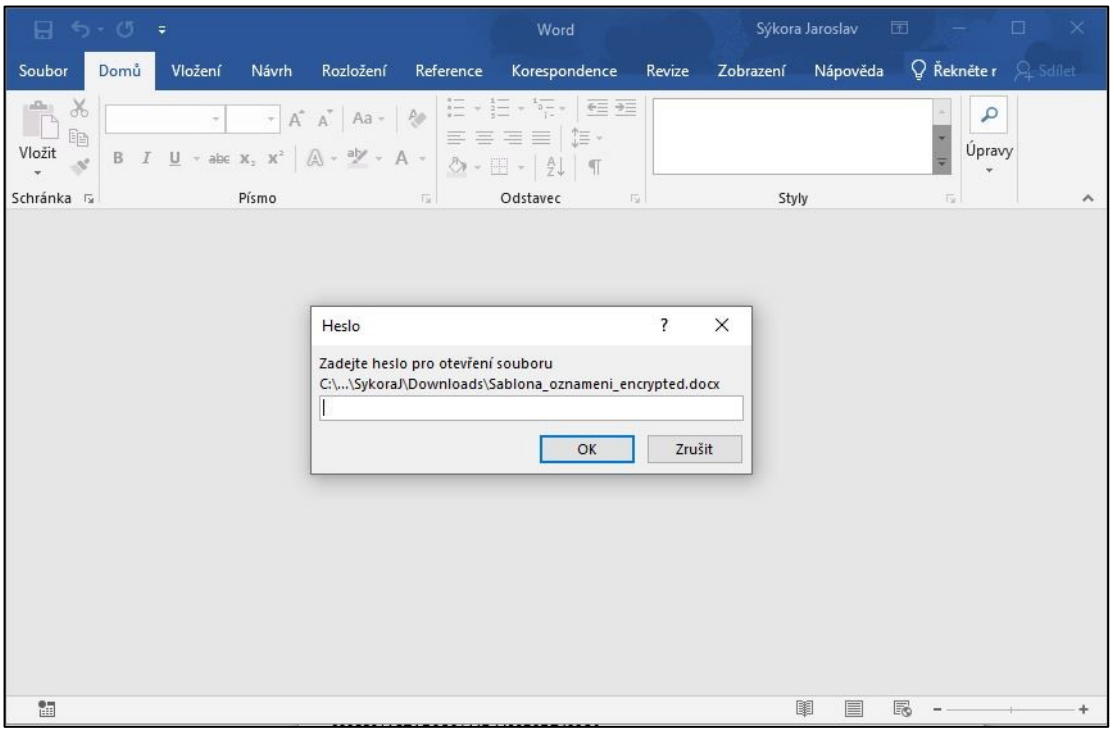
12.1.4 Šifrování a dešifrování

Pro šifrování a dešifrování jsou odlišně definovány podmínky (viz. Podkapitola 10.2 Podmínky). Uživatel si pro šifrování či dešifrování PDF dokumentu může zvolit, zda chce danou operaci provést s použitím certifikátu či hesla. V případě souborů MS Office či XML je pak možná pouze varianta se zadáním hesla, v opačném případě by se uživateli zobrazila chybová hláška o nepodporované akci (viz Obrázek 15).



