Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Multiplatformní vývoj aplikací pomocí frameworků MAUI a Blazor

Martin Homza

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Martin Homza

Informatika

Název práce

Multiplatformní vývoj aplikací pomocí frameworků MAUI a Blazor

Název anglicky

Multi-platform app development with frameworks MAUI and Blazor

---

Cíle práce

Hlavním cílem této práce je popsat a zhodnotit možnosti multiplatformního vývoje aplikací pomocí frameworků MAUI a Blazor. Toto zhodnocení bude založeno na vývoji testovací multiplatformní aplikace. Výsledkem práce bude vyhodnocení použité technologie a její praktické využitelnosti na základě zvolených kritérií.

Metodika

Teoretická část práce bude založena na studiu odborné literatury popisující

využité technologie a metody. Získané informace a poznatky budou následně formulovány do teoretických východisek práce.

V praktické části bude autorem popsána s danými požadavky testovací aplikace, která bude následně navržena a implementována. Tato aplikace bude vyvinuta za využití frameworků MAUI a Blazor.

Nakonec bude u aplikace hodnoceno využití a možnosti frameworků MAUI a Blazor na základě zvolených kritérií. Na základě získaných výsledků bude zformulován závěr práce.

Doporučený rozsah práce

60-80 stran

Klíčová slova

WebAssembly, MAUI, Blazor, C#, .NET 6, Multiplatformní vývoj

Doporučené zdroje informací

BAPTISTA, Gabriel a Francesco ABBRUZZESE. Software Architecture with C# 10 And . NET 6. 3. Birmingham: Packt Publishing, Limited, 2022. ISBN 9781484259283.

HIMSCHOOT, Peter. Microsoft Blazor. 2. Melle: Apress L. P., 2020. ISBN 9781484259283.

PRICE, Mark J. C# 10 and . NET 6 – Modern Cross-Platform Development. 6. Birmingham: Packt Publishing, Limited, 2021. ISBN 9781801076968.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 11. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 11. 2022

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Multiplatformní vývoj aplikací pomocí frameworků MAUI a Blazor" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za vedení a pomoc při zpracování této diplomové práce a také své rodině za podporu.

Multiplatformní vývoj aplikací pomocí frameworků MAUI a Blazor

Abstrakt

Tato práce pojednává o vývoji multiplatformních aplikací prostřednictvím frameworků *.NET MAUI* a *Blazor*. Hlavním cílem je popsání a zhodnocení daných frameworků včetně jejich možností a praktické využitelnosti dle zvolených kritérií na základě zjištěných poznatků z odborné literatury a vývoje testovací multiplatformní aplikace

Teoretická část práce definuje prostředí počítačových platforem, včetně jejich uživatelského zastoupení. Dále popisuje operační systémy, na které multiplatformní technologie nejčastěji cílí, včetně vývoje a distribuce aplikací pro dané operační systémy. Poté jsou popsány přístupy multiplatformního vývoje, včetně jejich vlastností a technologií. Nakonec je popsána platformat *.NET* s frameworky *.NET MAUI* a *Blazor*.

Praktická část práce je věnována analýze a implementaci testovací multiplatformní aplikace. V rámci analýzy jsou stanoveny požadavky na chování a implementaci aplikace, možnosti rozšíření, architektura, uživatelské rozhraní a datový model. Část popisující implementaci rozebírá přípravu projektu, včetně struktury a přídavných modulů a implementaci jednotlivých prvků aplikace.

Následně je framework *.NET MAUI* doplněný frameworkem *Blazor* zhodnocen na základě zvolených kritérií.

Klíčová slova: WebAssembly, MAUI, Blazor, C#, .NET 6, Multiplatformní vývoj, Hybridní vývoj

Multi-platform app development with frameworks MAUI and Blazor

Abstract

This thesis discusses the development of cross-platform applications using the *.NET MAUI* and *Blazor* frameworks. The main goal is the description and evaluation of the given frameworks, including their possibilities and practical usability according to the selected criteria, based on the findings from the professional literature and the development of a testing multi-platform application

The theoretical part of the thesis defines the environment of computer platforms, including their user representation. It also describes the operating systems that multiplatform technologies most often target, including the development and distribution of applications for those operating systems. Multiplatform development approaches are then described, including their features and technologies. Finally, the *.NET* platform with the *.NET MAUI* and *Blazor* frameworks is described.

The practical part of the work is devoted to the analysis and implementation of a test multiplatform application. As part of the analysis, requirements for application behavior and implementation, extension options, architecture, user interface and data model are determined. The part describing the implementation discusses the preparation of the project, including the structure and additional modules and the implementation of individual elements of the application.

Subsequently, the *.NET MAUI* framework supplemented by the *Blazor* framework is evaluated based on the selected criteria.

Keywords: WebAssembly, MAUI, Blazor, C#, .NET 6, Multi-platform app development, Hybrid app development

Obsah

1 Úvod

V současné době existuje široká škála hojně využívaných hardwarových a softwarových platforem, které využívají lidé jak k práci, tak k osobním i dalším potřebám. Mnoho lidí má například osobní počítač a mobilní telefon, tedy dvě pravděpodobně hardwarově i softwarově rozdílná zařízení. Na aplikace je však i přes tuto značnou platformní diverzitu kladen důraz, aby je bylo možné spustit na libovolném zařízení. Často se tak očekává, že aplikace určená pro osobní počítače lze pustit na operačních systémech *Windows*, *macOS* a na *linuxových distribucích*. Obdobně u mobilních aplikací se očekává, že existuje verze jak pro operační systém *Android*, tak i pro *iOS*. Lze se setkat však i se situací, kdy je potřeba, aby ani hardwarová platforma rozsah působnosti aplikace neomezovala a daná aplikace existovala jako nativní aplikace pro osobní počítače, mobilní telefony a zároveň například i jako webová aplikace pro webové prohlížeče.

Problém dostupnosti aplikací na několika platformách zároveň lze řešit několika způsoby. Prvním způsobem bylo vytváření nativní verze aplikace pro každý operační systém zvlášť. Tento způsob je často velmi náročný časově i finančně a vyžaduje odborníky na vývoj pro všechny cílené platformy. Nativní vývoj má však i své výhody v často vyšší výkonnosti aplikace a větších možnostech využití dané platformy, a tudíž se i nyní ve značné míře využívá.

Kvůli nevýhodám nativního vývoje začaly samozřejmě vznikat způsoby vývoje aplikací, které jsou platformě nezávislé. Vznikly tak metody vývoje multiplatformních aplikací, hybridních aplikací, progresivních webových aplikací a vývoj webových aplikací nabral na popularitě. Tyto metody často nabízejí dobrou znuvupoužitelnost kódu, časové a finanční úspory oproti nativnímu vývoji. Vývoj aplikací pomocí těchto metod je čím dál více populární i přes stále často využívaný způsob vývoje několika nativních aplikací. Způsob vývoje platformě nezávislých aplikací významně zpopularizoval například framework *Flutter*, ale i další frameworky jako *React Native*, *Kotlin Multiplatform Mobile*, *Ionic*, *Xamarin.Forms* a další. Právě metodami vývoje platformě nezávislých aplikací se zaměřením na nové frameworky *.NET MAUI* a *Blazor* se zabývá tato diplomová práce.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je popsat a zhodnotit možnosti multiplatformního vývoje aplikací pomocí frameworků *MAUI* a *Blazor*. Toto zhodnocení je založeno na vývoji testovací multiplatformní aplikace. Výsledkem práce je vyhodnocení použité technologie a její praktické využitelnosti na základě zvolených kritérií.

2.2 Metodika

Teoretická část práce je založena na studiu odborné literatury popisující využité technologie a metody. Získané informace a poznatky jsou následně formulovány do teoretických východisek práce. Na základě těchto informací a poznatků jsou popsány počítačové platformy, operační systémy, přístupy k vývoji multiplatformních aplikací a technologie pro tvorbu multiplatformních aplikací. Nakonec je popsána platforma *.NET* včetně frameworků *.NET MAUI* a *Blazor*.

V praktické části je autorem analyzována, navržena a popsána s danými požadavky na chování a implementaci testovací multiplatformní aplikace. Požadavky jsou stanovené tak, aby byly dostatečně prověřeny možnosti zkoumaných technologií. Tato aplikace je navržena pro vývoj za využití frameworků *.NET MAUI*, *Blazor*, lokální databáze *SQLite* a webových technologií. V rámci analýzy je také popsána architektura, uživatelské rozhraní a datový model aplikace. Testovací multiplatformní aplikace je následně implementována a postup implementace aplikace je popsán.

Nakonec je na základě vytvořené testovací aplikace a získaných poznatků z odborných zdrojů zhodnocen framework *.NET MAUI* doplněný frameworkem *Blazor* pro vývoj multiplatformních aplikací včetně jejich možností a praktické využitelnosti podle zvolených charakteristik. Tyto charakteristiky jsou zvoleny autorem práce na základě získaných poznatků z odborných zdrojů a vývoje testovací aplikace tak, aby byl udržen celkový pohled na problém a došlo k hodnocení z většího množství úhlů pohledu. Na základě získaných výsledků je zformulováno závěrečné vyhodnocení a závěr práce.

3 Teoretická východiska

3.1 Počítačové platformy

Pro pochopení technik multiplatformního vývoje je nejprve potřeba vysvětlit termín počítačová platforma. Platforma je v informatice definována jako pojem označující počítačový systém, který má určitý hardware a software. Platforma umožňuje běh aplikací, avšak pouze těch, jež jsou přímo pro ni vytvořeny s ohledem právě na její hardware a software.

3.1.1 Hardwarové a softwarové platformy

Hardwarově je platforma definována zejména vlastnostmi procesoru a schopností zpracovávat určitou instrukční sadu. Mohou být však brány v úvahu i podmínky na další hardwarové komponenty.

Softwarově definuje platformu zejména operační systém. Lze však dále specifikovat i podporovaná běhová prostředí programovacích jazyků a aplikační frameworky či knihovny. Operační systémy se mezi sebou odlišují v mnoha ohledech, a to jak vlastní architekturou, tak i přístupy a vlastnostmi působícími na uživatele i vývojáře. Vývojáři tak například jinak přistupují a ovládají prvky uživatelského rozhraní, či jinak přistupují k souborovému systému. Každý operační systém má také vlastní designový systém definující vzhled uživatelského rozhraní (Bigelow, 2021).

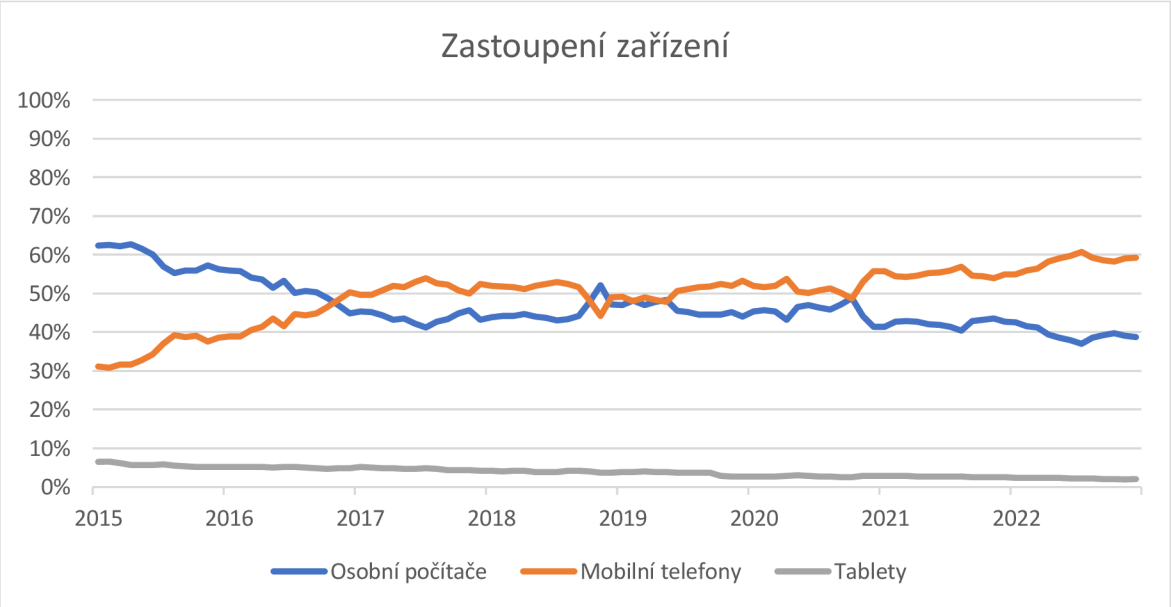
Pro účel vývoje univerzálních multiplatformních aplikací pro pracovní a osobní využití pomocí běžných multiplatformních nástrojů, lze jako hlavní zástupce hardwarových platforem považovat osobní počítače a mobilní zařízení. Přičemž mezi osobní počítače můžeme řadit stolní počítače a notebooky a mezi mobilní zařízení zejména mobilní telefony a tablety. Je však potřeba brát v úvahu i existenci dalších hardwarových zařízení, u kterých lze pro vývoj aplikací využívat techniky multiplatformního vývoje, jako jsou superpočítače, zástupci nositelné elektroniky, herní konzole, či jiná zařízení se specifickým účelem. Multiplatformní nástroje však většinou definují pouze operační systémy, které podporují, neboť se očekává, že jsou nainstalovány na svou primární hardwarovou platformu (LINFO, nedatováno).

Důležitým ukazatel při vývoji aplikací je mimo potřebných vlastností kladených na dané platformy také informace o počtu jejich uživatelů.

3.1.2 Uživatelské zastoupení platforem

Přibližné informace o počtu uživatelů, či jak často využívají jednotlivá zařízení a operační systémy lze získat například na webové stránce *gs.statcounter.com*, která získává data o počtech uživatelů z údajů o návštěvnosti několika referenčních webových stránek. Získaná data jsou zobrazena pomocí následujících grafů popisujících procentní uživatelské zastoupení platforem mezi roky 2015 a 2022.

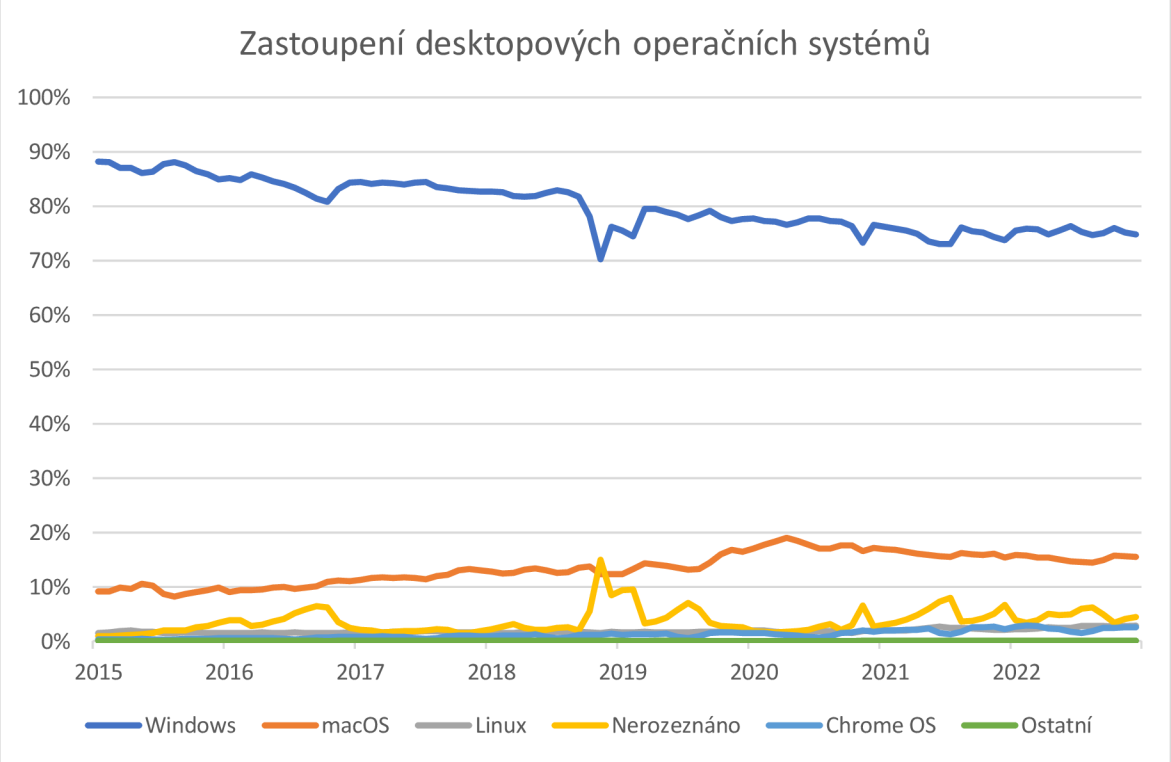
V prvním grafu je zobrazeno uživatelské zastoupení jednotlivých druhů zařízení, tedy osobních počítačů, mobilních telefonů a tabletů zvlášť. Po celou dobu zobrazovanou grafem jsou nejméně používaným typem zařízení tablety a jejich zastoupení se pohybuje v řádu několika jednotek procent. Hojně používané jsou osobní počítače a mobilní telefony. Téměř do konce roku 2016 byly primárně používané osobní počítače. Z mnoha důvodů a díky rozvoji mobilní platformy, co se týče jak hardwaru a softwaru mobilních telefonů, tak i optimalizaci webových stránek a aplikací, se staly na úkor osobních počítačů nejvíce používaným typem zařízení právě mobilní telefony. Existuje však mnoho aplikací, které jsou vhodné pouze pro osobní počítače, či pouze pro mobilní telefony a nelze tak očekávat, že by jedna platforma zcela přebrala zastoupení druhé (Statcounter Global Stats, nedatováno).



Graf 1 - Zastoupení zařízení během posledních několika let (Statcounter Global Stats, nedatováno)

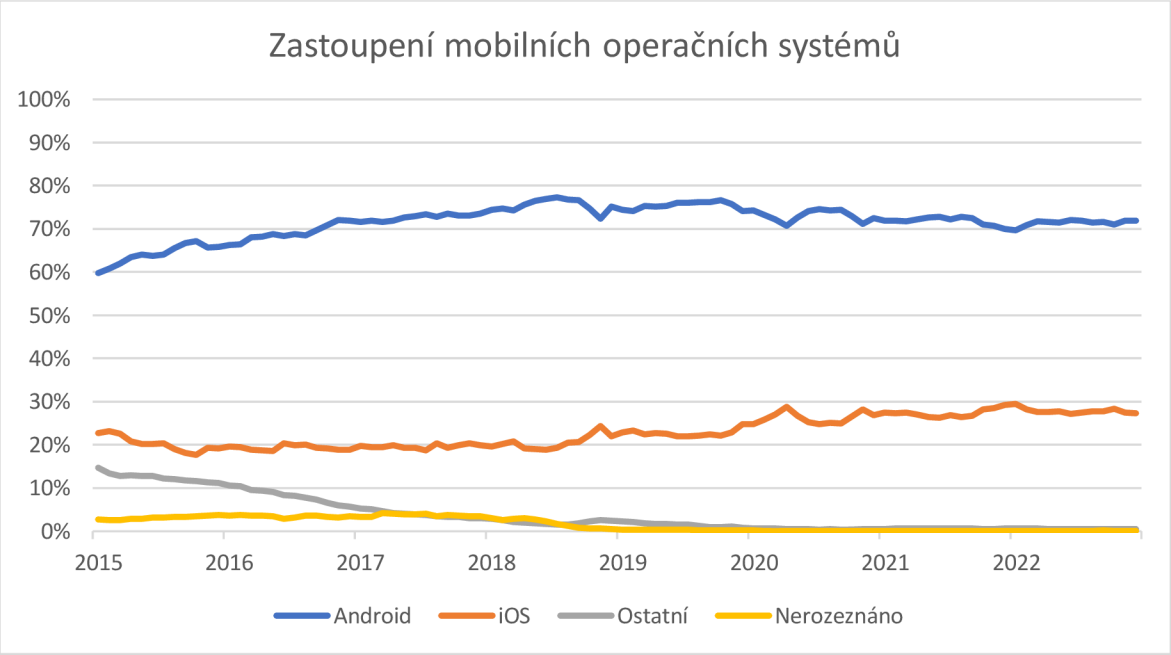
Dále lze popsat uživatelské zastoupení u jednotlivých operačních systémů. Jak osobní počítače, tak mobilní telefony mají své specifické operační systémy. Situace se liší u tabletů, pro které se tolik nevyvíjí specifické operační systémy a často využívají operační systémy pro osobní počítače, či mobilní telefony.

Na základě dat zobrazených v grafu 2, lze říct, že mezi nejvíce používané operační systémy pro osobní počítače patří *Windows*, *macOS*, *Chrome OS* a *Linuxové distribuce*. Značně největší dlouhodobé zastoupení má operační systém *Windows* od společnosti *Microsoft*, zejména díky jeho rozšířenosti na osobních počítačích několika výrobců. Jeho nejsilnějším konkurentem, co se týče počtu uživatelů je operační systém *macOS*, operační systém určený pro osobní počítače společnosti *Apple*. Nejmenší zastoupení má dlouhodobě systém *Linux*, pravděpodobně z důvodu složitější distribuce a systém *Google Chrome OS*, který je ze zmiňovaných systémů nejmladší (Statcounter Global Stats, nedatováno).



