VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTEGRACE DOTVVM DO .NET MAUI

INTEGRATION OF DOTVVM INTO .NET MAUI

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ MIKEŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144992

Ústav: Ústav informačních systémů (UIFS)
Student: **Mikeš Tomáš**
Program: Informační technologie
Specializace: Informační technologie
Název: **Integrace DotVVM do .NET MAUI**
Kategorie: Webové aplikace
Akademický rok: 2022/23

Zadání:

1. Seznamte se s problematikou vývoje hybridních aplikací.
2. Porovnejte různé přístupy v práci s aplikačními rámci používanými pro vývoj hybridních aplikací na úrovni spouštění, hostování a běhu aplikací. Zaměřte se na to, jakým způsobem aplikace pracují se systémovými prostředky, vykreslováním komponent a samotným provozem aplikace.
3. Analyzujte problematiku integrace webového aplikačního rámce DotVVM pro hostování v režimu hybridní aplikace s technologií .NET MAUI.
4. Dle výsledků analýzy navrhněte vhodné řešení integrace webového aplikačního rámce DotVVM s technologií .NET MAUI.
5. Implementujte navržené řešení.
6. Porovnejte vlastní implementaci se zkoumanými aplikačními rámci z bodu 2 a vytvořte s pomocí DotVVM vzorovou aplikaci na platformě .NET MAUI, která bude demonstrovat aspekty vývoje hybridních aplikací.

Literatura:

- Panhale, M. (2016). *Beginning Hybrid Mobile Application Development.* Apress.
- Wargo, J. M. (2017). *Apache Cordova 4 Programming.* Custom Publishing.
- Griffith, C. W. (2017). *Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova.* O'Reilly Media, Inc.
- Microsoft. (2022). *NET Multi-platform App UI documentation.* Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/ [cit. 2022-10-25].
- DotVVM. (2022). *DotVVM Documentation.* Dostupné z: https://www.dotvvm.com/docs [cit. 2022-10-25].

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 25.10.2022

Abstrakt

Cílem této práce je integrace technologií DotVVM a .NET MAUI, která má umožnit jejich vzájemnou komunikaci a možnost vykreslení webových stránek vyvinutých v DotVVM uvnitř multiplatformní aplikaci .NET MAUI. Součástí je popis implementovaného řešení založeného na komponentě WebView a také vzorové aplikace demonstrující hybridní způsob vývoje. Praktická část zahrnuje analýzu způsobů vývoje mobilních aplikací, aplikačního rámce DotVVM a .NET MAUI. Výsledky této práce umožňují aplikacím vyvinutým v DotVVM fungovat uvnitř multiplatformní aplikace a přistupovat tak k nativním funkcionalitám daných platforem.

Abstract

The goal of this bachelor thesis is to integrate DotVVM and .NET MAUI technologies to enable their mutual communication and the ability to render web pages developed in DotVVM within a multiplatform .NET MAUI application. Part of the thesis describes the implementation of the solution based on WebView component, as well as sample application demonstrating hybrid developement approach. The practical part is preceded by analysis of mobile application developement approaches, the DotVVM and .NET MAUI frameworks. The results of this thesis allow applications developed using DotVVM to run within multiplatform application and access the native functionalities of the platforms.

Klíčová slova

DotVVM, .NET MAUI, .NET, MVVM, WebView, Hybridní aplikace

Keywords

DotVVM, .NET MAUI, .NET, MVVM, WebView, Hybrid application

Citace

MIKEŠ, Tomáš. *Integrace DotVVM do .NET MAUI*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Integrace DotVVM do .NET MAUI

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D., a další informace mi poskytl Bc. Tomáš Herceg. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Tomáš Mikeš
5. května 2023

Poděkování

Poděkování patří vedoucímu mé práce Ing. Jiřímu Hynkovi, PhD. za hodnotnou zpětnou vazbu v průběhu zpracování této práce a Bc. Tomáši Hercegovi za odborné konzultace, zejména u rozšiřování aplikačního rámce DotVVM.

Obsah

1 Úvod **2**

2 Síťová komunikace webových aplikací **4**
2.1 Architektura klient-server 4
2.2 Webový prohlížeč 5
2.3 Protokol HTTP 6

3 Vývoj mobilních aplikací **8**
3.1 Progresivní webové aplikace 9
3.2 Hybridní aplikace 11
3.3 Interpretované aplikace 14
3.4 Kompilované aplikace 16

4 Analýza **21**
4.1 DotVVM 22
4.2 Komponenta WebView 28

5 Návrh **30**

6 Implementace **33**
6.1 Registrace DotVVM 33
6.2 Služba DotVVM Middleware 34
6.3 Přístup k lokálním souborům 36
6.4 Integrace komunikace 37

7 Testování **39**
7.1 Návrh vzorové aplikace 39
7.2 Implementace vzorové aplikace 41

8 Závěr **45**

Literatura **46**

Kapitola 1

Úvod

Představme si, že vyvíjíme software pro zákazníka, který si dle analýzy žádá rozsáhlou desktopovou aplikaci skládající se z komplexních grafických komponent. Tyto komponenty vytvoříme pomocí knihovny určeného pro vývoj uživatelského rozhraní, jenž je specifický pro vývoj desktopových aplikací. Později se však zákazník rozhodne, že by potřeboval zmíněné komponenty využít i ve své webové aplikaci a my jsme tedy nuceni všechny komponenty uživatelského rozhraní znovu vytvořit pomocí webové technologie. Takový vývoj stojí spoustu času, režie a také financí.