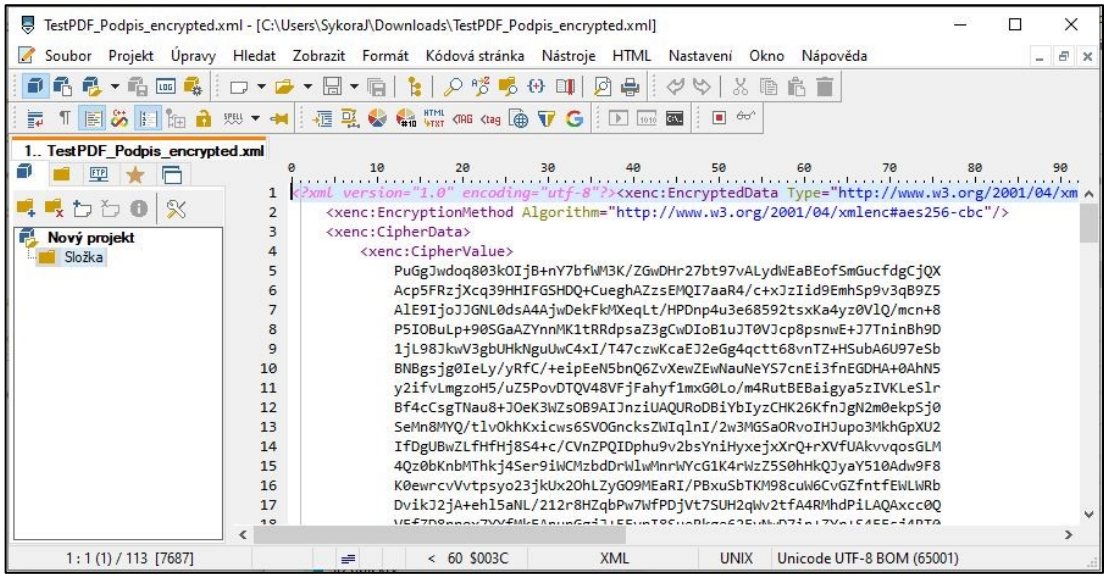
Obrázek 15 Chybová hláška při šifrování

Pro čtení, heslem zašifrovaného dokumentu, je pak potřeba po jeho otevření zadat heslo nebo pomocí nástroje daný dokument dešifrovat (viz Obrázek 16).



Obrázek 16 Ukázka zašifrovaného dokumentu

XML soubory jsou výjimkou, kdy nejste vyzváni k zadání hesla a zobrazují se pouze zašifrovaná data. Pro práci s tímto zašifrovaným souborem je tedy nutné soubor nejprve dešifrovat pomocí nástroje (viz Obrázek 17).



Obrázek 17 Ukázka zašifrovaného XML dokumentu

13 NAVRHOVANÁ DOPORUČENÍ

Ačkoli aplikace funguje tak, jak má a splňuje definované požadavky, stále má prostor pro případná vylepšení.

13.1 Nahrání certifikátu ze souboru

Kolekce zobrazovaných certifikátů by šla rozšířit o možnost nahrání certifikátu ze souboru. Ty by mohly být například typu PFX neboli Personal Information Exchange, který představuje certifikát zapouzdřený v heslem chráněném souboru. Často se takový soubor s certifikátem používá při podpisu kódu aplikace. Příkladem je použití takového certifikátu v průběhu publikování instalačního balíčku v této práci.

13.2 Načtení certifikátu z příslušného nosiče

Dále by bylo příhodné přidat možnost použít certifikát načtený přímo z čipové karty nebo jiného nosiče jako například USB tokenu. V případě použití takového certifikátu by se uživateli zkrátila doba hledání odpovídajícího certifikátu v kolekci.

13.3 Zlepšení navigace mezi certifikáty

K lepší práci s certifikáty by pak mohlo posloužit vyhledávací pole, kde by uživatel mohl vyhledávat například zadáním vlastníka certifikátu, nebo přidáním sekce naposledy použitých certifikátů. Taková sekce by mohla obsahovat například 3 naposledy použité certifikáty, což je dle mého optimální číslo pro zachování přehledného uživatelského prostředí. Každá sekce by pak mohla být vizuálně odlišena, aby se jednotlivé certifikáty z různých oblastí neprolínaly. Například část s certifikáty načtenými ze systémového úložiště a naposledy použitými.

ZÁVĚR

V rámci této diplomové práce jsem se věnoval analýze a vývoji multi-platformní aplikace s využitím moderních technologií. Teoretická část práce se zaměřila na vývoj multi-platformních aplikací a technologií, které jsou pro tento účel vhodné. Byly podrobně popsány technologie .NET MAUI a jazyk C#, které představují základní stavební kameny pro vývoj této aplikace. Dále jsem se zabýval objektově orientovaným programováním, reverzním inženýrstvím a zabezpečením dokumentů, které se také řadí mezi další důležité aspekty.

V praktické části jsem představil původní aplikaci a provedl její analýzu. Na základě této analýzy jsem vypracoval návrh nové verze aplikace, který využívá modernější technologie a přináší vylepšení v oblasti uživatelského rozhraní, výkonu a bezpečnosti. Vývoj nové aplikace probíhal v jazyce C# s využitím platformy .NET MAUI.