Graf 2 - Zastoupení desktopových operačních systémů během posledních několika let (Statcounter Global Stats, nedatováno)

Prostředí mobilních telefonů nabízí primárně dva operační systémy s většinovým zastoupením, tedy *Android* a *iOS*. Dlouhodobě se větším počtem uživatelů, přibližně se zastoupením kolem 70 %, pyšní operační systém *Android* vyvíjený společností *Google*, který obdobně jako *Windows* využívá několik výrobců hardwaru pro svá zařízení. Konkurenční systém *iOS* určený pouze pro mobilní telefony společnosti *Apple* dosahuje zastoupení okolo 30 % (Statcounter Global Stats, nedatováno).



Graf 3 - Zastoupení mobilních operačních systémů během posledních několika let (Statcounter Global Stats, nedatováno)

3.2 Operační systémy

Po definování pojmu počítačové platformy je možné popsat jednotlivé nejpoužívanější operační systémy pro osobní počítače a mobilní zařízení se zaměřením na vývoj a distribuci aplikací.

3.2.1 Android

Android je operační systém s otevřeným zdrojovým kódem a využívající *Linuxové jádro*, jež je určený pro využití na mobilních zařízení s dotykovým ovládáním. Výrobci mohou systém volně upravovat a využívat na svých zařízeních při dodržování určitých podmínek. Za vývojem systému stojí v současné době společnost *Google* a standardizuje jej konsorcium *Open Hanset Alliance*. Android využívá pro vzhled svého uživatelského rozhraní designový standard, zvaný *Material Design*. Design uživatelského rozhraní je však dále ovlivňován nástavbami systému od vývojářů zařízení.

Aplikace pro *Android* lze vytvářet ve vývojovém prostředí *Android studio* v programovacím jazyce *Java*, či ve vývojovém prostředí *IntelliJ IDEA* v programovacím jazyce *Kotlin*. Pro vývoj je také potřeba mít nainstalovaný baliček *Software development kit* poskytující různé služby (Vávrů a Ujbányai, 2013, s. 15-24).

Primárním prostředím pro instalaci aplikačního softwaru v operačním systému *Android* je obchod s aplikacemi zvaný *Google play*. Zde mohou své aplikace publikovat vývojáři s platnou vývojářskou licencí. Aplikace při publikaci prochází automatizovaným schvalovacím procesem. Aplikační software lze však instalovat i na dalších alternativních obchodech s aplikacemi nebo i přímo z uložiště (Darwin, 2017, s. 701-708).

3.2.2 iOS

Uzavřený operační systém *iOS* od společnosti *Apple* je určený pro běh na mobilních telefonech *iPhone* s dotykovým ovládáním. Dříve byl určený i pro tablety *iPad*, pro ně však vznikla upravená verze systému, zvaná *iPadOS*. *Apple* pro vzhled svých operačních systémů využívá vlastní designový standard, který lze na internetu najít pod názvem *Apple Human Interface Guidelines*. Systém je postavený jako odlehčená verze desktopového operačního systému *macOS*. Jedná se tedy o systém postavený na operačním systému *Unix*, ale je navíc doplněn o možnosti dotykového ovládání (Posey, 2022).

Hlavní cestou tvorby aplikací pro operační systémy od společnosti *Apple* je pomocí vývojového prostředí *Xcode* a programovacích jazyků *Objective-C* a *Switft*. K instalaci

aplikací na uživatelská zařízení slouží obchod s aplikacemi, zvaný *AppStore*. U tohoto systému se jedná o jediný legální způsob, jak instalovat aplikace do operačního systému. Pro publikování aplikací skrze obchod *AppStore* platí přísné podmínky a aplikace jsou schvalovány skutečnou pověřenou osobou (Klosowski, 2014).

3.2.3 Windows

Windows je operační systém vyvíjený společností *Microsoft* určený pro osobní počítače. Pro všechna zařízení využívá jednotné uživatelského rozhraní postavené na designovém standardu *Fluent design*.

Aplikace lze vytvářet v mnoha programových jazycích a nástrojích. Microsoft pro tvorbu aplikací nabízí vývojové prostředí *Visual Studio* a programovací jazyky *C#*, *F#* a *Visual Basic*. Pro publikaci a instalaci aplikací nabízí *Windows* primárně *Microsoft Store*, lze však využívat téměř neomezeně i další způsoby (Ramel, 2021).

3.2.4 macOS

Dalším desktopovým operačním systémem je systém *macOS*. Jedná se o systém určený pro počítače od společnosti *Apple* postavený na operačním systému *Unix*, konkrétně jeho verzi *FreeBSD*.

Jak již bylo zmíněno u operačního systému *iOS*, pro tvorbu nativních aplikací lze využít programovací jazyky *Objective-C* a *Swift* a pro publikaci a instalaci aplikací nativní obchod s aplikacemi *AppStore*, či v případě systému *macOS* i jiné prostředky (Klosowski, 2014; Powalowski, 2020).

3.2.5 Linux

Dalším používaným operačním systémem pro počítače je *GNU/Linux*. Jedná se o open source operační systém, který lze libovolně používat k různým účelům. Skládá se z jádra zvaného *Linux* a z operačního systému *GNU*. V praxi se používá jako komplexní distribuce, tedy kolekce programů obsahujících *GNU*, jádro *Linux* a další programy. Příkladem takové distribuce je například operační systém *Ubuntu*, či *Fedora*. Použití tohoto operačního systému se nejvíce uchytilo u počítačových serverů.

Pro *Linux* lze vytvářet aplikace v mnoha nástrojích a lze je volně publikovat a získávat. Avšak stejně jako u ostatních operačních systémů, zde také jednotlivé distribuce nabízejí nativní obchody s aplikacemi (Javatpoint, nedatováno).

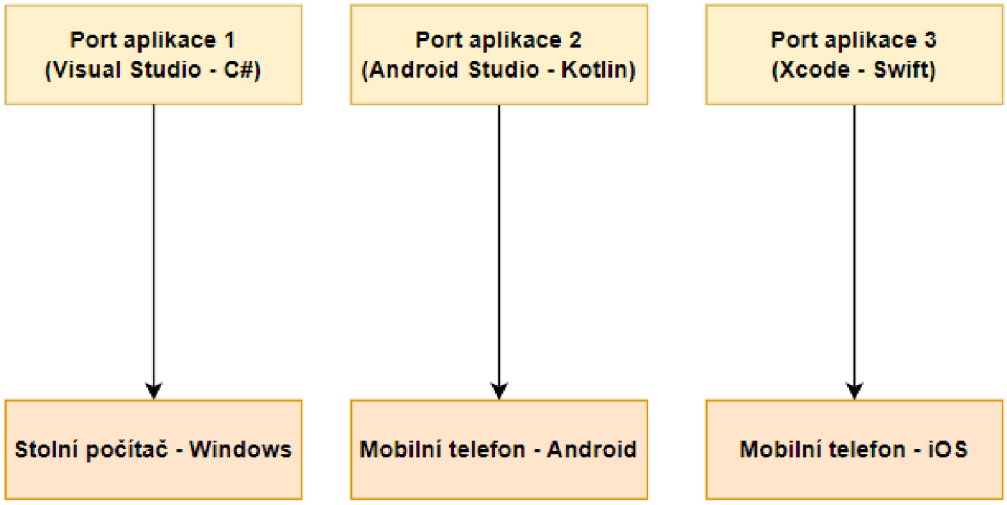
3.3 Dělení druhů aplikací dle přístupů k multiplatformnímu vývoji

Při plánování tvorby aplikace je mimo jiné potřeba určit i na jaké hardwarové a softwarové platformy má být aplikace vyvinuta. Tedy je potřeba říct, zda aplikace bude učena pouze pro telefony, či pouze pro počítače nebo pro obojí a pro jaké operační systémy. Toto určení záleží na několika faktorech, tedy na účelu aplikace, dostupných zdrojích, cíleném druhu uživatelů, zkušenostech vývojového týmu a tak dále. V mnoha případech je například požadováno cílit na co největší množství uživatelů, ti však používají různé platformy a je tak potřeba poskytovat aplikaci pro několik platforem zároveň. Právě problematikou, jak vytvářet aplikace, spustitelné pro více platforem najednou se zabývá oblast multiplatformního vývoje.

Přístupů, jak aplikovat multiplatformní vývoj vzniklo již několik. Každý tento přístup má své specifické metody a z toho i vyplívající vlastnosti, přístupy, ale i výhody a nevýhody. Pomocí přístupů multiplatformního vývoje lze tvořit nativní aplikace, webové aplikace, progresivní webové aplikace, multiplatformní aplikace a hybridní aplikace (Kotlin, 2022; LINFO, nedatováno).

3.3.1 Nativní aplikace

Nejvíce konzervativním přístupem k multiplatformnímu vývoji je přístup vývoje nativních aplikací. Vývoj nativní aplikace spočívá v běžné tvorbě aplikace, která je určena pro běh na jednom konkrétním operačním systému. Nejedná se tak v pravém slova smyslu o metodu určenou k tvorbě multiplatformních aplikací. Pokud chceme cílit na více operačních systémů, musíme pro každý tento operační systém vytvořit upravenou podporovanou verzi aplikace, která se v tomto případě označuje jako port. Každý operační systém má však své specifické vývojové prostředí, programovací jazyky a nástroje. Většinou je tak zapotřebí každou verzi aplikace vytvořit zcela od základů. Je tak například potřeba vytvořit verzi pro *Windows* ve vývojovém prostředí *Visual Studio* a v programovacím jazyce *C#*, *Visual Basic*, či *F#*, pro *Android* ve vývojovém prostředí *Android Studio* a v programovacím jazyce *Kotlin*, či *Java* a pro *iOS* a *macOS* ve vývojovém prostředí *Xcode* a v programovacím jazyce *Swift*, či *Objective-C*. Koncept tvorby několika portů nativní aplikace je ukázán na obrázku 1. Zde jsou vytvořeny tři porty, které jsou vyvinuty pomocí specifických technologií v závislosti na cílené zařízení a jeho operační systém (Kotlin, 2022).



Obrázek 1 - Koncept tvorby portů nativní aplikace [Vlastní zpracování dle textu (Kotlin, 2022)]

Každý přístup má své výhody a nevýhody. Většina výhod tohoto přístupu tkví v nuceném použití nativních programovacích jazyků, přístupů a nástrojů dané platformy. Mezi nejvýznamnější výhody nativního přístupu tvorby aplikací patří:

- **Optimalizace pro platformu** – nativní aplikace nabízí, díky dobré optimalizaci nativních nástrojů pro daný operační systém, vysoký výkon a optimální velikost aplikace
- **Vzhled aplikace** – každá platforma má svůj specifický vzhled a způsob ovládání, definovaný designovým systémem platformy, který by měly pro zjednodušení práce uživatele respektovat všechny aplikace. Nativní nástroje toto pravidlo vždy respektují a nativní aplikace tak vypadají stejně jako systémové aplikace daného operačního systému.
- **Přístup k funkcím platformy** – nativní nástroje umožňují přístup ke všem možným funkcionalitám dané platformy. Špatný přístup k funkcionalitám zařízení je jeden z nejběžnějších problémů ostatních vývojových přístupů.
- **Jednoduché nasazení aplikace** – nativní vývojová prostředí nabízí možnost tvorby instalačních balíčků aplikací, které lze následně vložit do nativních obchodů s aplikacemi.

Nevýhody tohoto přístupu jsou zapříčiněny zejména potřebou tvorby několika verzí aplikace pro různé platformy. Mezi tyto nevýhody tak můžeme zařadit:

- **Vysoká zdrojová náročnost** – tvorba několika verzí aplikace je jak finančně a časově náročná, tak i náročná na potřebné lidské zdroje.
- **Náročnost na management** – ve společnostech vhodně vznikají specializované týmy zaměřující se na jednu konkrétní platformu, které mezi sebou musí komunikovat. To má však nevýhody ve složitějším manažerském řízení.
- **Rozdílnost verzí** – při tvorbě více verzi aplikace hrozí, že se chování jednotlivých verzi bude časem rozcházet. Také tím vzniká větší prostor pro chyby (Kidecha, 2022).

Tento přístup je tak nejvhodnější pro své primární využití, tedy pro tvorbu aplikace pouze na jedinou platformu. Je však vhodný i v případech, kdy se tvoří aplikace podporující například pouze dvě platformy a je vyžadován vysoký výkon, specifický design pro dané platformy, či využití specifických funkcí daných platforem. Tento přístup je také vhodný ve firmách, kde již existují týmy specializované na dané platformy a není potřeba přecházet na nějaký nástroj pro tvorbu multiplatformních aplikací.

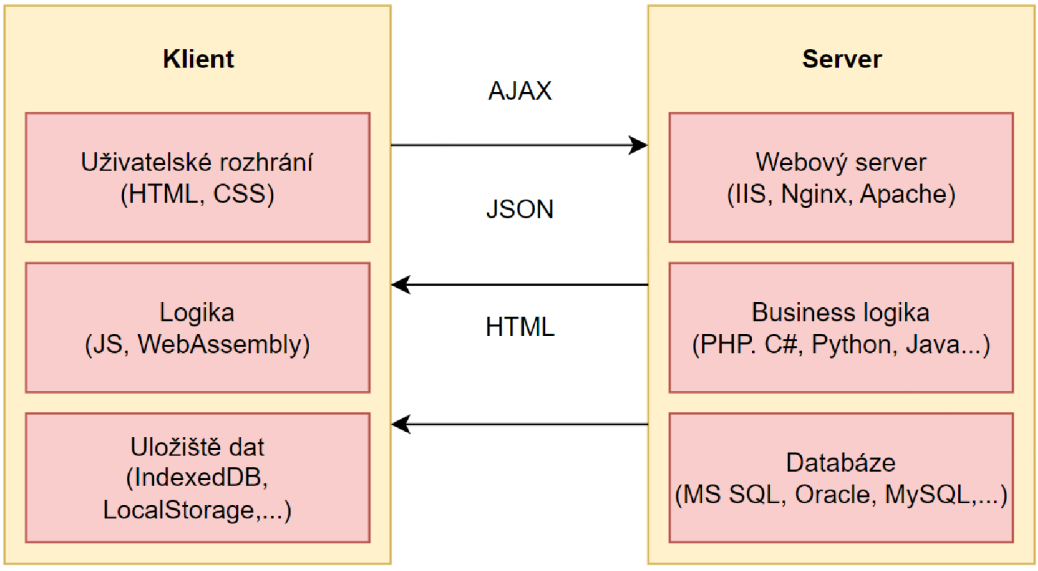
Pokud se například cílí na větší množství operačních systémů, je méně času a dalších zdrojů na tvorbu multiplatformní aplikace, či je zkrátka vhodnější udržovat mezi platformami jedinečný sdílený a také znovupoužitelný kód, je vhodné využít některý z dalších přístupů (Kotlin, 2022; Kidecha, 2022).

3.3.2 Webové aplikace

Jedním z prvních přístupů, který se pyšní velkou popularitou a prakticky umožňuje skutečnou tvorbu multiplatformních aplikací, respektive aplikací, které nejsou závislé pouze na jediném operačním systému je tvorba webových aplikací. Běh webové aplikace je omezen na prostředí a technologie webových prohlížečů, lze tak říct, že platformou aplikace z pohledu webové aplikace není operační systém, ale webový prohlížeč. Z pohledu tvorby multiplatformních aplikací je to výhodné, neboť webová aplikace může být spuštěna bez nutnosti dalších úprav na libovolném operačním systému, který umožňuje běh webových prohlížečů. Na rozdíl od nativních aplikací se webové aplikace neinstalují v prostředí operačního systému skrze například obchody s aplikacemi, ale na webový server a přistupuje se k nim přes webový prohlížeč a *URL* adresu jejich umístění v síti. Z toho vyplívá,

že je vhodnější jejich využití v případech, kdy několik uživatelů přistupuje k jedné instanci aplikace, neboť individuální instalace aplikace na webový server pro každého uživatele je komplikovaná. Jelikož uživatelé aplikaci neinstalují, je výhodou, že ani nemusejí instalovat aktualizace a používají vždy aktuální verzi. Obdobně jako nativní aplikace musejí používat nativní technologie platformy, musejí webové aplikace používat nativní technologie webových prohlížečů, tedy jazyky *HTML* a *CSS* pro strukturu, obsah a design, *JavaScript* a *WebAssembly* pro klientské zpracování dat a *Session Storage*, *Local Storage*, *Cookies* a *IndexedDB* pro klientské ukládání dat. Webová aplikace se však skládá kromě klientské části, která využívá tyto zmíněné technologie i ze serverové části, kde lze využít praktický libovolný programovací jazyk pro serverové zpracování dat a libovolnou databázi pro ukládání dat. Tyto dvě části spolu dále komunikují prostřednictvím *HTTP* protokolu a v případě komunikace od klienta k serveru, za pomocí technologie *AJAX*. Během komunikace si části vyměňují data v různých formátech jako je například *JSON*. Architektura webové aplikace je graficky zobrazena na obrázku 2.

Webové aplikace tak v mnoha případech nabízejí stejné možnosti jako nativní aplikace, i přes jejich omezení (Stack Path, nedatováno).



Obrázek 2 - Architektura webové aplikace [Vlastní zpracování dle (Patel, 2020)]

Jako výhody webových aplikací z pohledu vývoje multiplatformních aplikací lze považovat:

- **Závislost na webovém prohlížeči** – díky této závislosti jsou webové aplikace dostupné na velkém množství operačních systémů
- **Jediná verze aplikace** – značnou výhodou proti přístupu tvorby nativních aplikací je potřeba tvorby pouze jediné verze aplikace. Vývojáři tak mohou vytvářet jediný sdílený a znovupoužitelný kód.
- **Nízká zdrojová náročnost** – vývoj webových aplikací patří mezi jeden z nejméně zdrojově náročných přístupů tvorby aplikací. Webové aplikace jsou jak finančně i časově málo náročné, tak i málo náročné na lidské zdroje. Vývojáři totiž vytvářejí aplikace pouze pomocí webových technologií a nemusí znát jednotlivé nativní technologie operačních systémů.

Stejně jako nativní aplikace mají i webové aplikace své nevýhody, mezi ně patří:

- **Omezení webového prohlížeče** – prostředí webového prohlížeče nabízí webovým aplikacím značně nižší výkon, než je dostáván nativním aplikacím. Nevýhodou je také potřeba připojení k webovému serveru, na kterém je webová aplikace nainstalována, což u aplikací hostovaných na internetu znamená potřebu internetového připojení.
- **Webový vzhled aplikací** – prostředí webových prohlížečů nerespektuje designový systém hostujícího operačního systému, tedy každá webová aplikace je prezentována na všech operačních systémech stejně. Jelikož toto prostředí nabízí vlastní technologie s velkými možnostmi customizace, vypadají a ovládají se webové aplikace často velmi odlišně. U webových aplikací je také potřeba celou strukturu, vzhled a ovládání navrhnout a vytvořit, přičemž u nativních aplikací bývá toto vytvořeno automaticky. V tomto případě je tak například poměrně složité vytvořit vhodný design pro aplikaci, která bude používána jak na počítačích, tak na telefonech. Jednotný design napříč zařízeními lze však v některých případech brát i jako výhodu.

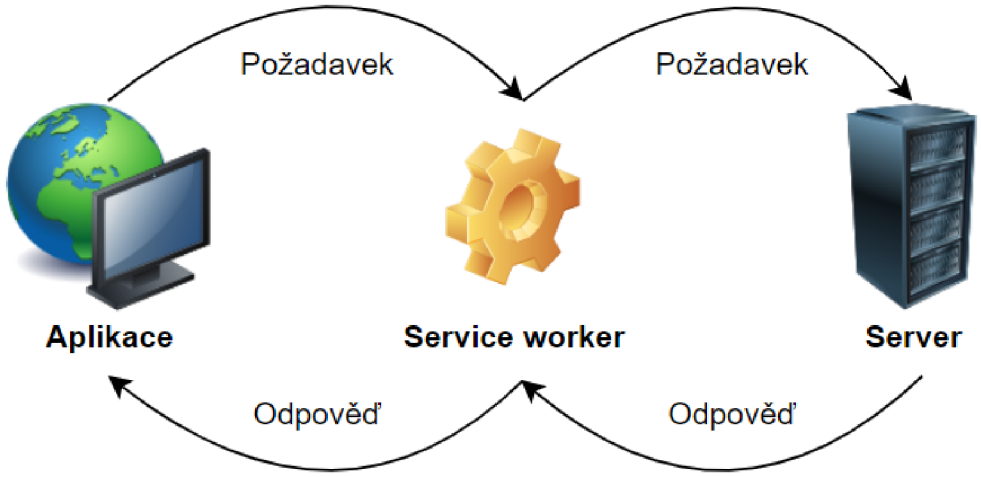
- **Nemožný přístup k funkcím zařízení** – jelikož jsou funkcionality webových aplikací omezeny pouze na prostředí webových prohlížečů, nemohou prakticky přímo samostatně přistupovat k funkcím zařízení a jeho operačního systému. Jedinou možností, jak přistupovat k funkcím zařízení ve webové aplikaci je skrz komunikaci s doprovodnou nativní aplikací.
- **Složitá instalace a přístup k aplikaci** – webové aplikace lze instalovat pouze na webové servery a lze k nim přistupovat pouze přes webový prohlížeč a *URL* adresu jejich umístění v síti, což může přinášet určitou komplikovanost a omezení (Stack Path, nedatováno).

3.3.3 Progresivní webové aplikace

Jelikož webové aplikace nabízejí dobré možnosti a výhody jak pro vývojáře, tak pro uživatele, staly se pro obě strany velmi populárními. Aby však mohly plnit stejné funkce a nabízet stejné možnosti jako nativní aplikace, bylo potřeba vyřešit některé jejich problémy. Tím bylo zejména omezení webových prohlížečů, působící na webové aplikace. Za tímto účelem byla vytvořena rovnou dvě možná podobná řešení, která kombinují nativní a webové aplikace a využívají tak výhody obou přístupů. Těmito řešeními byly progresivní webové aplikace a hybridní aplikace.