Řešení problému se snaží přinést koncept hybridní aplikace, který je založen na webových technologiích (HTML, CSS a javascript). Tyto aplikace jsou definovány jako multiplatformní, jelikož sdílí stejný zdrojový kód a mohou fungovat na více platformách. Kompatibilitu s více operačními systémy zajišťuje nasazení aplikace do nativního kontejneru využívajícího objektu WebView, který umožňuje zobrazovat webové stránky a oproti prohlížeči neobsahuje žádnou z funkcí, jako je navigační menu či vstupní pole s URL. V porovnání s progresivními webovými aplikacemi (PWA) dokáže využít i nativních funkcionalit, které webový prohlížeč nenabízí. Může mezi ně patřit například připojení k bluetooth, snímače přiblížení nebo také využití biometrické autentizace, jako je Touch ID, nebo Face ID u iOS.

Cílem této práce je provést integraci mezi webovým frameworkem DotVVM a nově vznikajícím multiplatformním frameworkem .NET MAUI tak, aby pomocí něj bylo možné vytvářet hybridní aplikace. Integrací je myšlena tvorba komunikačního kanálu mezi DotVVM, .NET MAUI a nativní WebView komponentou specifickou pro platformy Windows, Android a iOS umožňující vzájemné fungování. Součástí integrace je modifikace frameworku DotVVM takovým způsobem, aby mohl být na cílených platformách využit. V důsledku je umožněno, aby nativní prvky vytvořené v .NET MAUI dynamicky ovlivňovaly kontext, resp. stav DotVVM a naopak. Pro demonstraci reálného využití integrace je výsledkem také vzorová aplikace se základními i mírně pokročilejšími funkcionalitami.

Kapitola 2 pojednává o základních principech komunikace webových aplikací, stručně je shrnuje a popisuje protokol HTTP, kterého se v praktické části využívá. Další kapitola 3 popisuje způsoby vývoje mobilních aplikací, jejich různé typy a obecné fungování. Jsou zde rozebrány progresivní webové, hybridní, interpretované a kompilované aplikace včetně aplikačních rámců založených na jejich principech. Větší důraz je kladen na pojem hybridní aplikace, jelikož je tématem praktické části. V následující kapitole 4.1 je rozebrán framework DotVVM, jeho použití, interní architektura a fungování. Kapitola 4 se zaměřuje na analýzu způsobu hostování webového frameworku v .NET MAUI a je zde podrobně popsána klíčová systémová komponenta *WebView*, pomocí níž je následná integrace webového frameworku DotVVM provedena. Dále navazuje kapitola 5, jež se věnuje návrhu integrace obou fra-

meworků a je rozdělena do podkapitol, které se zaměřují zejména na rozdíly v integraci pro specifické platformy, tj. Windows, Android, iOS. Kapitola 6 popisuje implementační postup navrženého řešení, jehož funkčnost je poté prezentována na vzorové aplikace popsané v poslední kapitole 7.

Kapitola 2

Síťová komunikace webových aplikací

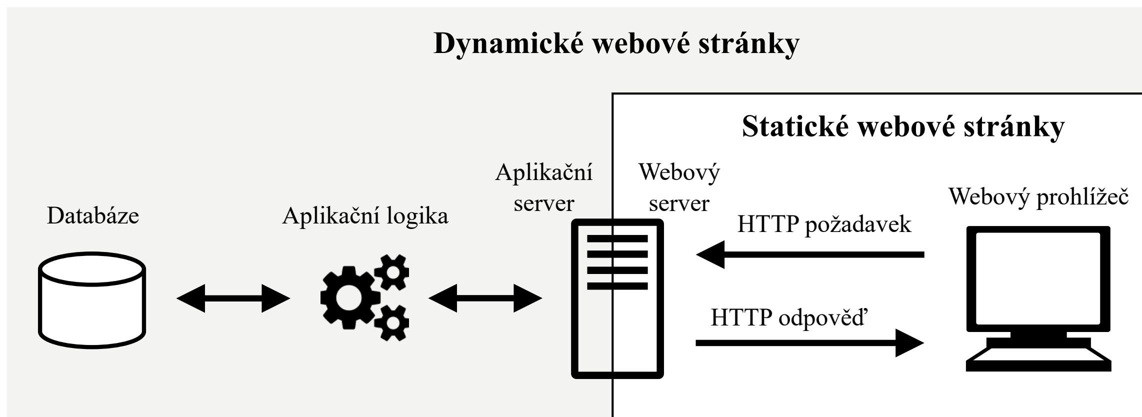
Webová aplikace je aplikace typu klient-server, která typicky používá webový prohlížeč jako klienta. Pro korektní návrh a vývoj webových aplikací je nutné chápat princip internetového protokolu HTTP (sekce 2.3), na němž jsou webové technologie postaveny. V této kapitole je rozebrána architektura webových aplikací zejména dle [18] a jsou zde vymezeny důležité pojmy, které se vyskytují i v následujících kapitolách.

2.1 Architektura klient-server

Při vývoji webových aplikací se běžně používá pojem klient-server, což je softwarová architektura tvořena klientem a serverem, přičemž klient odesílá požadavky na server, který je dále zpracovává a reaguje na ně [38]. Jak uvádí [3], server je tvořen ze serverového hostitele (tzv. *server host*) představujícího fyzické zařízení s přístupem do internetu, a serverové aplikace reprezentující program, který dokáže obsluhovat současně více klientských požadavků. Stejně tak je klient sestaven z klientského hostitele (tzv. *client host*) a z klientské aplikace představující softwarový program, který iniciuje relaci se serverem. Příkladem aplikací webového serveru jsou *Internet Information Services* (IIS) a *Apache HTTP Server*. Mezi aplikace webového klienta lze zařadit webové prohlížeče (sekce 2.2) jako jsou *Microsoft Edge*[1], *Google Chrome*[2] či *Mozilla Firefox*[3].