Během vývoje jsem se setkal s několika problémy, které jsem systematicky analyzoval a řešil. Byla provedena řada testů, které měly za cíl odhalit případné nedostatky a chyby v aplikaci. Tyto nedostatky byly následně odstraněny a aplikace byla důkladně odladěna.

V závěrečné fázi práce jsem prezentoval výslednou aplikaci. Současný stav technologií pro vývoj multi-platformních aplikací byl podrobně popsán a vhodné technologie pro tvorbu aplikace byly úspěšně zvoleny. Původní aplikace byla důkladně analyzována a na základě této analýzy byl vypracován návrh nové verze aplikace, který byl úspěšně implementován. Výsledná aplikace byla testována a odlaďována, čímž byly odstraněny případné nedostatky.

Závěrem této práce je formulování doporučení pro budoucí rozvoj aplikace. Na základě získaných poznatků a zkušeností je možné aplikaci dále vylepšovat a rozšiřovat o další funkcionality. Doporučuji také dále sledovat vývoj technologií a novinek v oblasti multi-platformního vývoje aplikací, aby bylo možné využít moderních nástrojů a technologií pro dosažení ještě lepších výsledků.

Tato diplomová práce přinesla přehled současných technologií pro vývoj multi-platformních aplikací a úspěšně splnila všechny body zadání. Výsledná aplikace využívá moderních technologií a díky provedené analýze, návrhu, implementaci, testování a odlaďování byly dosaženy kvalitní výsledky.

SEZNAM POUŽITÉ LITERATURY

[1] The Ultimate Guide to Cross Platform App Development Frameworks in 2023. Software Company & Web App Development Agency | Net Solutions [online]. Dostupné z: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/

[2] FILIPOVA, Olga a Rui VILÃO. Software Development From A to Z: A Deep Dive into all the Roles Involved in the Creation of Software. Imprint: Apress, 2018. ISBN 9781484239445.

[3] What Is Cross-Platform Software Development & Why Use It?. WEBO Digital – Customer Experience Transformation Partner [online]. Copyright © 2022 WEBO DIGITAL [cit. 19.05.2023]. Dostupné z: https://webo.digital/blog/what-is-cross-platform-software-development/

[4] 10 best cross platform app development frameworks in 2021. Latest Insights on Mobile App Development | MobileAppDaily [online]. Copyright © 2023 [cit. 19.05.2023]. Dostupné z: https://www.mobileappdaily.com/top-cross-platform-app-devp-frameworks

[5] What is Xamarin? - Xamarin | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 19.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[6] From Xamarin to MAUI, What has changed? - Flat Rock Technology. Flat Rock Technology [online]. Copyright © Flat Rock Technology Ltd [cit. 21.05.2023]. Dostupné z: https://flatrocktech.com/xamarin-forms-maui-migration/

[7] All About .NET MAUI. .NET MAUI | by Alper Ebiçoğlu | Volosoft | Medium. Medium – Where good ideas find you. [online]. Dostupné z: https://medium.com/volosoft/all-about-net-maui-5a1e774803ac

[8] Co je .NET MAUI? - .NET MAUI | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 21.05.2023]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/what-is-maui

[9] Describe the .NET MAUI architecture – Training | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 21.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/training/modules/build-mobile-and-desktop-apps/2-describe-maui-architecture?ns-enrollment-type=learningpath&ns-enrollment-id=learn.dotnet-maui.build-apps-with-dotnet-maui

[10] The history of C# – C# Guide | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 21.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

[11] HEJLSBERG, Anders. The C# programming language. 4th ed. Addison-Wesley Professional, 2004. ISBN 978-0-321-74176-9.

[12] ROBINSON, Simon. *C#: programujeme profesionálně*. Brno: Computer Press, 2003. Programmer to programmer. ISBN 80-251-0085-5.