Progresivní webové aplikace, zkráceně *PWA* jsou jednoduše řečeno klasické webové aplikace, obohacené o některé další funkce přinášející možnosti nativních aplikací. *PWA* je tak možné například z pohledu uživatele instalovat do operačního systému a jednoduše k nim přistupovat, pracovat s nimi bez připojení k internetu, či přístupu k webovému serveru, na kterém jsou nainstalovány, posílat nativní notifikace operačního systému, přistupovat k některým dalším možnostem zařízení a operačního systému (čtečka otisku prstů, fotoaparát, …) a další. Také nabízejí o něco vyšší výkon než webové aplikace.

Technicky lze rozšířit novou, či již existující klasickou webovou aplikaci na *PWA* přidáním dvou souborů. Prvním souborem je soubor *manifest* typu *JSON* sloužící jako metadata souboru popisující webovou aplikaci, aby vypadala a fungovala více jako nativní aplikace. Druhým souborem je skript napsaný v programovacím jazyce *JavaScript*, zvaný *Service worker* zprostředkovávající propojení *PWA* s prohlížečem a synchronizaci dat. Na obrázku 3 je zobrazena architektura progresivní webové aplikace. *Service worker* je společně s klientskou částí aplikace spouštěn v zařízení uživatele a vystupuje jako další komunikační vrstva mezi klientskou a serverovou částí aplikace (Koďousková, 2021).



Obrázek 3 - Architektura progresivní webové aplikace [Vlastní zpracování dle (Koďousková,2021))]

Jelikož jsou *PWA* běžnými webovými aplikacemi, instalují se na webové servery. Ovšem uživatel si může tuto aplikaci také nainstalovat zvlášť na své zařízení a díky tomu k ní dále přistupovat téměř jako k běžné nativní aplikaci. *PWA* je možné pomocí dalších úprav poskytovat uživatelům k instalaci skrze nativní obchody s aplikacemi. Běžnějším a jednodušším způsobem je však poskytovat je uživatelům k instalaci rovnou ve webovém prohlížeči. V mnoha případech jsou *PWA* aplikace postavené jako *Single page aplikace*, zkráceně *SPA*, což je druh webových aplikací, které mají všechen obsah na jediné stránce a aktualizují pouze potřebný dynamicky měnící se obsah. Tento přístup tak značně zrychluje odezvu aplikace.

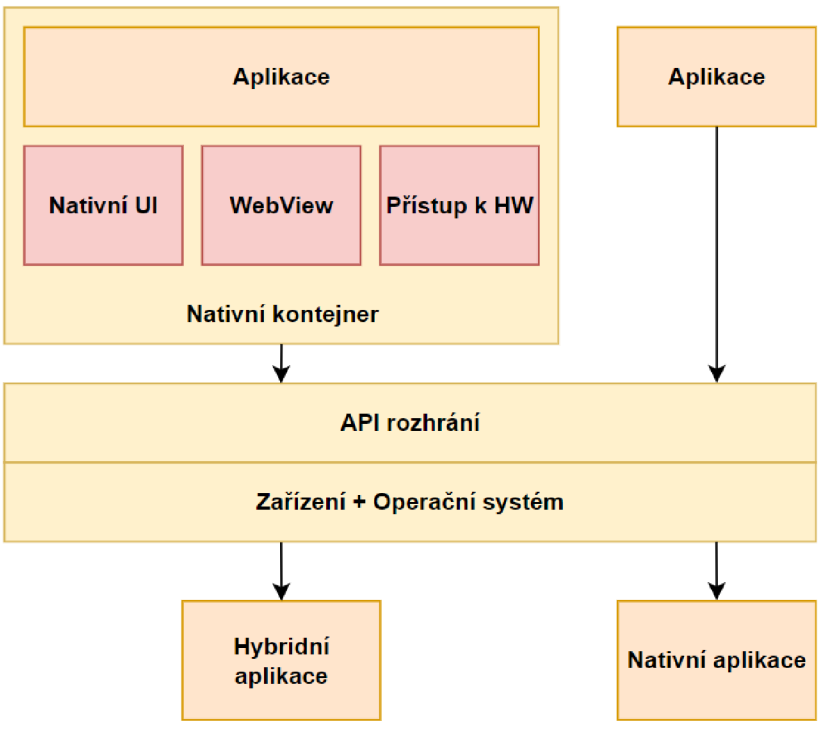
Jelikož se v základu jedná o klasickou webovou aplikaci obohacenou o další funkce, přejímá všechny její výhody a řeší některé její problémy dodáním některých funkcí nativních aplikací. Oproti klasické webové aplikaci u *PWA* je vyřešeno omezení webových prohlížečů, je umožněn přístup k některým nativním funkcím a uživatelský přístup k aplikaci je také snazší.

Oproti nativním aplikacím jsou progresivní webové aplikace stále podstatně méně výkonné a hůře optimalizované, nemají kompletní přístup ke všem možnostem zařízení a musejí se instalovat na webové servery. Ačkoliv mají i *PWA* klasický webový design, působí o trochu více jako nativní aplikace (Koďousková, 2021; Kotlin, 2022).

3.3.4 Hybridní aplikace

Jak již bylo zmíněno, dalším druhem aplikace, která kombinuje vlastnosti webové a nativní aplikace je hybridní aplikace. V případě hybridní aplikace dochází ještě k těsnějšímu propojení mezi nativní a webovou aplikací než u progresivně webové aplikace. V tomto případě je také aplikace tvořena zejména webovou aplikací, avšak vývojář má téměř neomezené možnosti přístupu k nativním funkcionalitám (Kotlin, 2022).

Technicky je tento přístup implementován značně jinak než u *PWA*. V tomto případě dochází k tvorbě nativní aplikace, obsahující speciální komponentu, nazývanou *WebView*, sloužící jako odlehčený minimalistický webový prohlížeč. Úkolem komponenty *WebView* je zobrazovat vně nativní aplikace vytvářenou webovou aplikaci. Vytvářená webová aplikace je tak v tomto případě přímo uložena a hostována v nativní aplikaci. Nepotřebuje tak například připojení k internetu. Architektura hybridní aplikace a její porovnání s architekturou nativní aplikace jsou zobrazeny na obrázku 4. Na obrázku je ukázáno, že architektura hybridní aplikace, jenž je zobrazena v levé části obrázku je značně složitější než architektura nativní aplikace, která je zobrazena v pravé části obrázku, neboť musí obsahovat několik dalších komponent jako je například zmiňované *WebView* (Shiotsu, 2021).



Obrázek 4 - Porovnání architektur hybridních a nativních aplikací [Vlastní zpracování dle (Shiotsu, 2021)]

Frameworky pro vývoj hybridních aplikací tedy umožňují jak tvorbu webových aplikací, tak tvorbu nativních aplikací pro mnoho operačních systémů, včetně jejich instalačních balíčků, které lze dále vložit do obchodů s aplikacemi. Některé nástroje dokonce umožňují využívat jak hybridní přístup, tak multiplatformní přístup, který bude popsán dále. To znamená, že v takových nástrojích lze například vytvářet i další nativní komponenty mimo komponentu *WebView*. Zajímavostí také je, že v případě, kdy framework využívá webové technologie, ale místo vykreslení obsahu kódu vně komponenty *WebView*, vykresluje tento obsah jako nativní komponenty dané platformy pomocí nativních *API*, považuje se výsledná aplikace za aplikaci multiplatformní, nikoliv hybridní. Je tak vidět, že hranice mezi těmito přístupy v podstatě mizí. Mezi technologie využívající tento přístup patří například *.NET MAUI*, *Apache Cordova* a *Ionic* (Kidecha, 2022).

Hybridní aplikace opět přejímají všechny výhody webových aplikací a mnoho výhod nativních aplikací.

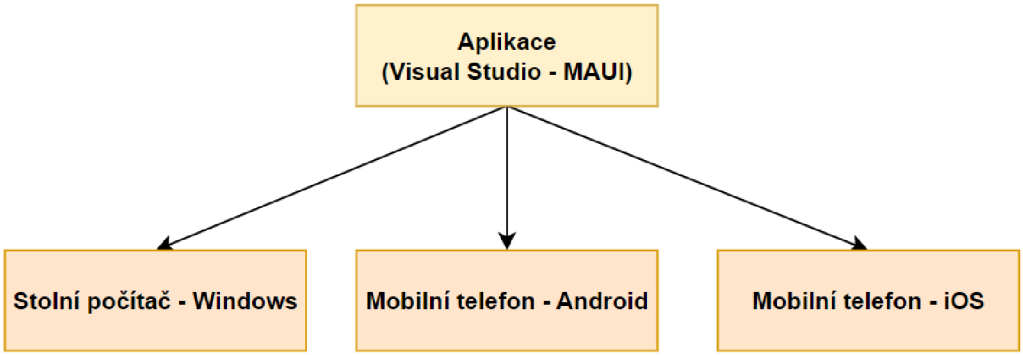
Oproti nativním aplikacím jsou i hybridní aplikace stále podstatně méně výkonné, složitěji se ladí a jsou objemově větší, neboť obsahují multiplatformní funkce a také komponentu *WebView*. Mají však většinou dobrý přístup k nativním funkcím zařízení a instalují se stejně jako nativní aplikace. Obsah komponenty *WebView* je sice webová aplikace, ale mimo ni se jedná o nativní aplikaci, díky čemuž hybridní aplikace nabízí opět o trochu větší dojem, že se jedná o nativní aplikaci (Shiotsu, 2021).

3.3.5 Multiplatformní aplikace

Poslední dobou získává značnou popularitu i poslední přístup, vývoj multiplatformních aplikací. V tomto případě není multiplatformní chování tvořeno pomocí webových technologií jako u předchozích třech přístupů, ale pomocí nativních technologií jako v prvním případě.

Přístup vývoje multiplatformních aplikací je založen na tvorbě jedné aplikace se sdíleným a znovupoužitelným kódem v určitém multiplatformním frameworku, pomocí jeho programovacího jazyka a nástrojů. Multiplatformní framework následně překládá a vykresluje vytvořený multiplatformní kód pomocí nástrojů podporovaných platforem do nativního kódu a komponent, čímž vytváří nativní aplikace vytvořené pomocí nativních technologií. Framework rovnou také vytváří i instalační balíčky, které lze dále publikovat v obchodech s aplikacemi. Podporovanými platformami zde mohou být jak různé operační systémy, tak i webový prohlížeč, v případě že to daný framework umožňuje. Koncept tvorby

multiplatformní aplikace znázorňuje obrázek 5, kdy dochází k tvorbě jediné multiplatformní aplikace, ze které lze dále sestavit nativní aplikace pro určené platformy (Kotlin, 2022; Kidecha, 2022).



Obrázek 5 - Koncept multiplatformní aplikace [Vlastní zpracování dle text (Kidecha, 2022)]

Mezi frameworky pro tvorbu multiplatformních aplikací patří například *Flutter*, *React Native*, *.NET MAUI* a *.NET Xamarin*. Stejně jako hybridní aplikace i multiplatformní aplikace nabízejí téměř všechny možnosti zařízení. Lze vytvářet i platformě specifický kód, který se aplikuje pouze pro danou platformu. Tuto funkcionalitu nabízí i mnoho hybridních frameworků (Kotlin, 2022; Kidecha, 2022).

Tento přístup má také své výhody a nevýhody v porovnání s přístupem tvorby nativních aplikací a s přístupy vývoje multiplatformních aplikací založenými na využití webových aplikací. Primární výhodou je právě tvorba sdíleného kódu, který lze dále rozšiřovat o platformě specifický kód. Vývoj je tak opět o dost méně náročný jak na zdroje, tak i na správu než v případě nativních aplikací, i když je potřeba, aby vývojář znal programovací jazyk a nástroje daného multiplatformního frameworku. Oproti přístupům založeným na webových aplikací, umožňuje tento přístup možnosti reakce na danou platformu. Výhodou mnoha multiplatformních frameworků je také skutečnost, že vykreslují obsah jako nativní komponenty, které respektují designový systém platformy.

Jako nevýhoda se opět nejčastěji udává větší objemová velikost aplikací a menší výkon oproti nativním aplikacím. V tomto případě je však výkonový rozdíl většinou o dost menší, nebo je dokonce výkon stejný (Kotlin, 2022).

3.4 Nástroje pro multiplatformní a hybridní vývoj aplikací

Na základě vybraného přístupu multiplatformního vývoje pro tvorbu multiplatformní aplikace je potřeba zvolit vhodné vývojářské nástroje a technologie. Jelikož tato práce zkoumá technologii *.NET MAUI* vytvářející aplikace dle přístupů hybridního, či multiplatformního vývoje, popisuje tato kapitola některé další nástroje využívající alespoň jeden z těchto přístupů. Samotná technologie *.NET MAUI* je popsána později.

3.4.1 Flutter

Flutter je populární framework od společnosti *Google*, vytvořený v roce 2017, určený pro tvorbu multiplatformních aplikací, spustitelných jak na operačních systémech určených pro telefony a počítače, tak i ve webových prohlížečích (Kotlin, 2022).

Flutter využívá pro tvorbu aplikací programovací jazyk *Dart*, který je také vyvíjen společností *Google*. Jazyk byl původně vytvořen jako alternativa k programovacímu jazyku *JavaScript* pro klientské zpracování dat ve webových aplikací. Pro svůj původní účel se však příliš neuchytil. Svou pozornost si získal až při vzniku frameworku *Flutter*, jako programovací jazyk určený pro tvorbu multiplatformních aplikací. Využití tohoto jazyka přináší mnoho výhod, neboť se jedná o moderní a objektově orientovaný programovací jazyk, podporující jak *JIT* kompilaci, tak i *AOT* kompilaci. *AOT* kompilace v tomto případě přináší vysoký výkon pro nativní aplikace a *JIT* kompilace například klíčovou funkcionalitu *hot-reload* umožňující okamžité překreslení aplikace po změně kódu, aniž by se musela celá znovu kompletně překládat (Windmill, 2019, s. 3-12).

Flutter pro vykreslování obsahu využívá svůj vlastní vykreslovací engine, jež podporuje designový systém *Material Design system*. Kromě toho framework obsahuje mnoho UI komponent, testovací framework a mnoho dalších nástrojů.

Pomocí tohoto frameworku byly vytvořeny například aplikace *eBay* a *Google Pay* (Kotlin, 2022).

3.4.2 React Native

Odlišný přístup pro tvorbu multiplatformních aplikací poskytuje framework *React Native* od společnosti *Meta Platforms*, vytvořený v roce 2015. Framework je určen pouze pro tvorbu aplikací na mobilní operační systémy *Android* a *iOS*. Vytvořený kód lze však dále použít i pro webovou aplikaci, neboť framework využívá *JavaScriptový* framework *React* a další webové technologie (Kotlin, 2022).

I přestože framework využívá webové technologie, vykresluje a zpracovává svůj obsah nativně a nejedná se tak o framework pro tvorbu hybridních aplikací. Využití webových technologií bylo u frameworku zvoleno zejména kvůli jejich velké popularitě mezi vývojáři. Jelikož však hybridní aplikace s komponentou *WebView* nabízejí horší výkon než nativní aplikace z důvodu náročného zpracovávání *DOM* modelu, byl u frameworku vybrán přístup, kdy JavaScriptový kód komunikuje s nativním kódem pomocí takzvaného *JavaScriptového mostu*. Takový přístup přináší značně vyšší výkon než hybridní přístup tvorby aplikací, neboť jeho zpracování není zatížené komponentou *WebView* a *DOM* modelem. Ani *JavaScriptový most* však nenabízí ideální výkon právě z důvodu náročnější komunikace *JavaScriptového* a nativního kódu. V případě potřeby využití co nejvyššího výkonu je proto vhodnější využít nějaký multiplatformní framework překládající svůj kód přímo do nativního kódu jako je například *Flutter* (Windmill, 2019, s. 8-10).

Nativní vykreslování uživatelského rozhraní výhodně nabízí zobrazení komponent s ohledem na designové systémy daných platforem. *React Native* také obsahuje prvek zvaný *Fast Refresh*, který funguje podobně jako *Hot Reload* frameworku *Flutter*

S využitím *React Native* byly vytvořeny aplikace *Skype*, *Shopify* a další (Kotlin, 2022).

3.4.3 Ionic

Kromě frameworků pro tvorbu multiplatformních aplikací je vhodné také zmínit nějakého zástupce frameworků pro tvorbu hybridních aplikací. Takovým frameworkem je například *Ionic*, vytvořený v roce 2013 pro tvorbu mobilních a desktopových aplikací. Pro tvorbu obsahu jsou tedy využity webové technologie s možností využití *JavaScriptových* frameworků *Angular*, *React* a *Vue*.

Zajímavou součástí frameworku je designový systém *SasS UI*, pluginy *Cordova* a *Capacitor* a vlastní vývojové prostředí *Ionic Studio* (Kotlin, 2022).

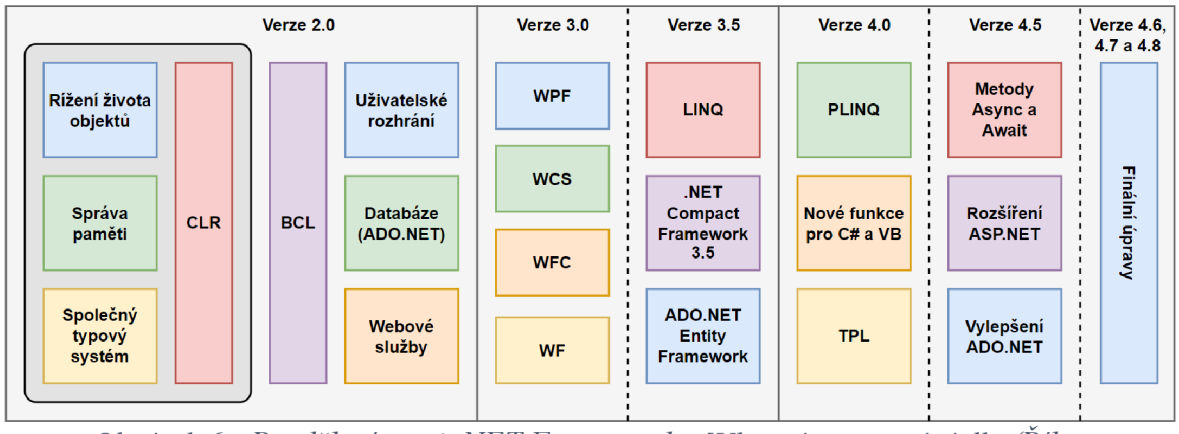
3.5 .NET platforma

Před konečným uvedením frameworků *.NET MAUI* a *Blazor* je nutno představit platformu *.NET*, do které právě i tyto frameworky spadají. *.NET* platforma je velkou sadou nástrojů, patří sem různé technologie, knihovny a frameworky pro tvorbu aplikací, programovací jazyky, vývojová prostředí a nástroje pro spolupráci. Technologie *.NET* platformy se využívají prostřednictvím její implementací, respektive vydání (Voborník, 2022).

3.5.1 Implementace

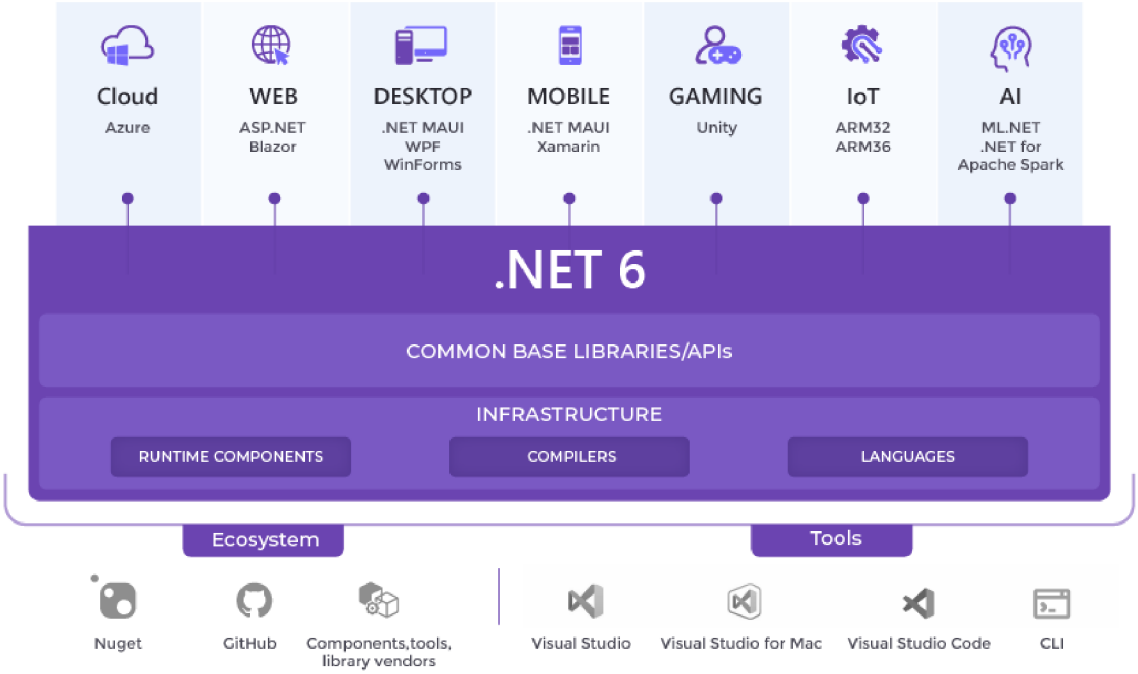
Historicky první implementací *.NET* platformy je *.NET Framework*. Ten měl být původně nezávislý na platformách, z důvodu pozdějších rozhodnutí však byl cílen pouze pro systém *Windows*. Jeho vývoj byl již zastaven, ovšem framework je stále podporovaný.