Webové stránky mohou být dle [3] statické nebo dynamické a jak lze vidět na obrázku 2.1, u statických webových stránek se jejich obsah nemění v závislosti na požadavku a je tedy vždy identický. Na rozdíl od dynamických webových stránek je v procesu vyžadován pouze webový server. U dynamických webových stránek se HTTP požadavek zpracovává aplikačním serverem, který na základě aplikační logiky (tzv. *business logic*) vygeneruje příslušný HTML kód a předá ho zpět webovému serveru, který ho přiloží k odchozí HTTP odpovědi. V procesu zpracování požadavku může vystupovat rovněž databázový server zajišťující perzistentní úložiště, s kterým komunikuje aplikační server.

[1] Microsoft Edge: `https://www.microsoft.com/cs-cz/edge`
[2] Google Chrome: `https://www.google.com/intl/cs_CZ/chrome/`
[3] Mozilla Firefox: `https://www.mozilla.org/cs/firefox/`



Obrázek 2.1: Schéma statických a dynamických webových stránek

Historicky jedny z prvních implementací architektury klient-server nevyužívaly neproprietárních protokolů[4], což mělo za důsledek velkou složitost klientských i serverových programů. Z toho důvodu je pro nás velice výhodná sada protokolů TCP/IP, která je dnes běžně implementována jako součást operačního systému. Poskytuje nám, mimo jiné protokoly, aplikační protokol HTTP (*Hypertext Transfer Protocol*) sloužící primárně pro přenos dokumentů. Stejně tak se nemusí u webových aplikací vytvářet vlastní klientský program pro zobrazování webových stránek, k tomu nám slouží webový prohlížeč nainstalovaný přímo v systému uživatele.

2.2 Webový prohlížeč

Webový prohlížeč je typickým a nejčastějším příkladem webového klienta. Je o něm možné získat informace pomocí objektu `navigator` ve skriptovacím jazyce *javascript*. Mezi jeho hlavní funkce patří:

1. **Vytváření a odesílání požadavků** webovému serveru na základě uživatelských akcích jako jsou: přesměrování pomocí odkazu, explicitní zadání URL nebo také vyplnění formuláře.

2. **Přijímání odpovědí** přicházejících z webového serveru a jejich interpretaci uživateli podle daného typu obsahu.

3. **Caching** neboli kešování (podrobněji rozebráno v podkapitole 2.3) sloužící k dočasnému uložení kopií dat z předchozích požadavků, a tudíž k urychlení odezvy odpovědi.

4. **Udržování stavu** (tzv. *session state*, *state maintenance* nebo *session maintenance*) dat koncového uživatele. Jelikož HTTP spadá mezi bezstavové protokoly a po tom, co klient obdrží odpověď, je spojení mezi klientem a serverem ukončeno. K udržení informací o relaci v prohlížeči slouží *cookies*, které webový server vygeneruje a požádá prohlížeč o uložení po určitou dobu. Webový prohlížeč poté přikládá cookies ke každému serverovému požadavku, který uživatel provede [7].

[4]Neproprietární protokol: Protokol, který není vlastněn žádnou společností a není vázán na žádný z jejich produktů [20].

5. **Vykreslování objektů** podle typu obsahu uvnitř okna webového prohlížeče bez nutnosti instalace dodatečného softwaru. Většina webových prohlížečů podporuje formáty *text/html*, *text/plain*, *image/gif* a *image/jpeg*. U komplexnějších objektů, jako je například audio nebo video, musí prohlížeč umožňovat otevření v externí aplikaci.

2.3 Protokol HTTP

Protokol HTTP je jeden z nejdůležitějších prvků umožňující výměnu dat mezi klientem a serverem. Definuje, jakým způsobem jsou data přenášena pomocí HTTP zprávy, jíž existují dva typy – HTTP požadavek a odpověď. Struktura HTTP požadavků i odpovědí je velice podobná, liší se zejména prvním řádkem, který udává u požadavků požadovanou akci (metodu) nebo stavový kód u odpovědí [31]. Mezi nejčastěji používané metody patří GET a POST. Obecně jsou oba typy zpráv tvořeny parametry URL, HTTP verzí, hlavičkami a tělem zprávy.

Identifikace zdrojů

Pro identifikaci jednotlivých zdrojů neboli *resources* na webovém serveru, slouží *Uniform Resource Identifier* (URI) používané napříč HTTP. Nejčastější formou URI je *Uniform Resource Locator* (URL), která je pro laiky spíše známá jako webová adresa. Formát URL [16] je značen na ukázce 2.1 a sémantika jeho jednotlivých části je následující:

1. ***scheme*** – určuje, který protokol musí webový prohlížeč použít při požadavku na daný zdroj. U webových stránek jde typicky o HTTPS (Hyper Text Transport Protocol Secure) a HTTP, ale lze se setkat také s protokoly udávajícími jinou syntaxy *file:* otvírající lokálně uložený soubor, *mailto:* otvírající mailového klienta a další.

2. ***domain*** – značící doménový název, případně IP adresu webového serveru, ke kterému přistupujeme.

3. ***port*** – je nepovinná část URL odpovídající portu, na kterém cílený webový server naslouchá. Implicitní hodnota je 80 pro HTTP a 443 pro HTTPS.

4. ***path*** – odpovídá cestě ke zdroji na odpovídajícím webovém serveru. Dříve tato cesta odpovídala lokaci webových prostředků v souborovém systému na webovém serveru, nicméně v dnešní době jde pouze o abstrakci spravovanou webovou aplikací.

5. ***query*** – je dodatečný parametr ve formátu `?key1=value1&key2=value2`, který je tvořen dvojicemi klíč-hodnota oddělenými symbolem `&`.

6. ***anchor*** – slouží jako kotvící bod v obsahu zobrazovaného zdroje. Může se jednat o určité místo na webové stránce či určitý čas ve videu či audiu.

```
scheme://domain[:port]/path/[query][#anchor]
```

Výpis 2.1: Formát URL

HTTP hlavičky

V dokumentaci internetových protokolů RFC 9110 [31] jsou specifikovány hlavičky, které jsou součástí HTTP požadavků i odpovědí a jsou umístěné před obsahem zprávy. Slouží primárně pro modifikaci a rozšíření sémantiky zpráv, mezi které patří popis odesílatele, definice příjemce nebo dodatečný kontext.