[13] What's new in .NET 6 | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 21.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6

[14] Lekce 1 - Úvod do C# a .NET frameworku. itnetwork.cz - Učíme národ IT [online]. Copyright © 2023 itnetwork.cz. Veškerý obsah webu [cit. 21.05.2023]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

[15] C# Tutorial – GeeksforGeeks. GeeksforGeeks | A computer science portal for geeks [online]. Dostupné z: https://www.geeksforgeeks.org/csharp-programming-language/;https://www.altexsoft.com/blog/c-sharp-pros-and-cons/

[16] What is Visual Studio? | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 21.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

[17] What is Object-Oriented Programming (OOP)?. Purchase Intent Data for Enterprise Tech Sales and Marketing – TechTarget [online]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

[18] C# OOP (Object-Oriented Programming). W3Schools Online Web Tutorials [online]. Dostupné z: https://www.w3schools.com/cs/cs_oop.php

[19] C# Corner : Looking for Something?. C# Corner – Community of Software and Data Developers [online]. Copyright ©2023 C [cit. 21.05.2023]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/84c85b/object-oriented-programming-using-C-Sharp-net/

[20] A Complete Guide To Object Oriented Programming In C#. C# Corner - Community of Software and Data Developers [online]. Copyright ©2023 C [cit. 21.05.2023]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/84c85b/object-oriented-programming-using-C-Sharp-net/

[21] NESTERUK, Dmitri. Design Patterns in Modern C++: Reusable Approaches for Object-Oriented Software Design. Imprint: Apress, 2018. ISBN 978-1-4842-3602-4.

[22] Design Patterns in C#. Refactoring and Design Patterns [online]. Dostupné z: https://refactoring.guru/design-patterns/csharp

[23] Advantages and Disadvantages of OOP – GeeksforGeeks. GeeksforGeeks | A computer science portal for geeks [online]. Dostupné z: https://www.geeksforgeeks.org/benefits-advantages-of-oop/

[24] How to Reverse Engineer Software (Windows) the Right Way | Apriorit. Apriorit – Software Development Company: Outsourcing Programming Services [online]. Copyright © 2004 [cit. 21.05.2023]. Dostupné z: https://www.apriorit.com/dev-blog/364-how-to-reverse-engineer-software-windows-in-a-right-way

[25] Static vs Dynamic Analysis. Site not found · GitHub Pages [online]. Dostupné z: https://rahulsinghinfosec.github.io/hackme/reverse-engineering/static-vs-dynamic-analysis.html

[26] Hacker Lexicon: What Is Fuzzing? | WIRED. WIRED – The Latest in Technology, Science, Culture and Business | WIRED [online]. Copyright © [cit. 21.05.2023]. Dostupné z: https://www.wired.com/2016/06/hacker-lexicon-fuzzing/

[27] 7 Types of Cyber Security Threats. Online Degree Programs | University of North Dakota [online]. Copyright © 2023 University of North Dakota. [cit. 21.05.2023]. Dostupné z: https://onlinedegrees.und.edu/blog/types-of-cyber-security-threats/

[28] 11 Best Practices for Document Management Security – N-able. MSP + IT Management Software: RMM, Backup, Security – N-able [online]. Copyright © 2022 N [cit. 21.05.2023]. Dostupné z: https://www.n-able.com/blog/11-best-practices-for-document-management-security

[29] What is Document Security: The Complete Guide (Including Tips). Information Management Simplified [online]. Copyright © 2023 The ECM Consultant, Barbour street, [cit. 21.05.2023]. Dostupné z: https://theecmconsultant.com/document-security/

[30] MENEZES, A. J., Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. ISBN 9780849385230.

[31] What is Document Security?. Nira | Real-time access control [online]. Copyright © 2023, Nira. [cit. 21.05.2023]. Dostupné z: https://nira.com/document-security/

[32] KATZ, Jonathan a Yehuda LINDELL. Introduction to modern cryptography. Second edition. Boca Raton, FL: CRC Press, [2015]. ISBN 1466570261id.

SEZNAM POUŽITÝCH ZKRATEK

.NET MAUI	Multi-platform App UI
3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
AES128	Advanced Encryption Standard 128
AES192	Advanced Encryption Standard 192
AES256	Advanced Encryption Standard 256
BCL	Base Class Library
CNA	Crypto Native Application
CSS	Cascading Style Sheets
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptography
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated development environment
MS	Microsoft
OOP	Object-oriented programming
PDF	Portable Document Format
PFX	Personal Information Exchange
RSA	Rivest-Shamir-Adleman
SDK	Software development kit
USB	Universal Serial Bus
UX	User experience
WinUI	Windows User Interface library
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD SE ZDROJOVÝMI KÓDY