.NET Framework byl vyvinut jako robustní řešení spjaté se systémem *Windows*, se kterým se i aktualizoval a byl do své verze 4.5.2 jeho oficiální součástí (Price, 2021, s. 10). Jak je vidět na obrázku 6, verze 1.0 a 2.0 přinesli naprostý základ frameworku včetně překladače *CLR* a základní knihovny tříd pro tvorbu aplikací *BCL*. Verze 3.0 obohatila portfolio formulářových aplikací o technologii *WPF*. Následující sestavení 3.5 a 4.0 uvedli významný dotazovací jazyk *LINQ*. Sestavení 4.5 opět rozšířilo portfolio formulářových aplikací o technologii *UWP* a vývoj frameworku byl ukončen finální verzí 4.8 (Říha, nedatováno).



Obrázek 6 - Rozdělení verzí .NET Frameworku [Vlastní zpracování dle (Říha, nedatováno)]

Jako nástupce implementace *.NET Framework* vznikl nejnovější a aktuálně vyvíjený open source framework *.NET*, původně zvaný *.NET Core*. Jeho ekosystém je ukázán na obrázku 7. Jedná se konečně o multiplatformní framework pro tvorbu programů na mnoho systémů, který podporuje podobné portfolio technologií jako *.NET Framework*. Oproti svému předchůdci se liší koncepcí architektury, neboť jí tvoří malé modulární jádro umožňující přidávání dalších knihoven. Také již není tak úzce spjatý s operačním systémem a nemusí tak do něj být přímo nainstalován (Price, 2021, s. 11-12).



Obrázek 7 - Přehled součástí .NET 6 (Čápka, nedatováno)

Kromě implementací *.NET* platformy vytvořených společností *Microsoft*, také vznikla open source implementace *Mono*. Implementace vznikla jako alternativa v té době k *.NET Frameworku* a umožňovala užití *.NET* platformy mimo systém *Windows*, například v *Linuxových systémech*. Framework umožnil například vznik významných technologií *Xamarin* a *Unity*. V prostředí multiplatformního vývoje však již bylo přímé využití *Mona* v mnoha případech vytlačeno multiplatformní implementací *.NET*, které jej ve své architektuře samo využívá a přebírá některé jeho prvky (Price, 2021, s. 10).

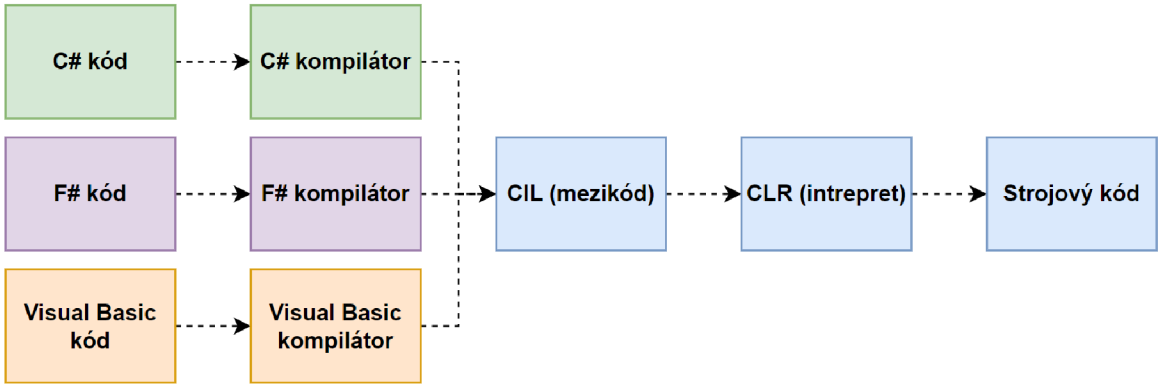
Speciálním případem implementace *.NET* platformy je *.NET Standard*. V tomto případě, jak již z názvu vyplívá se jedná pouze o formální standard a nelze jej samostatně využít pro tvorbu aplikací. Standard vznikl ve své době za účelem udržení kompatibility mezi instancemi *.NET* platformy *.NET Framework*, *.NET* a *Xamarin*, které se od sebe odlišovaly, co se týče možností a omezení i přesto, že se jedná o implementace jedné

platformy. Jedná se tak o nejuniverzálnější typ pro sdílení kódu mezi implementacemi *.NET* platformy. Od vzniku implementace *.NET 6* se snížil význam standardu, neboť vznikla jednotná implementace podporující všechny platformy obsahující jednotnou knihovnu *BCL* a dva *CLR* překladače. Prvním překladačem je *CoreCLR* určený pro servery a desktopy a druhým je *Mono* učený pro mobilní zařízení a webové prohlížeče. Standard však má stále význam pro udržení kompatibility s *.NET Frameworkem* (Price, 2021, s. 15-16).

3.5.2 Programovací jazyky

Primárním jazykem platformy *.NET* je *C#*. Jedná se o moderní, objektově orientovaný a mnohoúčelový programovací jazyk, založený na jazycích *C++* a *Java*. Jeho alternativou je funkcionální jazyk *F#* a jazyk *Visual Basic* (Voborník, 2022). Tyto jazyky spojuje stejná metoda zpracování pro zajištění multiplatformního využití, vykreslená na obrázku 8. Kód napsaný v těchto jazycích je nejprve přeložen kompilátorem (*Roslyn* v případě jazyka *C#*) do mezikódu *CIL* neboli *Common Intermediate Language*. Mezikód je při spuštění programu překládán do strojového kódu platformy pomocí virtuálního stroje *CLR*, respektive *Common Language Runtime* (Price, 2021, s. 17).

Mimo tyto základní jazyky platformy je možné využívat i jazyky *C++*, speciální implementaci jazyka *Python* zvanou *IronPython* a další (Voborník, 2022).



Obrázek 8 - Metoda zpracování programovacích jazyků .NET platformy [Vlastní zpracování dle (Price, 2021, s. 17)]

3.5.3 Vývojová prostředí

Důležitou součásti platformy jsou vývojová prostředí pro tvorbu a vývoj programů. Mezi tato prostředí se řadí komplexní vývojové prostředí *Visual Studio* určené pro systém *Windows* a vydávané v edicích *Community*, *Professional* a *Enterprise*. Novinkou je vývojové prostředí pro systém *macOS*. Poslední součástí je multiplatformní textový editor *Visual Studio Code* s možností instalace obrovské škály pluginů (Voborník, 2022).

3.5.4 Nástroje pro spolupráci

Další součástí platformy jsou nástroje pro spolupráci. Jedná se o různé nástroje umožňující spolupráci vývojářů při vývoji programů. První z těchto nástrojů je služba *Azure DevOps* určená pro sdílení a souběžnou týmovou práci na zdrojových kódech. Jde o obdobu technologie *Git* podporující verzování, dokumentaci, testování a další služby. Součástí portfolia platformy je také služba *GitHub* určená pro sdílení a spolupráci na zdrojových kódech. Služba je zejména známá pro spolupráci na open source projektech jako jsou *Linux*, *React*, *Bootstrap*, *VS Code* a mnoho dalších. Významnou součástí je také balíčkovací systém *Nuget* sloužící pro šíření hotových komponent a funkcionalit. Novinkou je služba *Live Share* pro vzdálenou spolupráci na zdrojovém kódu (Voborník, 2022).

3.5.5 Technologie pro vývoj desktopových aplikací

Platforma nabízí velkou škálu technologií pro tvorbu různých aplikací, které nyní budou představeny. Prvním typem aplikací, které si představíme jsou desktopové aplikace. Jejich nejjednodušší podobnou jsou konzolové aplikace. Tyto aplikace nemají grafické uživatelské rozhraní, ale pouze textové rozhraní a nejčastěji se využívají pro tvorbu bezobslužných aplikací běžících na pozadí.

Pokročilejším druhem desktopových aplikací s grafickým uživatelským rozhraním jsou aplikace formulářové. Jejich tvorbu umožňuje platforma ve třech různých technologických provedení. Nejstarší technologií s již zastaralým grafickým rozhraním jsou *Windows Forms*. Výkonný kód i uživatelské rozhraní je tvořeno jazykem *C#*. Pro jednodušší tvorbu UI je však k dispozici *WYSIWYG* editor umožňující vkládání a úpravu komponent, na základě kterého se automaticky generuje *C#* kód popisující uživatelské rozhraní. Technologie je úzce spjatá se systémem *Windows* a v případě užití *.NET Frameworku* chybí některé moderní designové funkce jako dotykové ovládání.

Nástupcem je technologie *WPF*, tedy *Windows Presentation Foundation*. Oproti *Windows Forms* je hlavní inovací obdobně jako u webového vývoje oddělení technologií pro tvorbu výkonného kódu a tvorbu *GUI*, které je nyní tvořeno pomocí jazyka *XAML*. Značkovací jazyk *XAML* je obdobou jazyka *HTML*, postavený na bázi *XML*.

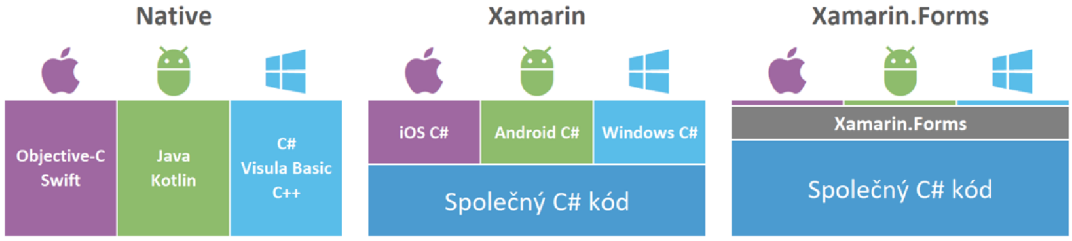
Nejmodernější způsob tvorby formulářových aplikací nabízí technologie *UWP*. Stejně jako *WPF* využívá *C#* pro tvorbu výkonného kódu a *XAML* pro tvorbu *GUI*, ten se však trochu liší oproti *WPF*. Primární inovací proti *WPF* je možnost tvorby univerzálních aplikací určených pro všechna zařízení, na kterých je nainstalovaný systém *Windows* (Počítače, *Xbox*, Telefony, *IoT*, …). Vytvářené aplikace z toho důvodu musejí mít responzivní design měnící své rozložení na základě rozlišení zařízení (Voborník, 2022).

3.5.6 Technologie pro vývoj webových aplikací

Dalším typem aplikací podporovaných v rámci platformy jsou webové aplikace, které je opět možné tvořit několika technologickými způsoby. Nejstarší technologií je *ASP.NET Web Forms*, která nabízela podobný přístup jako desktopové formulářové aplikace a většina aplikací se zpracovávala na straně serveru. Další variantou je *ASP.NET (Core) MVC*, kde došlo k většímu rozdělení vykonávání aplikace mezi klientskou a serverovou část a využití enginu *Razor*. Nejnovějším přístupem je využití technologie *Blazor*, která je popsána v nadcházejících kapitolách (Price, 2021, s. 15).

3.5.7 Technologie pro vývoj mobilních a multiplatformních aplikací

Platforma pro tvorbu běžných univerzálních aplikací mimo desktopová a webová řešení umožňuje ještě tvorbu mobilních aplikací, či dokonce i tvorbu hybridních a multiplatformních aplikací. Tvorbu nativních mobilních aplikací umožňuje technologie *Xamarin* vycházející z *Mona*. Již *Xamarin* obsahoval prvky pro sdílení kódu v rámci aplikace mezi systémy *Android* a *iOS*, konkrétně se jednalo o sdílení business logiky a *GUI* napsaného pomocí *XAML*. Multiplatformní možnosti technologie *Xamarin* podstatně obohatila nastávající technologie *Xamarin.Forms*, která také přidala mezi podporované systémy *Windows* prostřednictvím technologie *UWP*. Porovnání těchto dvou technologií a nativního vývoje je zobrazeno na obrázku 9. Frameworky *Xamarin.Forms* a *Xamarin* jsou předchůdci frameworku *.NET MAUI* (Baptista a Abbruzzese, 2022, s. 490-496).



Obrázek 9 - Porovnání technologií nativního vývoje, Xamarin a Xamarin.Forms (Voborník, 2022)

Mezi výhody frameworku se řadí adaptace nativních *UI* komponent dle platformy a stejně jako v případě frameworku *Flutter* funkce *Hot Reload*, která je však omezena pouze na překreslování uživatelského rozhraní. Nedokáže tedy reagovat na změny výkonného kódu (Baptista a Abbruzzese, 2022, s. 490-496).

Alternativou frameworků *Xamarin.Forms* a *.NET MAUI* v rámci platformy *.NET* jsou dva frameworky třetích stran *Uno Platform* a *Avalonia* umožňující tvorbu multiplatformních aplikací pro *Windows*, *macOS*, *Linux*, *iOS*, *Android*, a dokonce i webové prohlížeče prostřednictvím standardu *WebAssembly*. Zajímavé je technické provedení frameworku *Uno Platform*, který pro aplikace na *Windows*, *macOS*, *iOS* a *Android* využívá *Xamarin*, nikoliv tedy *Xamarin.Forms*. Pro webové aplikace využívá *Mono-WASM* běhové prostředí podobně jako *Blazor* a pro *Linux* využívá techonologii *Skia* pro kreslení uživatelského prostředí (Price, 2021, s 545-546).

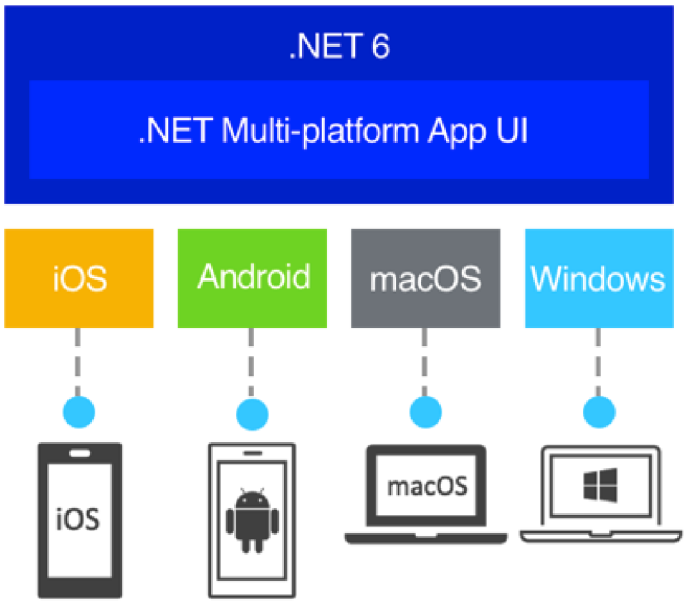
3.5.8 Ostatní vývojové technologie

Kromě desktopových, webových, mobilních a multiplatformních řešení úzce souvisejících s technologiemi *.NET MAUI* a *Blazor*, nabízí platforma ještě další kategorie aplikací. Jednou z těchto dalších kategorií je tvorba her. Pro tvorbu 2D a 3D her byl původně vytvořen *XNA* framework, ten byl však časem zrušen. Jako jeho náhrada slouží fanouškovský projekt *MonoGame*, založený na stejných principech. Další technologií pro tvorbu her je robustní herní engine *Unity* od společnosti *Unity Technologies*, podporující velké množství platforem. *Unity* obsahuje komplexní *WYSIWYG* editor a pro skripty využívá jazyk *C#* a *Visual Studio*.

Dále lze v rámci platformy používat technologie umělé inteligence (*ML.NET*) a vyvíjet aplikace pro Cloud a *IoT* zařízení (Voborník, 2022).

3.6 .NET MAUI

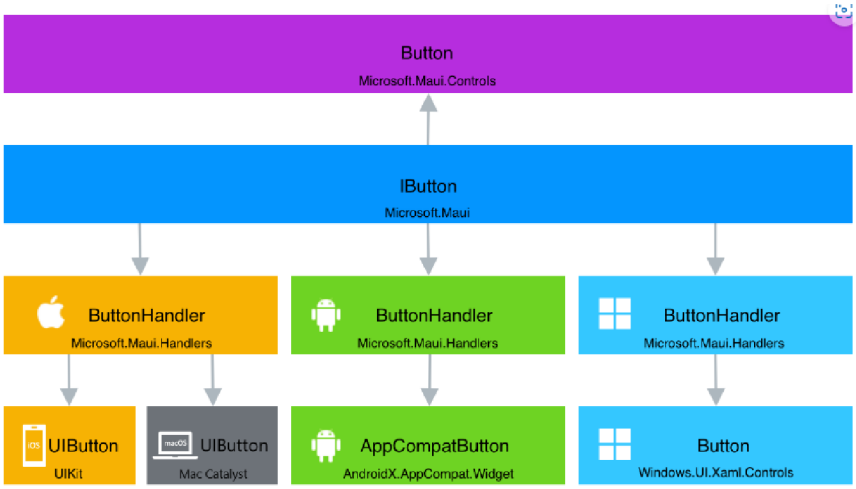
Po předešlém představení platforem, přístupů vývoje a *.NET* platformy je konečně možné popsat open source framework *.NET MAUI* neboli *Multi-platform App UI*, umožňující v rámci *.NET* platformy tvořit multiplatformní a hybridní aplikace pro desktopová a mobilní zařízení. Koncept vývoje multiplatformních aplikací za pomoci frameworku *.NET MAUI* lze vidět na obrázku 10. Oficiálně podporované platformy jsou *Windows*, *macOS*, *Android*, *iOS* a *Tizen* s podporou od společnosti *Samsung*. Pro *Linuxové systémy* zatím existuje pouze neoficiální komunitní podpora. Jak již bylo zmíněno, jedná se o nástupce technologie *Xamarin.Forms*, které je velmi podobný, avšak přináší mnoho inovací. Framework vznikl s hlavním cílem poskytovat prostředí, ve kterém je možná tvorba co nejvíce sdíleného kódu mezi platformy, co se týče jak výkonného, tak kódu pro uživatelské rozhraní. Kromě sdíleného kódu však lze opět psát i platformě specifický kód (Microsoft Corporation, nedatováno; Baptista a Abbruzzese, 2022, s. 496-497). Business logika se opět tvoří pomocí jazyka *C#* a uživatelské rozhraní pomocí jazyka *XAML*, který se opět mírně liší od ostatních technologií. Propojení dat, logiky a uživatelského rozhraní je primárně realizováno architekturou *MVVM* (Stonis, 2022, s. 9).



Obrázek 10 – Koncept .NET MAUI (Microsoft Corporation, nedatováno)

Co se týče inovací *.NET MAUI* vůči *Xamarin.Forms*, bylo přepracováno mnoho komponent, aby byla zajištěna co největší rozšiřitelnost a výkon. Také byla přidána podpora desktopových systémů *macOS*. Byla zjednodušena struktura projektu, kde oproti *Xamarin.Forms* s projektem pro každou platformu je využit pouze jeden projekt podporující všechny platformy. Ladění, sdílení zdrojů a tvorba nastavení je tak mnohem snazší. Obdobně byly sloučeny knihovny cílící na jednotlivé platformy, k dispozici jsou tak například multiplatformní knihovny pro ovládání hardwaru a práci se soubory. Služba *hot reload* je obohacena o plnou podporu a dokáže tak reagovat jak na změny uživatelského rozhraní, tak i na změny výkonného kódu. Také byla doplněna podpora hybridních aplikací prostřednictvím technologie *Blazor*.

Framework *.NET MAUI* obsahuje podobné portfolio komponent adaptujících se dle platformy jako v případě *Xamarin.Forms*. Princip adaptivních komponent lze vidět na obrázku 11. Stejně tak nabízí různé layouty pro rozvržení prvků, možnosti napojení dat na komponenty a různé možnosti zobrazení aplikace (Microsoft Corporation, nedatováno).



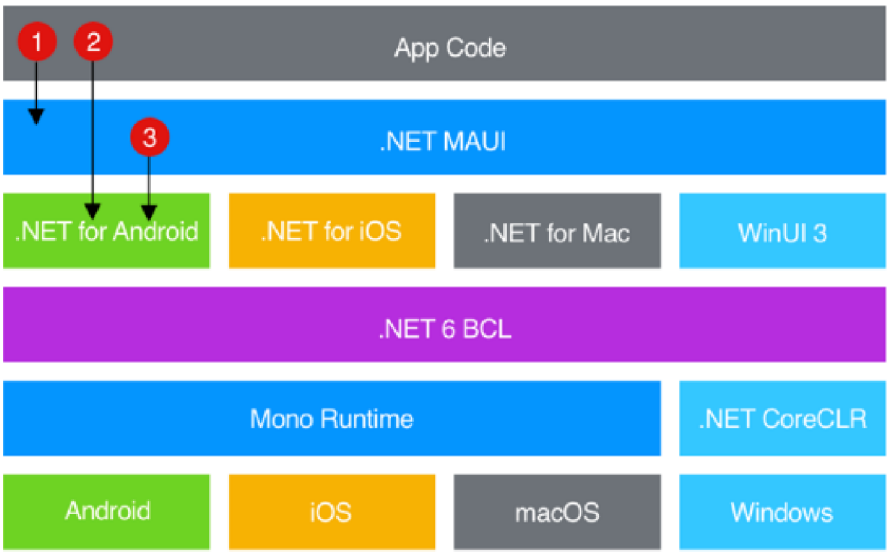
Obrázek 11 - Koncept tvorby komponent v .NET MAUI (Microsoft Corporation, nedatováno)

3.6.1 Architektura

Co se týče architektury na obrázku 12, z pohledu vyvíjeného kódu, *.NET MAUI* tvoří vrstvu zastřešující *.NET* frameworky pro jednotlivé platformy, ke kterým samo přistupuje. Mezi tyto frameworky pro platformy patří *.NET for Android*, *.NET for iOS*, *.NET for macOS* a *Windows UI 3* (*WinUI 3*). Vývojář může k jednotlivým platformním frameworkům přistupovat buď prostřednictvím vrstvy *.NET MAUI* nebo napřímo. Tyto přístupy jsou znázorněny na obrázku pod čísly jedna až tři. Platformní frameworky dále přistupují

k jednotné sadě knihoven *BCL*, abstrahující platformy od kódu a umožňující sdílení kódu. Na *BCL* navazují běhová prostředí *CLR*, kdy pro systémy *Android*, *iOS* a *macOS* je využito *Mono* a pro *Windows* *.NET CoreCLR*.