Mezi hlavičky rozšiřující obecný kontext zprávy a nikoliv konkrétního požadavku či odpovědi, mohou být zařazeny například následující:

```
Date: Wed, 21 Oct 2015 07:28:00 GMT
Connection: Close
```

U požadavků bývají často uvedeny hlavičky, kterými je klientovi umožněno přiložit dodatečné informace:

```
Host: developer.mozilla.org
Accept: text/html,application/xhtml+xml,application/xml
Referer: https://developer.mozilla.org/testpage.html
Cookie: TEST=123; PASSWD=Password123;
```

Ve zprávě s HTTP odpovědí se často vyskytují hlavičky, mezi které patří `Content-Type` skládající se typicky z typu média a kódování znaků:

```
Content-Type: text/html; charset=utf-8
Location: http://www.mywebsite.com/relocatedPage.html
Set-Cookie: mykey=myvalue; expires=Fri, 21-June-2022 01:21:00 GMT;
          Max-Age=3600; Path=/; secure
Keep-Alive: timeout=5, max=997
```

Stavové kódy

První řádek odpovědi obsahuje třímístný stavový kód a jeho stručný popis. Stavový kód slouží klientovi k identifikaci výsledku odeslaného požadavku a spadá do jedné z pěti kategorií dle počáteční číslice následovně:

- **1xx** – slouží čiště k informačním účelům.
- **2xx** – značí úspěšně zpracovaný požadavek implikující zaslání požadovaných zdrojů v odpovědi.
- **3xx** – značí nutnou akci potřebnou k dokončení požadavku. Typicky jde o přesměrování na jinou URL uvedenou v hlavičce *Location*.
- **4xx** – indikuje chybu klientského požadavku. Mezi nejčastější patří: neplatný požadavek 400 (*Bad request*), neplatná autorizace 401 (*Not Authorized*) a nenalezený zdroj 404 (*Not Found*).
- **5xx** – představuje serverové chyby, typicky 500 (*Internal Server Error*).

V této kapitole byly rozebrány základy komunikace webových aplikací, zejména protokol HTTP. Poznatky jsou uplatněny převážně v praktické části této práce počínaje kapitolou 4.1. Následující kapitola 3 porovnává typy mobilních aplikací, mezi něž patří progresivní webové a hybridní aplikace.

Kapitola 3

Vývoj mobilních aplikací

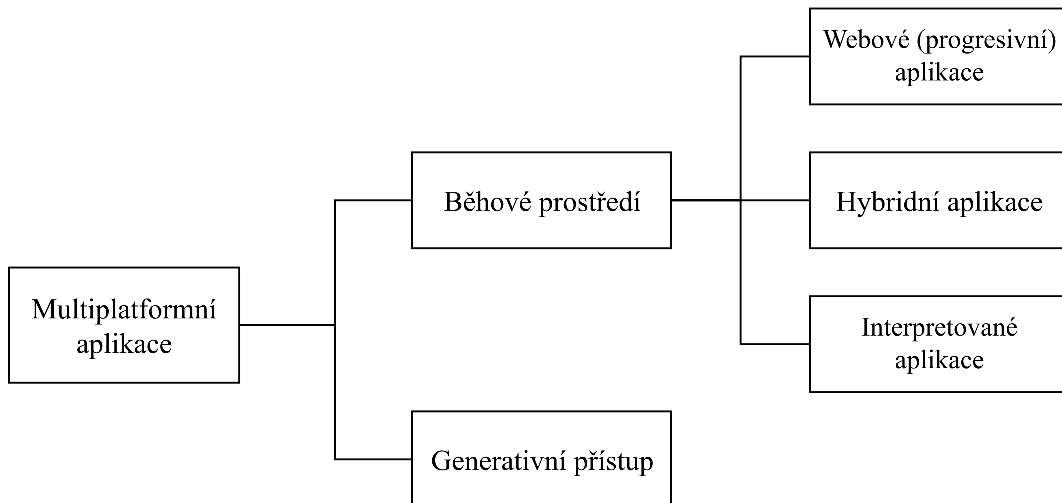
Existuje několik typů mobilních aplikací lišících se jak fungováním, tak způsobem jejich vývoje. Obvykle jsou mobilní aplikace vyvíjeny pro specifickou platformu, což znamená, že nemohou běžet na jiné platformě, než pro kterou byly navrženy. Takový typ vývoje mobilních aplikací je uváděn jako nativní přístup, jelikož jsou aplikace vyvinuty pomocí nástrojů a jazyků cílené platformy [6]. Mezi tyto jazyky patří Java pro Android, Objective-C nebo Swift pro zařízení iOS a C# a XAML pro Windows. Jak uvádí [11], výhodou nativního přístupu jsou vývojářské nástroje SDK (*Software Development Kit*) určené přímo pro specifickou platformu, díky čemuž mohou vývojáři například pracovat v IDE[1] konfigurovaných přímo pro vývoj mobilních aplikací. Spadá mezi ně Android Studio pro Android nebo Xcode pro iOS.

Další výhodou je dostupnost nativního rozhraní a funkcionalit bez nutnosti přídavných řešení třetích stran, což úzce souvisí s velmi dobrým výkonem, neboť mezi aplikací a nativním rozhraním není žádná mezivrstva, která by výkon ovlivňovala. U nativního přístupu jsou také ihned k dispozici nejnovější nativní funkcionality a není třeba čekat na aktualizace ze strany vývojářů aplikačních rámců multiplatformního vývoje [1].