Finální tvorba nativních aplikací je pro každou platformu realizována specifickým způsobem. V případě systému *Android* dochází k běžnému postupu, kdy je kód jazyk přeložen do mezikódu *CIL*, jež je dále za běhu překládán do strojového kódu. Stejná metoda je využita i u systému *Windows* ovšem se zastřešením technologií *WinUI 3*. Jiná situace nastává u operačních systémů společnosti *Apple*, kde v případě *iOS* je kód jazyka *C#* rovnou přeložen do *ARM* strojového kódu. Na systém *macOS* se dále aplikace převádějí pomocí technologie *Mac Catalyst* umožňující běh *iOS* aplikací na *macOS* (Microsoft Corporation, nedatováno; Price, 2021, s. 545).

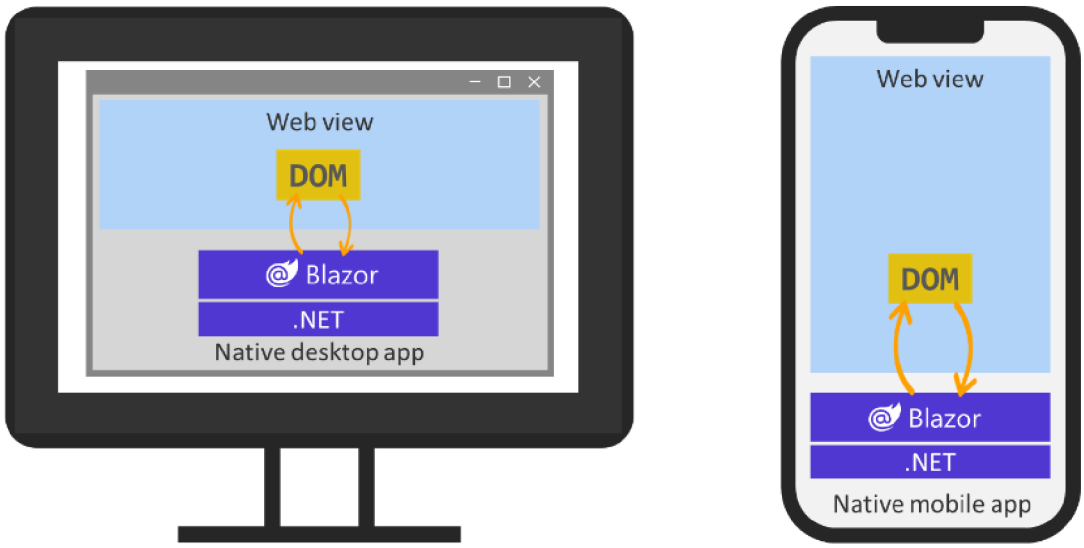


Obrázek 12 - Vysokoúrovňová architektura .NET MAUI (Baptista a Abbruzzese, 2022, s. 496)

3.6.2 Blazor Hybrid

Technologie *Blazor Hybrid* rozšiřuje framework *.NET MAUI* právě o hybridní přístup vývoje aplikací. Toto rozšíření je realizováno pomocí komponenty *WebView* v tomto případě zvanou *BlazorWebView*, která umožňuje používat framework *Blazor*, jeho znovupoužitelné komponenty a klientské webové technologie i v *.NET MAUI*. Kód frameworku *Blazor* se v tomto případě vykonává nativně prostřednictvím *.NET runtime* se simulací prostředí webového prohlížeče. Není tak využito překládání do standardu *WebAssembly* a *WebAssembly runtime*. Tento přístup je ukázán na obrázku 13. To vede u technologie *Blazor* k většímu výkonu a možnosti přístupu k nativním funkcím zařízení,

ke kterým u klasického použití ve webové aplikaci nemá. Dokonce lze nadále v rámci aplikace používat jak webové uživatelské rozhraní, tak i klasické nativní uživatelské rozhraní. Vytvořený projekt pomocí frameworku *Blazor* a webových technologií lze pak také samostatně nasadit jako webovou aplikaci. Při využití *.NET MAUI Blazor* tak získáme vyšší pokrytí, co se do počtu podporovaných platforem týče. Nevýhodou v tomto případě může být to, že oproti ostatním komponentám *.NET MAUI*, které se adaptují na danou platformu se vytvořený obsah neadaptuje a aplikace tak vypadá na všech platformách stejně. Kromě *.NET MAUI* lze technologii *Blazor Hybrid* využívat i pro *Windows Forms*, *WPF* a *UWP* (Baptista a Abbruzzese, 2022, s. 502-504).



Obrázek 13 - Koncept .NET MAUI Blazor (Microsoft Corporation, nedatováno)

3.7 Blazor

Jak již bylo v předchozích kapitolách zmiňováno, framework *Blazor* je nejnovější technologií platformy *.NET* pro tvorbu webových aplikací. Jedná se o open source framework pro jazyk *C#* umožňující tvorbu *SPA*. Jeho název poukazuje na využití vykreslovacího nástroje *Razor engine* na straně klienta. *Razor engine* spojuje webové technologie *HTML* a *CSS* s jazykem *C#* a umožňuje tvorbu znovupoužitelných komponent. *Blazor* je obdobou frameworků *Angular*, *React*, *Vue*, či *Svelte* pro jazyk *JavaScript*.

Blazor samotný nabízí dva modely tvorby aplikací. První model *Blazor WebAssembly* využívá webový standard *WebAssembly* a zpracovává veškerá data na klientské straně aplikace. Druhý model *Server-side Blazor* naopak vůbec nevyužívá standard *WebAssembly* a zpracovává všechna data na straně serveru, přičemž s klientskou částí komunikuje pomocí technologie *SignalR*. Oba tyto přístupy tak nabízejí velmi odlišné prostředí a možnosti, ačkoliv z pohledu vývoje se tak na první pohled nemusí zdát. Výhodou modelu *Blazor WebAssembly* je možnost běhu bez serverové části a efektivní využití výkonu zpracování dat na klientské části. Nevýhodou je však velikost aplikace a s tím se pojící i dlouhá doba načítání aplikace, neboť na klientské části spolu s aplikací musejí být i všechny potřebné technologie a knihovny *.NET* platformy. Výhodou druhého modelu je naopak skutečnost, že na klientskou část se posílají jen potřebná data, neboť se vše zpracovává na straně serveru. To však zase vede k neustálé potřebě mít připojení k serveru. Na server jsou v tomto případě také kladeny mnohem vyšší nároky, protože musí zpracovávat data pro všechny připojené instance aplikace. Za další, třetí možný model využití technologie *Blazor*, lze považovat právě technologii *Blazor Hybrid* (Himschoot, 2020, s. xvii-xxvi).

Téma frameworku *Blazor* je blíže popsáno v bakalářské práci autora (Homza, 2021, s. 43-46).

4 Vlastní práce

Jelikož je cílem této práce popis a zhodnocení možností multiplatformního vývoje frameworků *.NET MAUI* a *Blazor*, je v této kapitole popsána analýza a implementace testovací aplikace realizovaná zejména pomocí zmiňovaných frameworků. Ke zpracování vlastní práce jsou využity teoretické základy popsané v předchozí kapitole 3 a další poznatky, které autor získal během studia. Poznatky nalezené při zpracovávání vlastní práce, jsou následně využity v kapitole 5, při hodnocení využití a možností frameworků *.NET MAUI* a *Blazor*.

4.1 Analýza

První částí vlastní práce je analýza, ta spočívá v návrhu testovací aplikace, tedy v popisu aplikace a požadavků, popisu architektury, popisu a nákresů uživatelského rozhraní a popisu datového modelu. Jelikož vytvářená testovací aplikace slouží pouze pro využití v rámci této práce a žádná další strana není do vývoje zapojena, jsou požadavky na aplikaci určeny pouze autorem.

Pro získání dostatečných poznatků ohledně praktického využití frameworků *.NET MAUI* a *Blazor* a jejich následné zhodnocení, je vhodné vytvořit aplikaci o rozsahu obdobném produkčním aplikacím a splňující určité požadavky, podle kterých je dále použitelnost měřena. Není tak vhodná tvorba jednoduché aplikace jako je například kalkulačka. Nejprve je třeba nalézt základní znaky klasických produkčních aplikací, podle kterých jsou vytvořeny během analýzy požadavky na testovací aplikaci.

Mezi takové základní znaky včetně zřejmého zpracování určitých dat patří například využití uložišť pro zpracovávaná data. Uložiště dat mohou být přítomny lokálně na zařízení, kde je i používaná aplikace, či na vzdáleném serveru se kterým aplikace komunikuje, nebo může být dokonce využita kombinace obou uložišť se synchronizací, aby bylo možné aplikaci používat i bez přístupu k serveru a zároveň byly na serveru ukládány co nejvíce aktuální data. Z toho vyplívají další znaky produkčních aplikací, tedy komunikace se vzdáleným serverem a zjišťování stavu připojení zařízení a aplikace k internetu a serveru.

Protože frameworky *.NET MAUI* a *Blazor* umožňující tvorbu aplikací pomocí multiplatformního i hybridního přístupu je třeba klást důraz na další běžné znaky produkčních aplikací, tedy podporu více platforem a komunikaci s jejich nativními moduly, neboť ta je nejlépe optimalizována pro nativní vývojové technologie a u ostatních přístupů bývá často problematická. Ohledně využití nativních modulů je třeba prozkoumat, zda je ke všem potřebným modulům přístup, jak je tento přístup dostatečný a jeho využití náročné a zda je tento přístup jednotný pro všechny platformy, nebo se musí využívat specificky pro každou platformu zvlášť. Mezi tyto nativní moduly patří například hardware zařízení (fotoaparát, geolokace, …), systémové aplikace operačního systému (souborový systém, kalendář, kontakty, …), komponenty uživatelského rozhraní (tlačítka, upozornění, ...), push notifikace a jazyková lokalizace.

4.1.1 Popis aplikace a požadavků

Vyvíjenou aplikací je čtečka elektronických knih s možnostmi správy a čtení lokálně uložených elektronických knih ve formátu *EPUB*. Aplikace je vyvinuta pro dva operační systémy, tedy *Microsoft Windows* a *Android*.

Základním případem použití aplikace je situace, kdy má uživatel na svém zařízení lokálně uložené elektronické knihy ve formátu *EPUB* a tyto knihy nahraje do aplikace. Seznam nahraných knih je možné vidět v aplikaci na dvou stránkách. První stránkou s nahranými knihami je stránka *Domov*, kde se zobrazuje pět naposledy čtených knih seřazených podle posledního data čtení. U knih je zobrazován název a obrázek obálky knihy. Druhou stránkou s nahranými knihami je stránka *Knihovna*, kde se zobrazují všechny knihy, včetně názvu, obrázku obálky, jmen autorů, popisu a procentuálního počtu přečtených stránek. Stránky v knihovně je možné řadit podle názvu, procentuálního počtu přečtených stránek a posledního data čtení. Na obou těchto stránkách se nachází možnosti pro nahrávání nových knih do seznamu knih, mazání knih ze seznamu knih a otevírání nahraných knih ze seznamu knih k následnému čtení. Jako další možnost pro výběr knihy ke čtení aplikace využívá takzvané akce aplikace. Jedná se o seznam akcí, který se například v prostředí *Microsoft Windows* zobrazí při zmačknutí pravého tlačítka myši nad danou aplikací. Seznam akcí obsahuje stejně jako stránka domov pět naposledy čtených knih vyjádřených názvem, přičemž při stisknutí tlačítka s názvem dané knihy se otevře aplikace se stránkou čtečky a s otevřenou danou knihou.

Další stránkou aplikace je samotná čtečka, umožňující zobrazení pro čtení právě otevřené knihy a listování touto knihou. Nutnou funkcionalitou čtečky je hlídání postupu čtení knihy, aby uživatel při zavření knihy, či celé aplikace mohl při dalším spuštění aplikace a otevření knihy pokračovat ve čtení na stránce, kterou měl otevřenou jako poslední. Jednou z dalších funkcionalit je možnost zobrazení informací ze slovníku získaných ze vzdáleného serveru popisujících právě označené slovo na stránce knihy. Doplňkovými funkcionalitami čtečky je možnost tvorby záložek a nastavení vzhledu čtečky (velikost písma a barvy stránek a písma).

Poslední stránkou aplikace je stránka *Nastavení* s možnostmi nastavení notifikace pro připomenutí čtení v nastaveném čase, změnu jazyka na češtinu, či angličtinu a export a import dat.

4.1.2 Požadavky na implementaci

Po popisu aplikace a jejich požadavků je potřeba definovat požadavky na implementaci aplikace, respektive na využité technologie a vlastnosti aplikace. Primárním požadavkem je tedy tvorba programu v podobě multiplatformní aplikace pomocí frameworku *.NET MAUI* s využitím hybridního přístupu prostřednictvím frameworku *Blazor* a dalších webových technologií. Mezi další požadavky na využité technologie patří využití multiplatformní relační databáze *SQLite* a lokálního uložiště aplikace pro ukládání dat uživatele a využití komunikace s *REST API* na vzdáleném serveru, prostřednictvím technologií *HTTP* a *JSON*, pro získávání slovníkových dat.

Co se týče vlastností aplikace, má využívat především nativní uživatelské prostředí adaptující se dle dané platformy a asynchronně zpracovávat dlouho trvající operace, včetně komunikace s lokální databází a komunikace se vzdáleným serverem, aby nedocházelo k blokaci uživatelského prostředí aplikace. Aplikace má také reagovat na chyby zobrazením hlášení o chybě a možností pro další pokračování práce s aplikací.

4.1.3 Možnosti rozšíření

Produkčních aplikací realizujících funkci čtečky elektronických knih existuje velké množství. Aby se navržená čtečka elektronických knih plně vyrovnala ostatním produkčním čtečkám bylo by možné ji doplnit o další funkcionality jako je podpora více formátů knih, podpora aplikace pro více platforem, možnost předčítání knih, možnost nahrávání, čtení a stahování knih z internetu a další.

4.1.4 Zdůvodnění výběru aplikace

Popsaná aplikace splňuje několik znaků produkčních aplikací stanovených v předchozích subkapitolách a je možná její tvorba pomocí frameworků *.NET MAUI* a *Blazor*. Aplikace je tedy multiplatformní, využívá lokální uložiště dat, komunikaci s *REST API* na vzdáleném serveru, nativní moduly pro uživatelské rozhraní, přístup k souborovému systému, jazykovou lokalizaci a další prvky.

I přestože *.NET MAUI* podporuje operační systémy *Microsoft Windows*, *Android*, *macOS*, *iOS* a *Tizen*, byly vybrány pouze systémy *Microsoft Windows* a *Android* jako cílené platformy, protože pro otestování většiny funkcionalit stačí, neboť bude možné vyzkoušet aplikaci jak na osobním počítači, tak na mobilním zařízení. Také pro tvorbu instalačního souboru vyžadují tyto operační systémy pouze počítač se systémem *Microsoft Windows* a s nainstalovaným programem *Visual Studio*.

Formát *EPUB* byl pro čtečku vybrán, z důvodu jeho struktury založené na webových technologií, čímž dává v rámci aplikace dostatečný prostor pro práci s webovým frameworkem *Blazor* a jeho spolupráci jak s ostatními webovými technologiemi, tak s frameworkem *.NET MAUI*. Většina aplikace tak je vytvořena pouze pomocí frameworku *.NET MAUI*, tedy je využita kombinace jazyků *C#* a *XAML* a pouze pro stránku se samotnou čtečkou formátu *EPUB* je využit i framework *Blazor* s dalšími webovými technologiemi zapouzdřený v komponentě *WebView*. Takový způsob byl vybrán, aby se dostatečně prověřili možnosti spolupráce a kombinace frameworků *.NET MAUI* a *Blazor* a zároveň bylo možné vyzkoušet nativní komponenty uživatelského rozhraní. Jinak lze model *.NET MAUI Blazor Hybrid* samozřejmě využít pro celou aplikaci bez využití nativních komponent uživatelského rozhraní. Dokonce by tak mohla být aplikace vytvořena mimo nativní funkcionality i pro webové prohlížeče.

4.1.5 Architektura

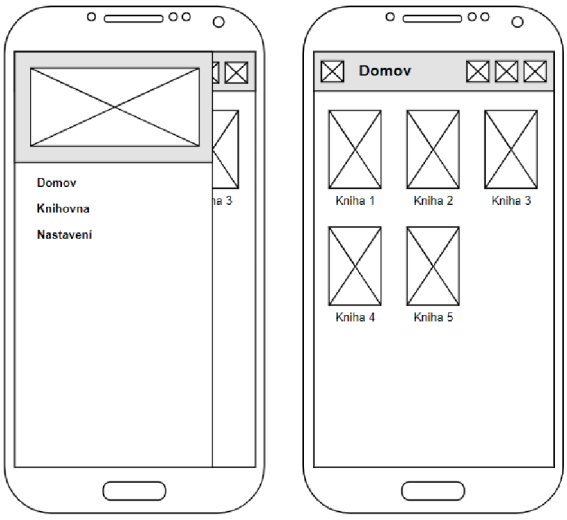
Následující částí analýzy je popis architektury testovací aplikace. Vytvářenou aplikací je hybridní aplikace vyvinutá pomocí technologie *.NET MAUI Blazor Hybrid*. Tedy jedná se o aplikaci využívající komponentu *WebView* a webové technologie, ale zároveň i multiplatformní technologie. Pracovní data aplikace jsou ukládána do lokální multiplatformní databáze *SQLite*. Aplikace také využívá komunikaci s *REST API* na vzdáleném serveru prostřednictvím protokolu *HTTP* pro získávání slovníkových dat o slovech. Data jsou s *REST API* předávána ve formátu *JSON*. Celá aplikace je tedy

nainstalována na klientském zařízení. Podporovanými platformami pro tvorbu instalačních balíčků jsou operační systémy *Microsoft Windows* a *Android*.

4.1.6 Uživatelské rozhraní

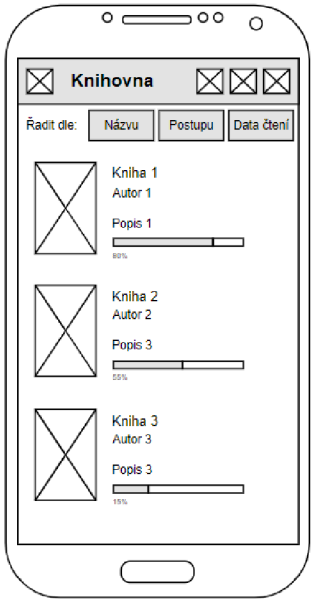
Další součástí analýzy je definice uživatelského rozhraní. Uživatelské rozhraní aplikace je na všech stránkách kromě stránky se čtečkou tvořeno nativními komponentami uživatelského rozhraní, tedy na každé platformě tyto komponenty vypadají rozdílně. Uživatelské rozhraní stránky se čtečkou je vytvořeno pomocí webových technologií a vypadá na všech platformách stejně. Pro popis základního průchodu aplikací a jednotnou definici uživatelské rozhraní jsou využity drátové modely (*wireframes*) se zobrazením pro telefon, ty slouží pro prvotní návrhy uživatelského rozhraní a jejich účelem je zejména definice rozmístění prvků na stránkách.

Základní stránkou je stránka *Domov*. Na této stránce, stejně jako na všech ostatních lze otevřít postranní menu s navigací kliknutím na tlačítko v levém horním rohu aplikace vedle titulku stránky a přepínat mezi stránkami *Domov*, *Knihovna* a *Nastavení*. Na stránce *Domov* je zobrazeno ve třech sloupcích maximálně pět naposledy čtených knih, včetně jejich názvu a obrázku obálky. Dvojitým kliknutím na knihu dojde k navigaci na stránku se čtečkou s otevřenou danou knihou. V pravém horním rohu se nacházejí tlačítka spouštějící akci po kliknutí pro otevření knihy, přidání knihy a odstranění knihy. Tlačítko pro otevření knihy spouští navigaci na stránku čtečky s otevřenou knihou. Tlačítko pro přidání knihy zobrazuje dialog pro výběr knihy a případné spouští akci pro přidání knihy. Poslední tlačítko, pro odstranění knihy, zobrazuje dialog pro ujištění, zda chce uživatel skutečně akci provést a případně spouští akci pro odstranění knihy. Tlačítka pro otevření knihy a odstranění knihy jsou aktivní po výběru některé knihy jednoduchým kliknutím. Vytvořený drátěný model je zobrazený na obrázku 14.



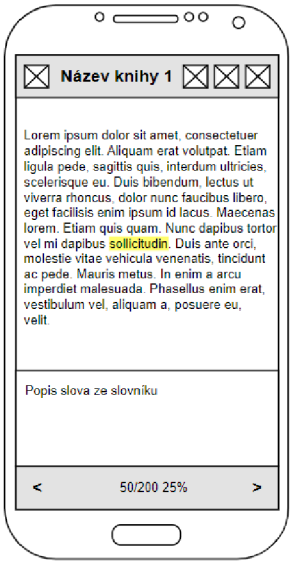
Obrázek 14 - Drátěný model stránky domov a menu (Vlastní zpracování)

Stránka *Knihovna*, jejíž drátěný model je zobrazený na obrázku 15, je podobná stránce *Domov*. Jedná se o seznam všech nahraných knih zobrazených v seznamu, včetně názvu, obrázku obálky, jmen autorů, popisku a znázornění postupu čtení pomocí zobrazeného indikátoru a vypsané procentuální hodnoty. Tato stránka nabízí stejné akce pro práci s knihami jako stránka *Domov*. Navíc lze seznam knih řadit pomocí tlačítek dle názvu, postupu čtení a posledního data čtení.



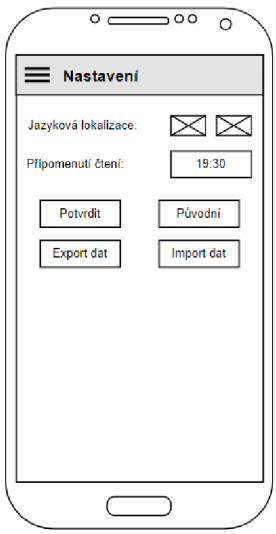
Obrázek 15 - Drátěný model stránky Knihovna (Vlastní zpracování)

Stránka zobrazující aktuálně otevřenou knihu, s drátěným modelem na obrázku 16, obsahuje několik prvků. Většinu stránky tvoří čtečka s otevřenou knihou, pod ní je při výběru slova zobrazován blok se slovníkovými informacemi o právě vybraném slově a dole jsou zobrazeny ovládací prvky čtečky s informacemi o postupu čtení. Horní část obsahuje tlačítko pro návrat na předchozí stránku, název knihy a tlačítka pro zobrazení nastavení, záložek a přidání záložky.



Obrázek 16 - Drátěný model stránky se čtečkou a slovníkem (Vlastní zpracování)

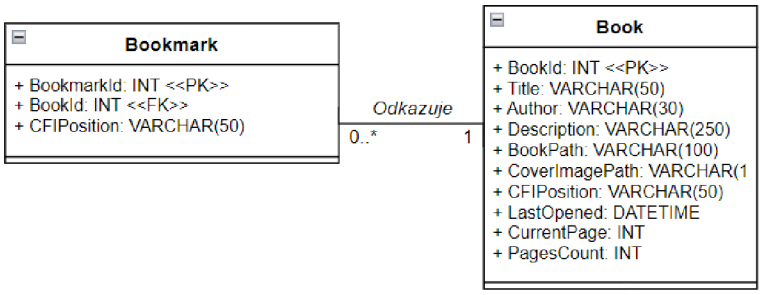
Poslední stránkou je *Nastavení* obsahující editační komponenty pro tvorbu notifikace k připomenutí čtení a změnu jazyka a tlačítka na potvrzení nastavení, návrat původního nastavení, export dat a import dat. Drátěný model stránky *Nastavení* je ukázán na obrázku 17.



Obrázek 17 - Drátěný model stránky Nastavení (Vlastní zpracování)

4.1.7 Datový model

Poslední část analýzy je věnována popisu datového modelu pomocí třídního modelu jazyka UML, kterou využívá testovací aplikace. Co se týče ukládaných dat do databáze a jejich relačního datového modelu, zobrazeného na obrázku 18, jedná se o velmi jednoduchý model o dvou tabulkách. První tabulka je nazvána Book a obsahuje základní informace o knize a informace o průběhu čtení uživatelem. V rámci dalšího rozšiřování aplikace by bylo vhodné tyto dva druhy informací rozdělit do dvou tabulek. Druhá tabulka je nazvána Bookmark, popisující založenou stránku v určité knize.



Obrázek 18 - Datový model testovací aplikace (Vlastní zpracování)

4.2 Implementace

Po provedení analýzy testovací aplikace je možné zpracovat druhou část vlastní práce, tedy samotnou implementaci navržené testovací aplikace. Kapitola implementace se tak věnuje vývoji celé aplikace od přípravy projektu a instalace dalších nástrojů a knihoven, přes tvorbu jednotlivých funkcionalit až po možnosti nasazení aplikace. Z důvodu většího rozsahu aplikace jsou některé techniky pouze naznačeny.

4.2.1 Příprava projektu

K vývoji celé aplikace byl využit počítač s operačním systémem *Microsoft Windows* a s nainstalovaným komplexním vývojovým prostředím *Visual Studio 2022 Community* od společnosti *Microsoft*. To umožňuje jednoduchou instalaci a správu všech potřebných doplňků v rámci celého vývoje. Pro založení projektu s předpřipravenou strukturou, nastavením a nainstalovanými doplňky byla v rámci vývojového prostředí využita šablona *.NET MAUI Blazor App* s verzí frameworku *.NET 7*.

4.2.2 Použité nástroje a knihovny

Pro vývoj testovací aplikace bylo využito několika rozšiřujících knihoven, určených pro frameworky *.NET MAUI* a *Blazor*. Tato rozšíření lze instalovat v programu *Visual Studio* pomocí správce balíčku *Nuget*. Jelikož vyvíjená aplikace podporuje hybridní prostředí s frameworkem *Blazor* a webovými technologiemi, lze využívat v rámci tohoto prostředí i různé rozšíření pro webové technologie. Toho bylo využito u stránky čtečky, kde je použita knihovna pro jazyk *JavaScript*. Mezi použité dodatečně nainstalované knihovny patří:

- **CommunityToolkit.Maui** – kolekce rozšiřujících komponent a funkcionalit pro framework *.NET MAUI*. V rámci aplikace je rozšíření využito především pro zobrazování notifikační komponenty *Toast*.
- **CommunityToolkit.Mvvm** – knihovna pro framework *.NET* pro zjednodušení práce v rámci modelu *MVVM*.
- **Microsoft.Extension.Localization** – rozšíření pro framework *.NET* umožňující práci s jazykovou lokalizací.

- **Plugin.LocalNotification** – knihovna pro tvorbu lokálních notifikací v rámci frameworku *.NET MAUI*. Bez využití této knihovny je potřeba implementovat lokální notifikace pro každou platformu zvlášť. Nevýhodou této knihovny je v současné době podpora pouze systému *Android* a *iOS*.
- **Sqlite-net-pcl** – knihovna pro práci s databází *SQLite* v rámci frameworku *.NET*.
- **SQLitePCLRaw.provider.dynamic_cdecl** – knihovna doplňující knihovnu *Sqlite-net-pcl* o podporu některých operačních systémů.
- **VersOne.Epub** – knihovna pro platformu *.NET* obsahující podporu pro čtení souborů typu *EPUB*.
- **EPUB.js** – knihovna pro jazyk *JavaScript* umožňující v rámci webové stránky zobrazení a procházení souboru typu *EPUB*.

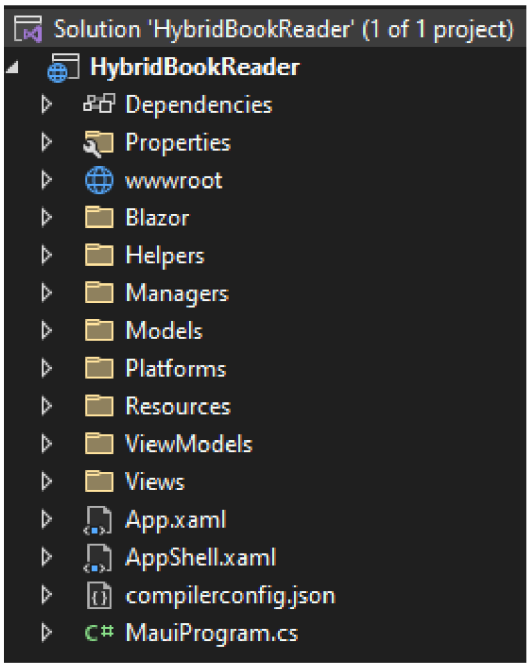
4.2.3 Struktura projektu

Vygenerovaná struktura projektu byla pro lepší přehlednost upravena a doplněna o některé další adresáře. Její finální podobu lze vidět na obrázku 19. Pro zjednodušení popisu struktury si rozdělíme adresáře a soubory struktury projektu na tři části, tedy na část obsahující základní nastavení aplikace, část obsahující modely, logiku a uživatelské rozhraní a část obsahující komponenty frameworku *Blazor* a webové technologie.

Část týkající se nastavení aplikace obsahuje zjednodušeně soubory MauiProgram pro konfiguraci aplikace, registraci závislosti a spuštění aplikace, App.xaml, pro definici zdrojů, App.xaml.cs, pro reakce na události životního cyklu aplikace a soubory AppShell.xaml a AppShell.xaml.cs, k definice možností navigace aplikace. Dále do této části patří adresář Platforms, obsahující specifický kód pro platformy a adresář Resources, obsahující zdroje jako jsou obrázky, fonty, lokalizace, či styly.

Do druhé části patří adresáře Helpers a Managers, obsahující sdílené pomocné třídy pro práci se specifickou částí logiky. Do této části také patří adresáře Models, ViewModels a Views obsahující třídy a stránky rozdělené podle modelu *MVVM*.

Součástí poslední části je adresář Blazor, obsahující komponenty vytvořené pomocí frameworku *Blazor* a adresář wwwroot, obsahující *HTML* stránky, *CSS* styly a *JavaScriptové* skripty a knihovny.



Obrázek 19 - Struktura projektu testovací aplikace (Vlastní zpracování)

4.2.4 Architektura projektu

Architektura projektu je postavená na modelu *MVVM*, spočívajícím v oddělení modelu dat, logické vrstvy a uživatelského rozhraní, čímž dochází k přehlednějšímu dělení kódu pro čitelnost a snazší tvorbu změn. V prostředí frameworku *.NET MAUI* samozřejmě lze tvořit aplikace i bez využití modelu, nebo pomocí jiných modelů.

Prvním prvkem modelu *MVVM* je tedy model, jenž popisuje data aplikace v jazyce *C#* a je oddělený od logiky. Dalším prvkem je *ViewModel*, taktéž vytvořený v jazyce *C#*, uchovávající stav aplikace pomocí speciálních datových struktur vyvolávající události při změně. Na tyto změny právě reaguje *View*, tedy stránka, aplikace, či dialog vytvořený v jazyce *XAML*, obsahující prvky uživatelského rozhraní, které podle stavu *ViewModelu* překresluje. *ViewModel* tedy slouží také jako spojovací prvek *Modelu* a *View*. Prvky vyvolávající události, které obsahuje *ViewModel* jsou kolekce `ObservableCollection<T>` a rozhraní `INotifyPropertyChanged`. *View* odkazuje na prvky *ViewModelu* pomocí příkazů `Binding` a `Command`. Mezi modely aplikace patří:

- **Book** – třída uchovávající data o knížce.
- **Bookmark** – třída obsahující data o záložce.
- **TransferModel** – třída sloužící pro export a import všech dat aplikace.
- **WordDictionary** – třída uchovávající přijatá slovníková data.

ViewModely a *View* jsou popsány společně, neboť v našem případě vždy *View* a *ViewModel* patří jedné stránce:

- **HomeView a HomeViewModel** – stránka *Domov* se seznamem naposledy čtených knih.
- **LibraryView a LibraryViewModel** – stránka *Knihovna* se seznamem všech knih.
- **ReaderView a ReaderViewModel** – stránka se čtečkou knih ve formátu EPUB.
- **SettingsView a SettingsViewModel** – stránka *Nastavení* pro změnu uživatelského nastavení.

Dále byli vytvořeni takzvaní manažeři a pomocné třídy. Tím jsou myšleny třídy obsahující nějakou specifickou část logiky, jež je využívána napříč aplikací. Jako manažeři byly označeny třídy plnící nějakou úrovňově důležitou činnost a případně uchovávají stav. V našem případě byli všichni vytvořeni za využití návrhového vzoru *Singleton*. Patří mezi ně:

- **DataManager** – manažer pro správu dat, komunikaci s databází, komunikaci s *REST API*, export dat a import dat.
- **LocalizationManager** – manažer pro správu jazykové lokalizace aplikace, nastavuje jazykovou lokalizaci a poskytuje dle nastavené jazykové lokalizace texty aplikace.
- **SettingsManager** – manažer pro správu uživatelského nastavení.
- **StateManager** – hlavní manažer poskytující ucelené funkce a spravující stav aplikace.

Pomocné třídy byly v našem případě vytvořeny jako úrovňově méně důležité a statické třídy, které neuchovávají stav a poskytují pomocné funkce. Mezi pomocné třídy patří:

- **UIHelper** – pomocná třída pro práci s uživatelským rozhraním, jedná se především o práci s dialogy a notifikacemi.
- **FileHelper** – pomocná třída pro práci se soubory a souborovým uložištěm.

4.2.5 Využití modelu MVVM

Po předchozím představení projektu a jeho vlastností je dále popsána implementace jednotlivých částí aplikace s ukázkami zpracování. Jako první je ukázána obecná tvorba stránky *Domov* pomocí architektury *MVVM* s daným *View*, *ViewModelem* a *Modelem*. Jako model pro tuto stránku je ukázaná třída Book ve zdrojovém kódu 1. Jedná se tedy o prostou třídu s vlastnostmi, která neplní žádnou logiku aplikace.

```
public class Book
{
   public int BookId { get; set; }
   public string Title { get; set; }
   public string Author { get; set; }
   public string Description { get; set; }
   public string BookPath { get; set; }
   ...
}
```

Zdrojový kód 1

Část třídy HomeViewModel je zobrazena v bloku zdrojový kód 2. Tato třída obsahuje logiku stránky *Domov*, poskytuje tedy seznam knih, právě vybranou knihu, funkce pro přidání, odstranění a čtení knihy a další. Ukázka obsahuje odkaz na objekt pro lokalizaci LocationManager, seznam knih library, funkci pro přidání knihy AddBook a funkci pro načtení knih SetLibrary. Jak je vidět ve zdrojovém kódu, třída odkazuje na seznam knih pomocí kolekce ObservableCollection<Book>, ale neimplementuje rozhraní INotifyPropertyChanged. Namísto implementace tohoto rozhraní, je třída HomeViewModel potomkem třídy ObservableObject z knihovny CommunityToolkit.Mvvm. Toto rozšíření pro proměnné začínající malým písmenem označené notací ObservableCollection, automaticky generuje stejně pojmenovaný parametr začínající velkým písmenem a využívající rozhraní INotifyPropertyChanged. Obdobně v případě funkcí označených notací RelayCommand rozšíření generuje stejně pojmenovanou funkci doplněnou o text Command na konci, využívající rozhraní INotifyPropertyChanged. Využitím tohoto rozšíření tak dochází ke značné úspoře zápisu.

```
public partial class HomeViewModel : ObservableObject
{
   public LocalizationManager LocalizationManager
      => LocalizationManager.Instance;

   [ObservableProperty] ObservableCollection<Book> library;
   public Book SelectedBook { get; set; }

   [RelayCommand]
   private async Task AddBook()
   {
      await App.StateManager.AddBook();
      SetLibrary();
      SelectedBook = null;
   }

   public void SetLibrary()
   {
      Library = new ObservableCollection<Book>(App.StateManager.Library
            .OrderByDescending(x => x.LastOpenedAsDate()).Take(5));
   }
   ...
}
```

Zdrojový kód 2

Jako poslední zbývá popsat *View* dané stránky, jehož zjednodušený kód lze vidět ve zdrojovém kódu 3. Pro tvorbu stránek a uživatelského rozhraní nabízí jazyk *XAML* mnoho možností a komponent. Na ukázce je kromě využití jazyka *XAML* pro tvorbu stránky ContentPage, panelu nástrojů ToolbarItems a seznamu knih CollectionView zobrazeno využití příkazů Binding a Command pro připojení na funkce a vlastnosti třídy HomeViewModel.

```
<ContentPage x:DataType="viewmodel:HomeViewModel"
         Title="{Binding LocalizationManager[Home]}">
   <ContentPage.ToolbarItems>
      <ToolbarItem Command="{Binding ReadBookCommand}"
               Text="{Binding LocalizationManager[Read]}"/>
   </ContentPage.ToolbarItems>
    <CollectionView ItemsSource="{Binding Library}">
      <CollectionView.ItemTemplate>
         <DataTemplate x:DataType="model:Book">
            <VerticalStackLayout>
               <VerticalStackLayout>
                  <Image Source="{Binding CoverImagePath}"/>
                  <Label Text="{Binding Title}" />
               </VerticalStackLayout>
            </VerticalStackLayout>
         </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
         . . .
</ContentPage>
```

Zdrojový kód 3

4.2.6 Navigace

Další část se věnuje způsobům tvorby navigace v rámci aplikace v prostředí *.NET MAUI*. Pro tvorbu hlavní navigace lze využívat postranní nabídku, záložky, nebo jejich kombinaci. Hlavní navigace aplikace, zobrazená v ukázce zdrojového kódu 4, je uložena v souboru AppShell.xaml a je vytvořená jako postranní nabídka pomocí komponent Shell a FlyoutItem, definující jednotlivé stránky názvem Title a obsahem ContentTemplate.

```
<Shell xmlns:view="clr-namespace:HybridBookReader.Views">
  <Shell.FlyoutHeader>
    <Grid HeightRequest="100" BackgroundColor="#037cd3">
      <Image Source="read.png"/>
    </Grid>
  </Shell.FlyoutHeader>
  <FlyoutItem Title="{Binding LocalizationManager[Home]}">
    <ShellContent ContentTemplate="{DataTemplate view:HomeView}" />
  </FlyoutItem>
  <FlyoutItem Title="{Binding LocalizationManager[Library]}">
    <ShellContent ContentTemplate="{DataTemplate view:LibraryView}"/>
  </FlyoutItem>
  <FlyoutItem Title="{Binding LocalizationManager[Settings]}">
    <ShellContent ContentTemplate="{DataTemplate view:SettingsView}"/>
  </FlyoutItem>
</Shell>
```

Zdrojový kód 4

Kromě toho je možné používat hierarchickou navigaci mezi stránkami, kdy je při přesměrování na stránku možné se vrátit pomocí tlačítka zpět na předchozí stránku. Tuto navigaci lze realizovat zavoláním metody GoToAsync() objektu Shell.Current, jak je ukázáno ve zdrojovém kódu 5.

```
await Shell.Current.GoToAsync(nameof(ReaderView));
```

Zdrojový kód 5

Aby tuto navigaci bylo možné vytvořit, musí se stránka, na kterou se při navigaci přechází registrovat funkcí Routing.RegisterRoute() v konstruktoru třídy AppShell v souboru AppShell.xaml.cs, jako v ukázce zdrojového kódu 6.

```
public AppShell()
{
  InitializeComponent();
  Routing.RegisterRoute(nameof(ReaderView), typeof(ReaderView));
}
```

Zdrojový kód 6

4.2.7 Nativní prvky

Následující část se zabývá implementací nativních prvků a funkcí, mezi které patří jazyková lokalizace, notifikace, tvorba nativních komponent a další. Pro velké množství nativních funkcionalit má *.NET MAUI*, či jiné rozšiřující knihovny připravené multiplatformní funkce, avšak je možné narazit i na chybějící funkcionality, které je potřeba zpracovat pomocí specifického kódu pro dané platformy. Při využívání nativních funkcionalit operačního systému, také některé systémy z důvodu bezpečnosti vyžadují registraci požadavku na využívání dané funkcionality. Takovým případem je například systém *Android*, u kterého je potřeba tyto požadavky zapsat do atributů tagů uses-permission do souboru AndroidManifest.xml, jako v ukázce kódu 7.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
        ...
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
</manifest>
```

Zdrojový kód 7

Testovací aplikace podporuje nastavení jazykové lokalizace na češtinu, či angličtinu. Lokalizované texty jsou zapsány v aplikaci do souborů Localization.cs.resx a Localization.en.resx, pomocí zápisu *klíč – hodnota*. K poskytování funkcí lokalizace v rámci aplikace slouží třída LocalizationManager, zobrazená v ukázce zdrojového kódu 8. Jedná se o třídu využívající návrhový vzor *Singleton* a implementující rozhraní INotifyPropertyChanged. Při zavolání objektu Instance v podobě Instance[„text“], kde text je *klíč* lokalizovaného řetězce, vrací hodnotu lokalizovaného řetězce. Významná je také metoda pro nastavení jazyka SetCulture().

```
public class LocalizationManager : INotifyPropertyChanged
{
   public static LocalizationManager Instance { get; }
      = new LocalizationManager();
   public event PropertyChangedEventHandler PropertyChanged;
   public string SavedLanguage { get; private set; }

   public object this[string resourceKey]
      => Localization.ResourceManager.GetObject(resourceKey, Localization.Culture) ??
Array.Empty<byte>();

   public void SetCulture(string formatName)
   {
      CultureInfo culture = new CultureInfo(formatName);
      SavedLanguage = formatName;
      Localization.Culture = culture;
      Thread.CurrentThread.CurrentCulture = culture;
      Thread.CurrentThread.CurrentUICulture = culture;
      CultureInfo.DefaultThreadCurrentCulture = culture;
      CultureInfo.DefaultThreadCurrentUICulture = culture;
   }
   ...
}
```

Zdrojový kód 8

V rámci aplikace je potřeba zobrazovat různé dialogy a notifikační komponenty. Aplikace využívá dialogy pro upozornění, potvrzení akce a výběry souborů. Pro upozornění je však zejména v aplikaci používaná notifikační komponenta Toast. Ve zdrojovém kódu 9 je funkce zobrazující dialog pro potvrzení záznamu s nadpisem, popisem a tlačítky, vracející rozhodnutí uživatele jako logickou hodnotu typu pravda, či nepravda. Tyto dialogy jsou součástí frameworku *.NET MAUI* a zobrazují se pomocí odkazu na aktuální objekt aplikace Application.Current.