Zásadní nevýhodou nativního vývoje aplikací je nutnost znalosti jazyků cílených platforem. Obvykle je aplikace vyvíjena jak pro Android, tak pro iOS, avšak žádný z klientského kódu nemůže být použit u další platformy. Znamená to tedy, že je nutné pro každou platformu napsat kód znovu, i přes to, že je žádáno stejné logiky. Takový vývoj stojí spoustu času, financí a náročnost se vztahuje i na údržbu aplikace.

V důsledku těchto nevýhod vznikly různé alternativní přístupy, mezi něž patří multiplatformní vývoj aplikací, který umožňuje implementaci aplikací spustitelných na více platformách pomocí jednoho sdíleného kódu [12]. Jak lze vidět na obrázku 3.1, tento přístup se dále rozlišuje na aplikace, které využívají prostředí běhu programu (tzv. *runtime environment*), a na ty, které jsou generovány během kompilace sdíleného kódu přímo na aplikace pro danou platformu (tzv. *generative approach*). Zatímco zdrojový kód aplikace je nezávislý na platformě, prostředí běhu programu musí být specifické pro určitou platformu a poskytovat API pro přístup k nativním funkcionalitám [1]. Existují tři typy prostředí: webový prohlížeč zaobalující webové (progresivní) aplikace, hybridní a nativní komponenty využívané hybridními aplikacemi, a nezávislé běhové prostředí (tzv. *self-contained runtime*), jež je základem interpretovaných aplikací.

[1] Vývojářské prostředí neboli IDE (*Integrated Development Environment*): Software pro vytváření aplikací, který kombinuje běžné vývojářské nástroje do jednoho uživatelského rozhraní (GUI). Typicky se skládá z editoru kódu, nástroje pro sestavování a ladění. [32]



Obrázek 3.1: Kategorizace multiplatformního vývoje aplikací. Adaptováno z [12].

Mobilní webové aplikace vyvinuté v HTML, CSS a jazyce javascript, běží ve webovém prohlížeči a nemohou být v zařízení nainstalovány. Dnešní prohlížeče umožňují přístup k některým základním funkcionalitám zařízení jako je lokace, kamera, mikrofon nebo úložiště. K webovým aplikacím může být sice přistoupeno pomocí URL, ale nemohou být distribuovány v GooglePlay či AppStore, což je značné omezení pro komerční aplikace [12].

3.1 Progresivní webové aplikace

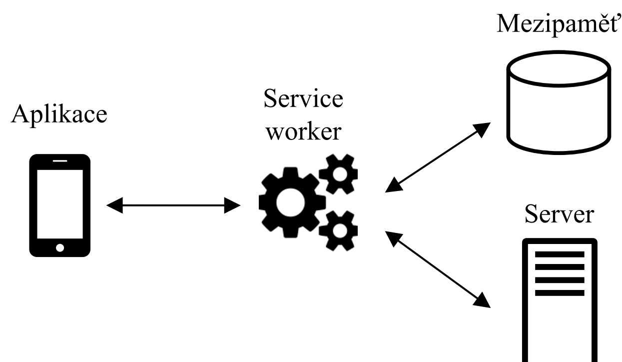
Z hlediska uživatelské přívětivosti vyvinula společnost Google progresivní webové aplikace (PWA), které vylepšují webové aplikace a využívají funkcí webového prohlížeče s cílem přiblížit chování aplikace co nejvíce nativnímu chování. Hlavní výhodou je znatelně jednodušší vývoj než u nativních mobilních aplikací, což implikuje i nižší náklady vývoje a údržby aplikace.

Mezi typické nativní vlastnosti, které progresivní aplikace umožňují oproti běžným webovým aplikacím, patří [43]:

- **Instalace** – uživatelé mohou aplikace rychle a jednoduše nainstalovat na plochu.
- **Cache** – aplikace si udržuje UI v mezipaměti, proto jsou PWA mnohem svižnější než běžné webové aplikace.
- **Úlohy na pozadí** – aplikace může díky tzv. *service workers* zpracovávat úlohy na pozadí.
- **Notifikace** – PWA mohou přijímat upozorňující notifikace nezávisle na tom, zda je aplikace právě spuštěna.

PWA jsou postaveny na několika technologiích, bez kterých by nebylo možné transformovat webovou aplikaci na progresivní. Jedná se o metadata aplikace (tzv. soubor *manifest*), která umožňují instalaci aplikace a *service workers* (SW), kteří zajišťují další výše zmíněné vlastnosti. Jak lze vidět na obrázku 3.2, service worker může fungovat jako prostředník (tzv. *proxy*) mezi webovým prohlížečem a serverem. Pokud je SW aktivní a má

zařízení internetové připojení, je požadavek zaslán serveru a stránka se při získání odpovědi uloží do mezipaměti (*cache*). V případě, že se jedná o offline požadavek, je stránka načtena z mezipaměti.



Obrázek 3.2: Schéma fungování progresivních webových aplikací.

Uživatelé dnešních webových prohlížečů jsou schopni instalovat webové aplikace na plochu, nicméně pokud se nejedná o progresivní webovou aplikaci (postrádá soubor manifest a service workers), je vytvořena pouze ikona, která funguje jako zástupce otevírající výchozí prohlížeč s danou webovou stránkou.

Soubor manifest je dokument ve formátu JSON a je složen z parametrů pro spuštění aplikace [42]. Tento typ webových aplikací může být publikován do obchodů Google Play[2] i Microsoft Store[3], které umožní progresivní webovou aplikaci zapouzdřit do nativní mobilní aplikace. Na App Store[4] nicméně podpora doposud nevznikla.

Zmíněný soubor manifest je jeden ze zdrojů, který je klientovi poskytnut, pokud se na něj nachází reference v hlavičce webové stránky, jak je vyobrazeno níže.

```
1  <head>
2      <link rel="manifest" href="manifest.json">
3  </head>
```

Výpis 3.1: Reference na soubor manifest v hlavičce HTML.

Organizace IANA[5] pro tento webový prostředek registrovala standardizovaný typ média *application/manifest+json* a soubor manifest tak může existovat s příponou *.manifest* nebo *.json*. Jeho typická struktura definuje ve své podstatě metadata aplikace.

Jak uvádí [43], service worker je část javascript kódu, který webová aplikace instaluje do prohlížeče při určitých podmínkách a je spouštěna na základě událostí v prohlížeči. Aby mohl service worker běžet, nemusí být otevřena webová aplikace, ale je nutný spuštěný prohlížeč. Na většině mobilních zařízeních prohlížeč běží neustále, tudíž service worker není omezen. U desktopových systémů stejně tak některé prohlížeče běží v pozadí a pokud ne, mohou tak být konfigurovány. Podmínkami pro instalaci služby service worker prohlížečem jsou:

- Prohlížeč musí podporovat service workers.

[2]Google Play: https://play.google.com/store/apps/
[3]Microsoft Store: https://apps.microsoft.com/store/apps/
[4]App Store: https://www.apple.com/cz/app-store/
[5]IANA (Internet Assigned Numbers Authority): Celosvětová organizace, která spravuje kořenové zóny DNS, přidělování IP adres a dalších prostředků internetových protokolů [15].

- Prohlížeč musí načíst webovou aplikaci pomocí TLS (HTTPS) spojení nebo z adresy `localhost`.
- Zdrojový kód služby service worker musí být načten ze stejného serveru jako byla načtena samotná webová aplikace.

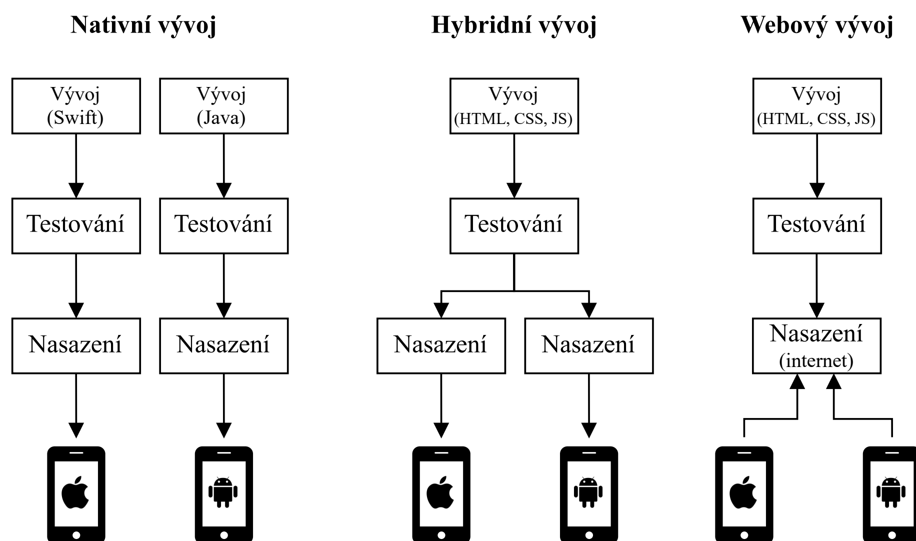
Progresivní webové aplikace mají sice přístup k funkcionalitám platformy nad rámec webových aplikací, nicméně jsou stále limitovány na ty, které jsou poskytovány webovým prohlížečem, v němž aplikace běží [6]. V následující sekci je vysvětlen přístup hybridních aplikací, který již není tímto omezením ovlivněn a přináší několik dalších vylepšení.

3.2 Hybridní aplikace

Hlavní výhodou hybridních aplikací je sdílený kód zajišťující kompatibilitu napříč různými operačními systémy kombinací webových technologií a nativních funkcionalit [8]. Běžným příkladem nativních funkcí je přístup k lokálnímu úložišti, notifikacím, galerii a také hardware zařízení jako je kamera, připojení k bluetooth, snímače přiblížení, nebo také využití biometrické autentizace jako je *Touch ID* nebo *Face ID* u iOS. I přes použití webových technologií je tento typ aplikací schopen fungovat offline a ukládat data v lokální databázi jako je například *SQLite* [8]. Hybridní přístup zapouzdřuje soubory HTML, CSS a javascript do instalovatelné a publikovatelné aplikace, která je využitím komponenty *WebView* vykresluje [6]. Mezi známé aplikační rámce patří Apache Cordova a Ionic, které usnadňují tvorbu hybridních aplikací včetně jejich inicializace, nastavení komponenty WebView a komunikačních protokolů mezi nativním kódem a WebView. Aby dokázal kód napsaný v jazyce javascript komunikovat s nativním kódem aplikace a přistoupit tak k nativním funkcím, je využíváno techniky zvané *bridging* nebo také FFI (*Foreign Function Interfaces*) [5]. WebView je nativní komponenta sloužící k vykreslování webového obsahu uvnitř okna nativní aplikace. Ve své podstatě jde o programovatelné rozhraní zaobalující vestavěný webový prohlížeč systému [44]. V hybridních aplikacích může být komponenta WebView vykreslena společně s dalšími nativními komponentami.

Obrázek 3.3 ukazuje, jak se může lišit vývoj pro tři různé platformy použitím nativního vývoje, hybridního přístupu a webového vývoje. Jak lze vidět, u nativního vývoje jsou rozděleny procesy vývoje, testování i sestavení do separátních větví pro jednotlivé platformy, jelikož je nutné vyvíjet v konkrétních jazycích (například Swift nebo Objective-C pro iOS a Java pro Android). Tento způsob vývoje je poměrně drahý a náročný, avšak umožňuje aplikaci přístup ke všem nativním funkcionalitám zařízení, lepší výkon, bezpečnost a také snadnější vyhovění požadavků pro publikaci na App Store či Google Play.