```
public static async Task<bool> ShowQuestionDialog()
{
   return (await Application.Current.MainPage.DisplayAlert(
      "Upozornění",
      "Opravdu chcete smazat záznam?",
      "Ano",
      "Ne"));
}
```

Zdrojový kód 9

Ukázku tvorby a zobrazení notifikační komponenty Toast s popiskem o právě přidané knize obsahuje zdrojový kód 10. Toast nepatří mezi základní komponenty frameworku *.NET MAUI*, ale je součástí knihovny CommunityToolkit.Maui.

```
public static async Task ShowBookAddedToast(string bookTitle)
{
   await Toast.Make(
      $"Kniha {bookTitle} byla úšpěšně přidána").Show();
}
```

Zdrojový kód 10

Testovací aplikace pro připomenutí času vyhrazeného ke čtení nabízí možnost nastavení notifikace, která se zobrazí v určitou dobu. Lokální notifikace jsou v aplikaci využívány pomocí knihovny Plugin.LocalNotification. V ukázce zdrojového kódu 11, je zobrazena funkce SetNotification vytvářející notifikaci prostřednictvím objektu request s danými parametry třídy NotificationRequest. Tato knihovna byla využita, neboť framework *.NET MAUI* nenabízí zatím vlastní multiplatformní knihovnu pro práci s lokálními notifikacemi. Nevýhodou využité knihovny je v současné době podpora pouze pro systémy *Android* a *iOS*, přičemž zbylé systémy mají dostat podporu později. Možnost nastavení lokální notifikace pro připomenutí čtení tak nabízí pouze verze aplikace pro systém *Android*.

```
public void SetNotification(DateTime notif)
   NotificationRequest request = new NotificationRequest
   {
      NotificationId = 1154,
      Title = LocalizationManager.Instance["ReadingTime"].ToString(),
      BadgeNumber = 24,
      CategoryType = NotificationCategoryType.Reminder,
      Schedule = new NotificationRequestSchedule
      {
         NotifyTime = NotificationTime,
         RepeatType = NotificationRepeat.Daily
      },
   };
   LocalNotificationCenter.Current.Show(request);
}
```

Zdrojový kód 11

Právě z důvodu chybějící podpory pro operační systém *Windows* knihovny Plugin.LocalNotification pro tvorbu lokálních notifikací, bylo potřeba vytvořit kód specifický pro určité platformy. Možnost tvorby specifického kódu pro platformy v rámci jazyka *C#* pomocí bloku #if/#endif je zobrazena v ukázce zdrojového kódu 12. Volání funkce UpdateNotification() se tak v tomto případě provede pouze na zařízeních se systémy *Android* a *iOS*.

```
private async Task SaveSettings()
{
   if (SettingsChanged)
   {
      #if ANDROID || IOS
      UpdateNotification();
      #endif
      ...
   }
}
```

Zdrojový kód 12

Kód specifický pro určité platformy lze však psát i v jazyce *XAML*, k tomu slouží syntaktické rozšíření jazyka OnPlatform. Ve zdrojovém kódu 13 je ukázán popisek, jehož viditelnost je omezená pouze na systémy *Android* a *iOS* právě pomocí rozšíření OnPlatform.

```
<Label IsVisible="{OnPlatform False,Android=True, iOS=True}"
      FontSize="18"
      VerticalOptions="Center"
      Text="{Binding LocalizationManager[SetNotif]}"/>
```

Zdrojový kód 13

Součástí moderních aplikací jsou akce aplikace umožňující rychlé spuštění nějaké úlohy aplikace, bez jejího předchozího spouštění. Pro využití akcí aplikace je nejprve tuto funkcionalitu potřeba zaregistrovat pomocí funkce OnAppAction ve funkci CreateMauiApp třídy MauiProgram, jak je ukázáno ve zdrojovém kódu 14. Parametrem funkce OnAppAction je funkce, která reaguje na spuštění akce aplikace, v našem případě to je funkce HandleAppActions třídy App.

```
public static MauiApp CreateMauiApp()
{
   var builder = MauiApp.CreateBuilder();
   builder
      .UseMauiApp<App>()
      .UseMauiCommunityToolkit()
      .ConfigureEssentials(essentials =>
      {
         essentials
         .OnAppAction(App.HandleAppActions);
      });
   ...
}
```

Zdrojový kód 14

Aplikace v rámci akcí aplikace zobrazuje nanejvýš pět naposledy čtených knih, kde při spuštění akce aplikace nad libovolnou knihou aplikace přejde na stránku se čtečkou s otevřenou danou knihou. Tento seznam akcí je v aplikace aktualizován po přidání, odebrání, či čtení nějaké knihy. Příklad zdrojového kódu 15 obsahuje plnění seznamu akcí aplikace pěti naposledy čtenými knihami pomocí funkce SetAsync objektu AppActions.Current. Tato funkce přijímá seznam akcí, které se skládají z identifikátoru a popisku.

```
private async Task CreateAppActions()
{
   if (AppActions.Current.IsSupported)
   {
      List<AppAction> actions = new List<AppAction>();

      List<Book> actionsLibrary = Library
         .OrderByDescending(x => x.LastOpenedAsDate())
         .Take(5).ToList();

      foreach (Book book in actionsLibrary)
         actions.Add(new AppAction(book.BookId.ToString(),
            book.Title));
   }

   await AppActions.Current.SetAsync(actions);
}
```

Zdrojový kód 15

Po spuštění libovolné akce aplikace se tedy zavolá funkce HandleAppActions, zobrazená v příkladu zdrojového kódu 16. Jelikož se provede prvotní spuštění aplikace, je potřeba aplikaci inicializovat pomocí funkce Init, která naváže spojení s databází a načte seznam knih. Po inicializaci je možné přejít na stránku se čtečkou a zobrazit vybranou knihu pro čtení.

```
public static void HandleAppActions(AppAction appAction)
{
   Current.Dispatcher.Dispatch(async () =>
   {
      await StateManager.Init();
      await StateManager.OpenBook(int.Parse(appAction.Id));
   });
}
```

Zdrojový kód 16

4.2.8 Práce s daty

Další část implementace aplikace je věnována práci s daty ve smyslu správy dat a komunikaci s uložišti, databázemi a vzdálenými servery.

Nejsnadněji přístupnou možností pro lokální ukládání dat aplikace v rámci frameworku *.NET MAUI* je lokální uložiště typu *klíč – hodnota*, kde *klíč* je reprezentovaný textovým řetězcem. Jedná se tak o uložiště vhodné pro ukládání především dat jednoduchých datových typů jako jsou čísla, textové řetězce, datumy a tak dále. Způsob práce s tímto uložištěm je zobrazen v příkladu zdrojového kódu 17, zde je využita funkce `Set` pro zápis dat do uložiště a funkce `Get` pro získání dat z uložiště. K tomuto uložišti je možné přistupovat pomocí statické třídy `Preferences` s funkcemi pro dotazování, čtení, zápis a mazání dat.

```
Preferences.Set("NotificationDate", notificationDate);
string value = Preferences.Get("valueName", string.Empty);
```

Zdrojový kód 17

Komplexnější možnosti pro ukládání dat nabízí multiplatformní lokální databáze *SQLite*. Jedná se o odlehčenou relační databázi, která má celý obsah uložen v rámci jednoho souboru. Práce s databází je zajištěna pomocí knihovny `Sqlite-net-pcl` pro framework *.NET MAUI*. S databází lze pracovat stejně jako s ostatními běžnými relačními databázemi, pomocí dotazů napsaných v jazyce *SQL*. V ukázce kódu 18 je vytvořená funkce pro získávání všech záznamů z tabulky `Book` pomocí funkce `QueryAsync`. Použitá knihovna nabízí jak synchronní, tak asynchronní spojení s databází, které bylo využito, aby nedocházelo k blokaci reagování a vykreslování uživatelského rozhraní. Funkce `QueryAsync` je také obalena blokem `try-catch` z důvodu možného výskytu chyby spojení s databází.

```
public async Task<List<Book>> GetBooksFromDb()
{
    try
    {
        return await _conn.QueryAsync<Book>(@"SELECT * FROM Book;");
    }
    catch (Exception ex)
    {
        await UIHelper.ShowUnexpectedErrorDialog(ex.Message);
        return null;
    }
}
```

Zdrojový kód 18

Jak již bylo zmíněno, aplikace pro získávání slovníkových dat o vybraných slovech využívá volně přístupné *REST API* na vzdáleném serveru. Taková komunikace funguje

stejně jako například u webových aplikací, za využití protokolu *HTTP* a formátu *JSON*. Funkce získávající slovníková data je obsažena v příkladu zdrojového kódu 20. Ve funkci je nejprve v rámci první podmínky if zkontrolováno, zda má aplikace přístup k připojení k internetu. Jako další dochází ke komunikaci s *REST API* prostřednictvím protokolu *HTTP* a jeho metody *GET*. Na základě tohoto volání jsou obdržena data ze vzdáleného serveru ve formátu *JSON*. Pokud aplikace obdržela data se statusovým kódem značící úspěch, jsou data dále přečtena a převedena z formátu *JSON* do objektu třídy WordDictionary určeného pro tato data.

```
public async Task<WordDictionary> GetWordDefinition(string word)
{
   string error = string.Empty;
   WordDictionary wordDictionary = null;
   if (Connectivity.Current.NetworkAccess == NetworkAccess.Internet)
   {
      HttpResponseMessage response = await _httpClient.GetAsync(
         $"https://api.dictionaryapi.dev/api/v2/entries/en/{word}");
      if (response.IsSuccessStatusCode)
      {
         string cont = await response.Content.ReadAsStringAsync();
         List<WordDictionary> wordDictionaryList =
            JsonSerializer
               .Deserialize<List<WordDictionary>>(content);

         if (wordDictionaryList.Count > 0)
            wordDictionary = wordDictionaryList.FirstOrDefault();
         else
            wordDictionary = null;
      }
   }
   return wordDictionary;
}
```

Zdrojový kód 19

4.2.9 Využití hybridního přístupu

Následující významnou částí implementace testovací aplikace je využití hybridního přístupu, frameworku *Blazor* a dalších webových technologií. Framework *.NET MAUI* nabízí mnoho možnosti jak hybridní přístup a framework *Blazor* využít. Lze jej využít pro celou aplikaci, jednotlivé stránky, či komponenty a oba přístupy velmi volně kombinovat. V rámci testovací aplikace je hybridní přístup využit pouze pro stránku čtečky. Mezi využité technologie této stránky tak patří framework *Blazor* a jazyky *HTML*, *CSS* a *JavaScript*. Pro zobrazení a procházení knihy je využívána knihovna EPUB.js pro jazyk *JavaScript*. Důvodem využití hybridního přístupu a frameworku *Blazor* je právě zmiňovaná knihovna pro jazyk *JavaScript*, která nemá pro samotný framework *.NET MAUI* dostatečnou obdobu. Využití

hybridního přístupu tak je ukázáno na samotné čtečce a komunikaci *Blazoru* s jazykem *JavaScript*. V ukázce kódu 22 je zobrazen kód vytvořený v jazyce *HTML* obsahující *HTML tagy* pro vytvoření prostoru pro čtečku a ovládání.

```
<div id="viewer" class="spreads"></div>
<a id="prev" href="#prev" class="arrow">‹</a>
<a id="next" href="#next" class="arrow">›</a>
<div id="controls">
  <input id="current-percent" size="3" value="0" /> %
</div>
...
```

Zdrojový kód 20

V dalším kroku je potřeba využít vytvořený kód v jazyce *JavaScript*, který je uložený v samostatném souboru reader.js a používá knihovnu EPUB.js. Tato část je zobrazena v ukázce zdrojového kódu 23. Jelikož skript reader.js používáme v rámci frameworku *Blazor* pouze na jediné stránce, není tento soubor nahrán globálně pro celou hybridní část aplikace, ale samostatně pro danou stránku jako modul do proměnné _module. Jako další dochází k vytvoření reference objRef na danou *Blazor* stránku, která je potřebná, aby mohl skript jazyka *JavaScript* volat funkce vytvořené na stránce frameworku *Blazor*. Dále dochází k volání funkcí skriptu jazyka *JavaScript* z frameworku *Blazor*, kde je nejprve odeslána reference na danou stránku a poté odeslána samotná kniha jako pole bajtů.

```
protected override async Task OnAfterRenderAsync(bool firstRender)
{
  if (firstRender)
  {
    _module = await js.InvokeAsync<IJSObjectReference>("import",
      "./js/reader.js?v=1");

    DotNetObjectReference<ReaderView> objRef =DotNetObjectReference
      .Create(this);
    await _module.InvokeVoidAsync("registerBlazorReference",
      objRef);

    byte[] bookArray = App.StateManager.CurrentBookToByteArray();
    await _module.InvokeVoidAsync("initReader", bookArray,
      App.StateManager.CurrentBook.CFIPosition);
    }
}
```

Zdrojový kód 21

Volaná funkce skriptu reader.js pro uložení reference na stránku frameworku *Blazor* registerBlazorReference je ve zdrojovém kódu 24. Jedná se o běžnou funkci jazyka *JavaScript* avšak, aby byla viditelná pro framework *Blazor*, musí mít klíčové slovo export. Díky uložení této reference může právě *JavaScript* volat funkce stránky frameworku *Blazor* pomocí funkce invokeMethodAsync.

```
export function registerBlazorReference(dotNetHelper) {
   _blazorReference = dotNetHelper;
}
```

Zdrojový kód 22

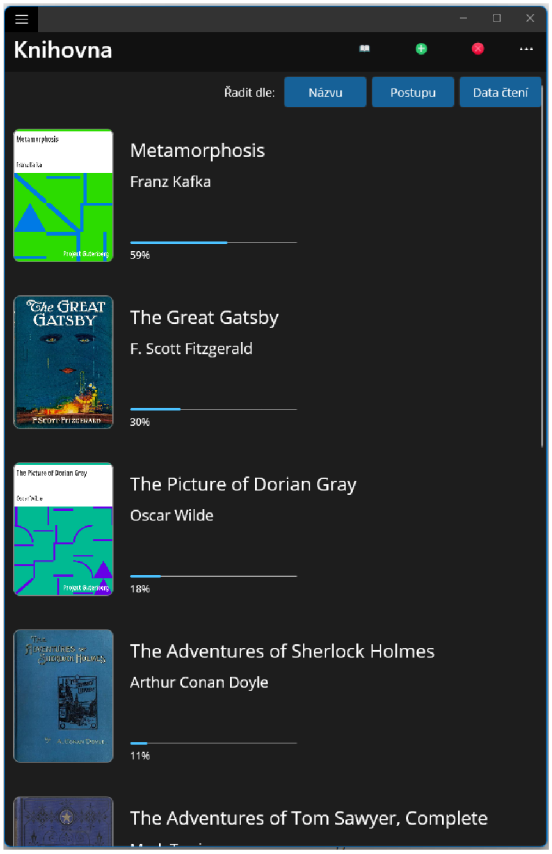
Funkce stránky vytvořené v rámci frameworku *Blazor*, které chceme, aby bylo možné volat za využití jazyka *JavaScript* musí být veřejné a musí obsahovat anotaci JSInvokable. Dále je možné libovolně přijímat parametry a reagovat na dané volání.