Hybridní přístup sjednocuje běžné fáze vývoje i testování do jedné větve, což razantně redukuje náklady na vývoj tohoto typu aplikací. Namísto vývoje ve třech různých jazycích jsou u hybridních aplikací použity webové technologie, které jsou podporovány na všech platformách. Možná nevýhoda tohoto přístupu je uživatelské rozhraní tvořené pomocí HTML, které se může chovat odlišně od nativního, i přes to, že je nastylováno dle designových směrnic platformy [6].

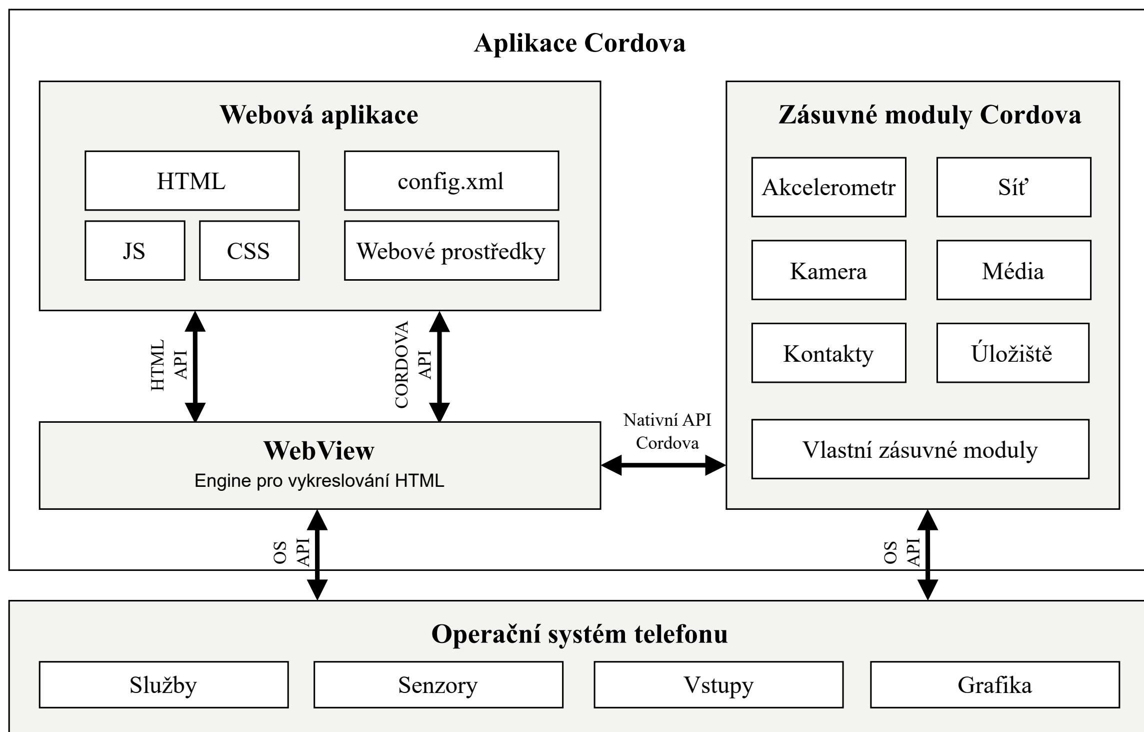


Obrázek 3.3: Porovnání vývoje mobilních aplikací. Adaptováno z: [2]

V následující části práce jsou popsány existující rámce pro tvorbu hybridních aplikací fungujících na principu vykreslování pomocí jazyka HTML a javascript v komponentě WebView. V navazující sekci 3.3 jsou rozebrány interpretované aplikace, které využívají při běhu programu nezávislou komponentu poskytující API nativních funkcionalit. Jsou zde popsány i aplikační rámce, které pod tento typ spadají. Sekce 3.4 analyzuje kompilované aplikace, jež jsou kompilovány na kód cílené platformy.

3.2.1 Apache Cordova

Tato část se zaměřuje a rozebírá pojem *Apache Cordova* (dříve *PhoneGap*) dle [44]. *Apache Cordova* je veřejně dostupný framework pro tvorbu mobilních aplikací, který umožňuje vývoj multiplatformních aplikací pomocí standardních webových technologií jako jsou HTML, CSS a javascript [39]. Webová aplikace je během procesu zapouzdřena do nativního kontejneru pro konkrétní cílovou platformu a je sestavena z několika komponent, jak lze vidět na obrázku 3.4. Uživatelské rozhraní nativní aplikace je tvořeno jednou obrazovkou, která obsahuje pouze komponentu *WebView* zaujímající veškerý prostor na obrazovce. Při spuštění aplikace je typicky načtena výchozí stránka `index.html` do WebView, která dále řídí interakci uživatele s webovou aplikací. Aplikace se chová stejně jako ve webovém prohlížeči a je schopna načítat webové prostředky, které byly zapouzdřeny společně s ní nebo je může získávat ze vzdáleného serveru. O logiku aplikace se stará javascript.



Obrázek 3.4: Architektura Apache Cordova. Adaptováno z: [39].

Jak lze vidět na obrázku 3.4, webová aplikace se skládá ze zdrojových souborů včetně obrázků a dalších webových prostředků nutných pro korektní běh aplikace. Je zde podstatný soubor `config.xml`, který specifikuje způsob fungování aplikace, například, zda má reagovat na změnu orientace (otočení) zařízení.