```
[JSInvokable]
public void LocationChanged(string position, int count, int page)
{
   if (pagesCount > 0)
   {
      App.StateManager.CurrentBook.CFIPosition = position;
      App.StateManager.CurrentBook.PagesCount = count;
      App.StateManager.CurrentBook.CurrentPage = page;
   }
}
```

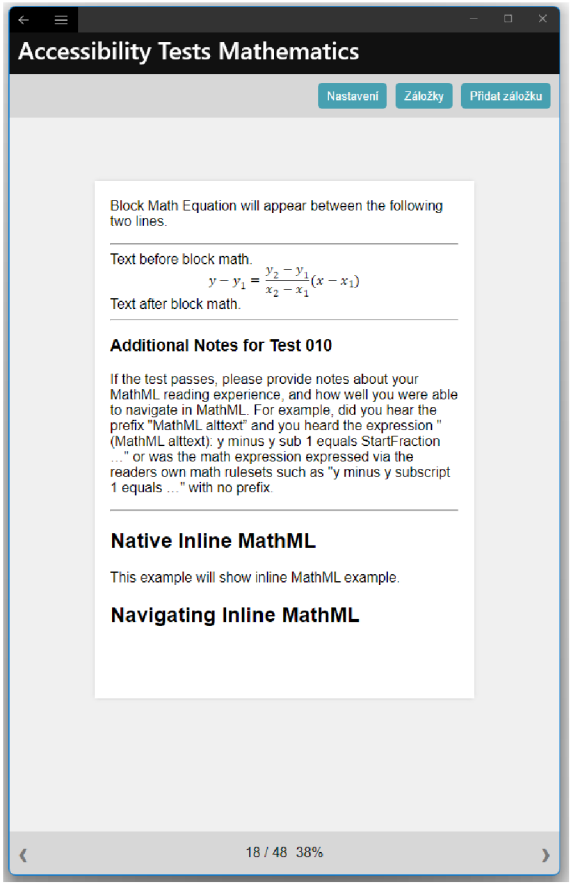
Zdrojový kód 23

4.2.10 Stránky

Pro porovnání implementovaných stránek s vytvořenými drátovými modely v rámci analýzy aplikace je na obrázku 20 ukázána stránka *Knihovna* a na obrázku 21 stránka se čtečkou knih. Tyto stránky až na některé drobné detaily odpovídají vytvořeným drátovým modelům.



Obrázek 20 - Ukázka stránky Knihovna (Vlastní zpracování)



Obrázek 21 - Ukázka stránky se čtečkou (Vlastní zpracování)

4.2.11 Nasazení aplikace

Posledním krokem vývoje je samotné nasazení aplikace, v našem případě se jedná o nasazení aplikace pro operační systémy *Android* a *Windows*. V případě operačního systému *Android* je potřeba pomocí programu *Visual Studio* nejprve vytvořit verzi aplikace k vydání a certifikát pro podepsání aplikace za účelem zamezení neoprávněné publikace aplikace. Program *Visual Studio* umožňuje vygenerovat instalační soubor aplikace s příponou *aab* určený pro distribuci v rámci obchodů s aplikacemi, či s příponou *apk* pro přímou instalaci mimo tyto obchody. Při distribuci aplikace například prostřednictvím obchodu s aplikacemi *Google Play* je potřeba založit u obchodu obchodní účet a dále postupovat při nasazování podle pokynů obchodu.

Pro nasazení aplikace určené pro operační systém *Windows* lze vytvořit a konfigurovat pomocí vývojového prostředí instalační balíček aplikace typu *MSIX*, který je dále možné distribuovat prostřednictvím obchodu *Microsoft Store*, či jinými způsoby.

4.2.12 Možnosti implementace rozšíření

Jak již bylo zmíněno, navržená a implementovaná aplikace je první verze aplikace obsahující především základní funkcionality čtečky elektronických knih. Aby se dostala na úroveň reálných produkčních aplikací je potřeba aplikaci doplnit o další funkce.

Aplikaci by tak bylo vhodné například rozšířit o podporu více platforem. V případě implementace podpory systémů *macOS* a *iOS* by nemělo dojít k žádným komplikacím, neboť jsou tyto systémy přímo podporované frameworkem *.NET MAUI*. O něco složitější by byla implementace podpory pro *Linuxové systémy*, neboť ty jsou v této době podporované frameworkem pouze v rámci komunitního rozšíření frameworku. Bylo by také možné přidat podporu pro webové prohlížeče, v tomto případě by však bylo potřeba téměř celou aplikaci vyvinout pomocí frameworku *Blazor* a verzi pro webové prohlížeče omezit o nativní funkcionality a publikovat ji v rámci samotného projektu.

Dále by bylo možné rozšířit aplikaci o funkci vyhledávání slov v rámci knih, funkci předčítání textu pomocí vestavěné třídy frameworku *.NET MAUI* TextToSpeech, podporu dalších formátů knih pomocí rozšiřujících knihoven, přístupnost pomocí sémantických vlastností jazyka *XAML* a serverovou část s vlastním uložištěm, které by se synchronizovalo s lokálním uložištěm aplikace pomocí frameworku *Microsoft Sync Framework*. Také by bylo vhodné doplnit obchodní model aplikace.

5 Výsledky a diskuse

Tato kapitola je věnována zhodnocení praktického využití, možností a kvality frameworku *.NET MAUI* doplněného frameworkem *Blazor* pro tvorbu multiplatformních aplikací z vývojářského hlediska. Pro hodnocení byly stanoveny autorem na základě poznatků z předchozích kapitol a informačních zdrojů jednotlivé charakteristiky, ty byly stanoveny se snahou udržení celkového pohledu na problém a zhodnocení softwaru z většího množství úhlů pohledu.

Problémem tohoto zhodnocení je skutečnost, že nedochází k přímému hodnocení vytvořené testovací aplikace v kapitole 4. Ta slouží pouze k získání informací pro hodnocení celého vývojářského softwaru, který byl pro vytvoření testovací aplikace využit. Tento fakt dává prostor pro nepřesné a subjektivní hodnocení, neboť pro toto hodnocení je testovací aplikace zatížena chybou existence dalších prvků, jako jsou další využité technologie, chyby v kódu a další. Zhodnocení je rozděleno do dvou oblastí, tedy prvotní zhodnocení a zhodnocení vývoje.

5.1 Prvotní zhodnocení

První část hodnocení je věnována celkovému pohledu na framework *.NET MAUI* a možnostem, které pro vývoj nabízí, neboť na základě těchto informací se vývojář prvotně rozhoduje, zda danou technologii využije. Jedná se tak o zhodnocení nabízených možností, výhod proti konkurenci, velikosti komunity a další.

5.1.1 Možnosti využití

Hodnocením možností využití jsou myšleny možné oblasti vývoje, technologické pozadí a nabízené funkcionality. Velkou výhodou frameworku je právě jeho technologické pozadí, neboť je přímou součástí velké platformy technologií *.NET*. Při jeho využívání je tak přístupné velké množství technologií, knihoven a funkcionalit. Naopak framework *.NET MAUI* dobře rozšiřuje ekosystém platformy *.NET* a vývojářům platformy *.NET* zjednodušuje tvorbu multiplatformních aplikací. Framework také vhodně rozšiřuje a zjednodušuje možnosti svého předchůdce *Xamarin.Forms* o více multiplatformní přístup, jednodušší práci v rámci projektu a další funkcionality. Framework je však velmi podobný frameworku *Xamarin.Forms* a tak není pro vývojáře přechod na framework *.NET MAUI* příliš náročný. Framework *.NET MAUI* tak nabízí mnoho možností a oblastí, kdy jej pro tvorbu aplikace lze

vhodně využít. Existují však i specifické situace, kdy je vhodnější využít například nativní vývojové nástroje než *.NET MAUI*, takovým případem mohou být například graficky náročné aplikace.

Bohužel je potřeba frameworku vytknout nedostatky, které některé konkurenční multiplatformní technologie nabízejí. Takovým nedostatkem je například chybějící oficiální podpora některých významných platforem jako je operační systém *Linux*. Některé multiplatformní nástroje také nabízejí přímou tvorbu webových aplikací, zatímco u frameworku *.NET MAUI* je potřeba využít framework *Blazor* a webovou aplikaci publikovat pomocí samostatného webového projektu. Frameworku lze také v celkovém pohledu vyčíst, že na relativně nový framework příliš nerozšířil možnosti prostředí multiplatformních nástrojů, neboť nepřinesl moc inovací.

5.1.2 Dostupnost rozšíření a informačních zdrojů

Další pohled hodnocení se zaměřuje na podporu technologie, možnosti technologického rozšíření vytvářených projektů, velikost komunity a dostupnost informačních zdrojů. Jak již bylo zmíněno, výhodou frameworku je jeho zasazení do *.NET* platformy a správa společností *Microsoft*, čímž je zajištěna podpora, velká škála dostupných doplňkových technologií, funkcionalit a informačních zdrojů. Dokumentace a informační zdroje ohledně frameworku *.NET MAUI* od společnosti *Microsoft* jsou na dostatečné úrovni, popisují většinu funkcionalit technologie a během vývoje nebylo potřeba ve většině případů využívat jiné informační zdroje. Mimo to je však velmi důležitá i vývojářská komunita ohledně frameworku, neboť také může poskytovat různá rozšíření, knihovny a informační zdroje. Jelikož se jedná o relativně nový framework, lze předvídat, že komunita ohledně frameworku bude zatím menší a nestihla ještě vytvořit velké množství rozšíření a informačních zdrojů. Framework je však přímou součástí *.NET* platformy, vychází z technologie *Xamarin.Forms* a podporuje webové technologie prostřednictvím frameworku *Blazor*. Nabízí se tak dobrá příležitost pro růst komunity spojením komunit ohledně těchto technologií.

Významnou výhodou frameworku je také jeho možnost přidání hybridního přístupu prostřednictvím frameworku *Blazor*. díky čemuž dojde právě k propojení frameworku *.NET MAUI* s frameworkem *Blazor* a prostředím webových technologií. Toto propojení je velmi flexibilní a nabízí mnoho možností, jak jej využít. Mimo to také zpřístupňuje další dostupné knihovny a rozšíření pro tvorbu aplikací.

5.2 Zhodnocení vývoje

Druhá část hodnocení je již věnována zhodnocení vývoje aplikace od přípravy projektu po nasazení.

5.2.1 Příprava a správa projektu

Prvním krokem při tvorbě aplikace v novém prostředí je příprava projektu. Její náročnost je v rámci *.NET* platformy velmi kladně hodnocena. Stačí pouze nainstalovat vývojářské prostředí *Visual Studio*, v rámci, kterého lze dále doinstalovat potřebné balíčky pro tvorbu určitého druhu aplikací. Na základě vybraného druhu aplikace dále program připraví strukturu projektu. Do vytvořeného projektu lze dále v rámci programu vkládat další moduly, rozšíření a knihovny. Projekt lze tak například pro zlepšení udržovatelnosti jednoduše doplnit o jednotkové testy. Není tak ani po celou dobu vývoje většinou potřeba využívat jiné programy.

Dalším krokem vývoje je správa projektu a jeho struktury, která je na začátku automaticky vygenerována programem *Visual Studio*, čímž dojde ke značnému zjednodušení. Oproti předchozí technologii *Xamarin.Forms* se také struktura zjednodušila z roztroušené struktury více projektů zvlášť pro každou platformu na jeden, který obsahuje všechny části aplikace pro všechny cílené platformy. V základu je tak aplikace soustředěna v jediném projektu, který lze dále modifikovat a strukturovat podle potřeby do dalších složek, projektů či dokonce řešení. Správa projektu a jeho struktury je tak jednoduchá a nabízí dostatečné možnosti pro další dělení.

5.2.2 Tvorba logiky aplikace a uživatelského rozhraní

Po přípravě prostředí a projektu se dále může vývojář zaměřit na samotnou tvorbu logiky aplikace a uživatelského rozhraní. Zdrojový kód aplikace je souhrnně tvořen prostřednictvím technologií *C#, XAML* a v případě modelu *Blazor Hybrid* ještě webových technologií. Nový vývojáři se tak musí naučit používat velké množství technologií, včetně frameworku *.NET* a případně i modelu *MVVM*. Jazyk *C#* umožňuje jak tvorbu logiky, tak i uživatelského rozhraní a nabízí mnoho funkcionalit pro jednoduchý vývoj a přehlednou správu kódu, díky čemuž lze poměrně dobře předvídat chování programu. Avšak pro nové vývojáře může být složitý například vývoj asynchronních funkcí. Více problematický bývá jazyk *XAML* pro tvorbu uživatelského rozhraní, ten je nestandardní, náročný pro nové vývojáře a není k němu doplněn žádný *WYSIWYG* editor. Naopak silnou stránkou

frameworku je minimální potřeba tvorby funkcí specifických pro dané platformy a z toho vyplívající dobrá úroveň znovupoužitelnosti. Kladně lze také hodnotit skutečnost, že množství kódu nesouvisejícího s vykonávanými funkcionalitami aplikace, zejména určeného pro nastavení prostředí je minimální.

5.2.3 Rychlost vývoje a možnosti ladění aplikace

Tato část je věnována dvěma klasifikacím, které úzce souvisí s procesem tvorby logiky aplikace a uživatelského rozhraní. První klasifikace se zaměřuje na rychlost vývoje aplikace, co se týče chování a možností daného prostředí. Překlad tvořené aplikace trvá relativně dlouhou dobu a s dalším rozšiřováním aplikace a struktury projektu se tato doba dále prodlužuje. Tento problém částečně řeší funkce *Hot reload*, která při ladění aplikace rovnou aplikuje provedené změny bez překládání kódu celé aplikace. Tato funkce reaguje jak na změny uživatelského rozhraní prostřednictvím jazyka *XAML*, tak i na změny logiky vytvořené v jazyce *C#*. Při vývoji testovací aplikace tato funkce významně zrychlovala vývoj, často se však stávalo, že se dané změny neprojevily a bylo potřeba stejně celou aplikaci přeložit.

Druhá klasifikace hodnotí možnosti vývojového prostředí pro ladění aplikace. *Visual Studio* nabízí bohaté ladící prostředí a aplikace lze ladit jak pomocí emulátorů, tak i na reálných zařízení.

Nepříjemností vývoje je omezení kompilace a ladění aplikace napříč některými operačními systémy. Není tak možné na samotném počítači s operačním systémem *Windows* kompilovat a ladit verze aplikace určené pro operační systémy *macOS* a *iOS*. Naopak na počítači s operačním systémem *macOS* není možné kompilovat verze aplikace pro systémy *Windows* a *Android*. Tato nepříjemnost je však tvořena zejména právním omezením a vyskytuje se ve všech nástrojích pro vývoj multiplatformních, či hybridních aplikací. *Visual Studio* jako řešení tohoto problému nabízí možnost připojení přes síť k dalšímu počítači s daným operačním systémem, na který se odesílá kód ke kompilaci.

5.2.4 Splnění požadavků na aplikaci

Následující hodnocení je věnováno zhodnocení splnění požadavků na chování a implementaci na testovací aplikaci, které bylo stanoveno v rámci analýzy v kapitole 4 a následně implementováno. Jedná se tak o hodnocení, jak je framework pro vývoj dané aplikace vhodný a zda podporuje tvorbu daných funkcionalit. Chování vytvořené testovací

aplikace odpovídá stanoveným požadavkům a bylo tak možné implementovat funkcionality jako je zobrazení knih typu *EPUB* pro čtení, podpora více platforem, nativní prvky uživatelského rozhraní, navigace v rámci aplikace, práce s nativními moduly, podpora více jazyků, notifikace, lokální ukládání dat, komunikace se vzdáleným serverem, využití asynchronních metod, aby nedocházelo k blokaci uživatelského rozhraní, ošetření a reagování na chyby a další.

Většinu funkcionalit testovací aplikace bylo možné jednoduše implementovat pomocí vestavěných funkcí frameworku *.NET MAUI*, či platformy *.NET*. O něco složitější byla implementace čtení knih formátu *EPUB*, jedná se však o velmi specifickou problematiku a byla řešena pomocí komunitních knihoven. Pomocí komunitní knihovny byly také řešeny lokální notifikace, pro které framework .NET MAUI nenabízí žádnou vestavěnou multiplatformní podporu. Použitá komunitní knihovna pro lokální notifikace navíc v současné době podporuje pouze operační systémy *Android* a *iOS*. V rámci testovací aplikace, tak využití lokálních notifikací nabízí pouze verze aplikace pro systém *Android*. Je však možné dodatečně doplnit podporu pro další systémy pomocí specifického kódu pro dané platformy. Konečně lze usoudit, že využití frameworku *.NET MAUI* pro tvorbu dané testovací aplikace je i přes některé problémy vhodné a nabízí mnoho výhod oproti využití nativních technologií pro každou platformu zvlášť.

5.2.5 Nasazení

Tvorba instalačních balíčků a následná distribuce aplikace je v rámci frameworku *.NET MAUI* opět relativně jednoduchá, neboť program *Visual Studio* nabízí pro každou platformu specifické potřebné funkce pro realizaci nasazení aplikace.

5.3 Závěr hodnocení

Využití frameworku *.NET MAUI* doplněného frameworkem *Blazor* za účelem tvorby multiplatformních aplikací je i přes některé většinou řešitelné problémy a nedostatky ve výsledku hodnoceno kladně a jedná se tak v mnoha případech o zajímavou a výhodnou náhradu technologií nativního vývoje. Jelikož se jedná o relativně ranou technologii, lze také do budoucna očekávat významný vývoj, který některé jeho nedostatky vyřeší. Je však potřeba zdůraznit skutečnosti, že všechny požadavky na testovací aplikaci se nepovedlo ideálně splnit a že hodnocení samotného frameworku bylo zatíženo chybou existence dalších prvků a dalšími chybami.

Vývoj popularity frameworku mezi vývojáři nelze v současné době předvídat. Existuje totiž relativně velké množství dalších technologií pro vývoj multiplatformních technologií, které nabízí podobné možnosti. Framework *.NET MAUI* má potenciál zejména zaujmout stávající vývojáře platformy *.NET*. Zajímavý tak bude pravděpodobně především pro vývojáře využívající technologii *Xamarin.Forms*, ze které se framework *.NET MAUI* vyvinul a vývojáře využívající technologii *Blazor*, kterou framework *.NET MAUI* přímo podporuje. Podobnou otázku na celkový budoucí vývoj a budoucí vývoj popularity lze klást i na samotné přístupy vývoje multiplatformních, či hybridních aplikací. Kromě těchto přístupů jsou populární i další alternativní přístupy vývoje vůči nativnímu vývoji jako je vývoj webových, či progresivních webových aplikací. Lze však očekávat, že i přes některé nedostatky se budou tyto přístupy dále rozvíjet.

6 Závěr

V diplomové práci byly popsány a zhodnoceny dle zvolených kritérií frameworky *.NET MAUI* a *Blazor* určené pro multiplatformní vývoj aplikací včetně jejich možností a praktické využitelnosti na základě zjištěných poznatků z odborné literatury a vývoje testovací multiplatformní aplikace.

Teoretickou část práce lze rozdělit na dvě další části. První část byla věnována prostředí operačních systémů a přístupům k vývoji. Nejprve tak byly definovány počítačové platformy, včetně jejich uživatelského zastoupení. Dále byly popsány nejpoužívanější operační systémy, na které dané vývojové technologie nejčastěji cílí, se zaměřením na vývoj a distribuci aplikací. Následně bylo možné popsat jednotlivé přístupy k vývoji multiplatformních aplikací, včetně jejich výhod a nevýhod.

Druhá část teoretické práce popisuje samotnou platformu *.NET* včetně jejích frameworků *.NET MAUI* a *Blazor*. U platformy *.NET* byly popsány její verze, programovací jazyky, technologie a další nástroje. Popis frameworku *.NET MAUI* byl zaměřen především na jeho možnosti, architekturu a možnost tvorby hybridních aplikací pomocí modelu *Blazor Hybrid*. Nakonec byl popsán framework *Blazor* včetně jeho možností a modelů.

Následující praktická část práce byla věnována analýze a implementaci testovací multiplatformní aplikace. Během analýzy byly stanoveny požadavky na chování a implementaci aplikace, tak aby došlo k dostatečnému otestování možností využitých frameworků. Následně byly navrženy další možnosti rozšíření aplikace, její architektura, uživatelské rozhraní a datový model. Kapitola popisující implementaci testovací aplikace nejprve popisuje samotnou přípravu projektu, jeho struktury a modulů. Zbytek je věnován implementaci jednotlivých prvků jako je architektura s *MVVM* modelem, navigace napříč aplikací, nativní prvky, práce s daty a další. Nakonec je naznačen postup nasazení aplikace a možnosti implementace rozšíření aplikace.

V poslední kapitole byl zhodnocen framework *.NET MAUI* doplněný frameworkem *Blazor* na základě zvolených kritérií, stanovených s účelem udržení celkového pohledu na problém a zhodnocení z dostatečného množství úhlů pohledu. U frameworků byla nalezena pozitiva i negativa. I přes některé nedostatky a problémy bylo závěrem využití těchto frameworků pro tvorbu multiplatformních aplikací hodnoceno kladně.

7 Seznam použitých zdrojů

.NET documentation. Microsoft Docs [online]. Microsoft Corporation [cit. 2023-01-20]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui

BAPTISTA, Gabriel a Francesco ABBRUZZESE. Software Architecture with C# 10 And . NET 6. 3. Birmingham: Packt Publishing, Limited, 2022. ISBN 9781484259283.

BIGELOW, Stephen J. What is a platform?. *TechTarget* [online]. 2021 [cit. 2022-12-06]. Dostupné z: https://www.techtarget.com/searchitoperations/definition/platform

ČÁPKA, David. Lekce 1 - Úvod do C# a .NET frameworku. *Itnetwork* [online]. [cit. 2023-01-21]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

DARWIN, Ian F. *Android Cookbook*. 2nd edition. Gravenstein Highway North, Sebastopol: O'Reilly Media, 2017. ISBN 9781449374433.

HIMSCHOOT, Peter. Microsoft Blazor. 2. Melle: Apress L. P., 2020. ISBN 9781484259283.

HOMZA, Martin. *Alternativy jazyka JavaScript v prostředí webu na straně klienta*. Praha, 2021. Bakalářská práce. Česká zemědělská univerzita v Praze. Vedoucí práce Doc. Ing. Vojtěch Merunka, Ph.D.

KIDECHA, Sanjay. Native vs Hybrid vs Cross-Platform: How and What to Choose?. *Dzone.com* [online]. 27.7.2022 [cit. 2022-11-05]. Dostupné z: https://dzone.com/articles/native-vs-hybrid-vs-cross-platform-how-and-what-to

KLOSOWSKI, Thorin. I Want to Write iOS Apps. Where Do I Start?. *Lifehacker* [online]. 2014 [cit. 2022-12-06]. Dostupné z: https://lifehacker.com/i-want-to-write-ios-apps-where-do-i-start-1644802175

KOĎOUSKOVÁ, Barbora. Co jsou progresivní webové aplikace (PWA) a jaké mají výhody: 18.10.2021. *Rascasone.com* [online]. [cit. 2022-11-01]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody

Kotlin. What is cross-platform mobile development?: 6.9.2022. *Kotlinlang.org* [online]. Praha: Jetbrains [cit. 2022-11-01]. Dostupné z: https://kotlinlang.org/docs/cross-platform-mobile-development.html

LINFO. Platform Definition. *The Linux Information Project* [online]. [cit. 2022-11-05]. Dostupné z: http://www.linfo.org/platform.html

PATEL, Tapan. How To Choose A Web Technology Stack: A Complete Guide. *ThirdRockTechkno* [online]. 19.5.2020 [cit. 2023-01-17]. Dostupné z: https://www.thirdrocktechkno.com/blog/how-to-choose-a-technology-stack-for-web-application-development/

POSEY, Brian. Apple iOS. *TechTarget* [online]. 2022 [cit. 2022-12-06]. Dostupné z: https://www.techtarget.com/searchmobilecomputing/definition/iOS

POWALOWSKI, Kamil. Past, present and future of native macOS app development. *10clouds* [online]. 2020 [cit. 2022-12-06]. Dostupné z: https://10clouds.com/blog/mobile/native-macos-app-development/

PRICE, Mark J. C# 10 and . NET 6 - Modern Cross-Platform Development. 6. Birmingham: Packt Publishing, Limited, 2021. ISBN 9781801076968.

RAMEL, David. Windows 11 Development: Open Ecosystem Store, Project Reunion Rebrand and More. *Visual Studio Magazine* [online]. 2021 [cit. 2022-12-06]. Dostupné z: https://visualstudiomagazine.com/articles/2021/06/24/windows-11.aspx

ŘÍHA, Pavel. Lekce 1 - Struktura prostředí .NET. *Itnetwork* [online]. [cit. 2023-01-19]. Dostupné z: https://www.itnetwork.cz/csharp/historie-net/net-struktura-prostredi

SHIOTSU, Yoshitaka. What Is a Hybrid App? (Detailed Guide for 2022). *Upwork* [online]. 22.11.2021 [cit. 2023-01-17]. Dostupné z: https://www.upwork.com/resources/hybrid-app

STONIS, Michael. *Enterprise Application Patterns using .NET MAUI* [online]. Redmond, Washington: Microsoft Corporation, 2022 [cit. 2023-02-15]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/architecture/maui/

Statcounter Global Stats [online]. [cit. 2022-12-06]. Dostupné z: https://gs.statcounter.com

VOBORNÍK, Petr. *Technologie .NET: 2022* [online]. [cit. 2023-01-19]. Dostupné z: https://view.officeapps.live.com/op/view.aspx?src=https%3A%2F%2Fraw.githubusercontent.com%2FPetrVobornik%2Fprednasky%2Fmaster%2FXamarin.Forms%2F01-OOP-a-NET%2Ftechnologie-net.ppsx&wdOrigin=BROWSELINK

VÁVRŮ, Jiří a Miroslav UJBÁNYAI. *Programujeme pro Android*. 2., rozš. vyd. Praha: Grada, 2013. Průvodce (Grada). ISBN 978-80-247-4863-4.

WINDMILL, Eric. *Flutter in Action*. 1. New York: Manning Publications, 2019. ISBN 9781617296147.

What is a Web Application?. *Stackpath.com*. Stack Path [online]. [cit. 2022-11-05]. Dostupné z: https://www.stackpath.com/edge-academy/what-is-a-web-application/

What Is Linux. *Javatpoint* [online]. [cit. 2022-12-06]. Dostupné z: https://www.javatpoint.com/what-is-linux

8 Seznam obrázků, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam grafů

8.3 Seznam použitých zkratek

AJAX – Asynchronous JavaScript and XML

AOT – Ahead of time

API – Application Programming Interface

BCL – Base Class Library

CIL – Common Intermediate Language

CLR – Common Language Runtime

CSS – Cascading Style Sheets

DOM – Document Object Model

EPUB – Electronic publication

GUI – Graphical user interface

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JIT – Just in Time

JSON – JavaScript Object Notation

LINQ – Language Integrated Query

MAUI – Multi-platform App User Interface

MVVM – Model-view-viewmodel

PWA – Progressive Web App

REST – Representational State Transfer

SPA – Single-page application

SQL – Structured Query Language

UI – User interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

UWP – Universal Windows Platform

WPF – Windows Presentation Foundation

WYSIWYG – What you see is what you get

XAML – Extensible Application Markup Language

Přílohy

Součástí elektronické verze je projekt vytvořené aplikace v souboru zip.