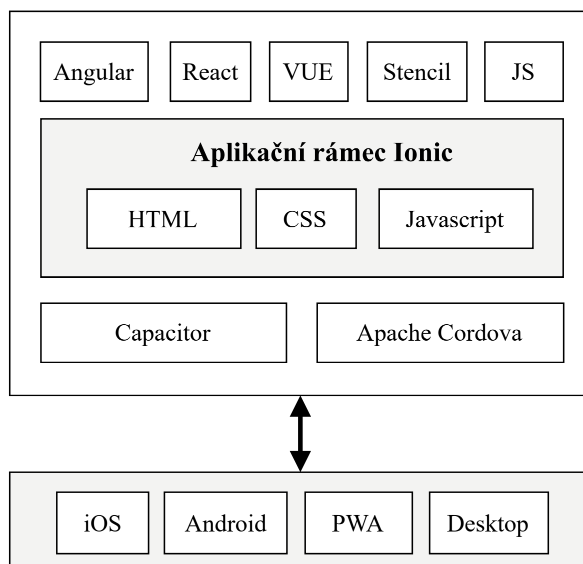
Zásuvné moduly jsou klíčovou částí aplikace Apache Cordova a jsou definovány v oficiální dokumentaci [40] jako balíčky kódu, které umožňují WebView komunikovat s nativní platformou, na které běží. Tyto moduly vytvářejí rozhraní a vazby mezi aplikací a nativními komponentami tak, aby mohly vzájemně komunikovat. Webová aplikace je následně schopna volat nativní funkce jazykem javascript. Do oficiálně poskytovaných balíčku zásuvných modulů od Apache Cordova spadají běžné nativní funkce jako je kamera, úložiště a další ilustrované na obrázku 3.4. Dodatečně mohou být importovány moduly třetích stran zpřístupňující rozhraní dalších funkcionalit, které nemusí být nutně dostupné na všech platformách. Může se jednat například o skenery čárových kódů nebo NFC komunikaci. Alternativně je vývojářům umožněno vytvářet vlastní moduly.

3.2.2 Ionic

Ionic je framework uživatelského rozhraní vytvořený pomocí HTML, CSS a javascript pro vývoj hybridních aplikací. V dřívějších verzích kombinoval několik technologií, jak uvádí [11], na jejímž vrcholu byl samotný framework Ionic poskytující uživatelské rozhraní aplikace. Součástí byla i knihovna pro tvorbu webových aplikací Angular a framework Apache Cordova, jenž byl rozebrán v předchozí podkapitole 3.2.1. Hlavním cílem rámce Ionic je poskytnout komponenty uživatelského rozhraní, které nejsou běžně dostupné při webovém vývoji. Jde například o navigační prvek *Tab bar* dostupný na většině mobilních zařízení. Ten je rámcem vytvořen pomocí webových technologií, chová se a vypadá stejně jako

nativní. Ionic dále poskytuje CLI (*Command Line Interface*) pro snazší tvorbu, sestavení a nasazení aplikací.

Jak uvádí spoluzakladatel frameworku [19], postupem času začal Ionic pracovat na nové knihovně Capacitor[6], která vznikla jako náhrada za framework Apache Cordova, který je stále možné využít, jak lze vidět na schématu technologií 3.5. Dále začaly být podporovány i jiné knihovny než Angular jako jsou React, Vue, Stencil a byla vytvořena podpora pro další JS knihovny ve formě rozhraní v jazyce javascript.



Obrázek 3.5: Technologie rámce Ionic. Obrázek adaptovaný z [17].

3.3 Interpretované aplikace

Interpretovaný přístup vývoje mobilních aplikací spočívá ve využití nezávislé komponenty v prostředí běhu programu, která je implementována pro každou cílenou platformu a poskytuje vývojářům aplikace API pro přístup k nativním funkcionalitám. U interpretovaných aplikací není použita komponenta WebView, ale podobně jako u hybridních aplikací, framework vývojářům typicky poskytuje rozšířitelný můstek (tzv. *bridge*), který umožňuje volat funkce nativního kódu [5] a je následně nasazen jako nativní balíček společně s enginem frameworku [12]. Typickým příkladem aplikačního rámce fungujícím na tomto principu je React Native napsaný v jazyce javascript.

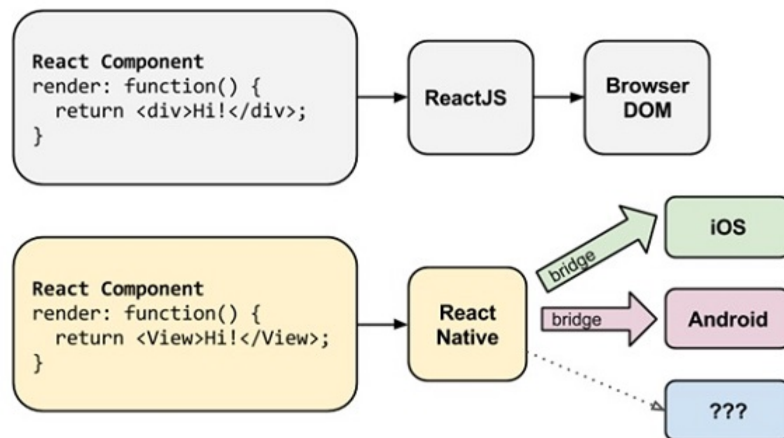
3.3.1 React Native

React Native, popsaný v knize [9], je veřejně dostupný javascript framework pro tvorbu mobilních aplikací založený společností Meta. Je postaven na knihovně React, která slouží k tvorbě uživatelského rozhraní. Stejně jako u webových aplikací vyvinutých knihovnou React jsou aplikace React Native vyvíjeny pomocí jazyka javascript a jeho rozšířenou syntaxí JSX (JavaScript XML) sloužící k popsání grafických komponent.

React používá programovací koncept VDOM (Virtual DOM), který uchovává reprezentaci uživatelského rozhraní v paměti a udržuje jeho stav synchronizovaný s reálným mode-

[6] Capacitor: https://capacitorjs.com/

lem DOM (*Document Object Model*) [21]. Tento koncept slouží jako vrstva mezi popisem, jak má UI vypadat a jaké úkony jsou nutné k vykreslení aplikace. Jak lze vidět na obrázku 3.6, tvoří se tak abstrakce mezi kódem vývojáře a samotným vykreslováním, čímž je možné použít pro vykreslení nativní API v Objective-C pro iOS nebo Java API pro Android namísto prohlížeče. Aby nebyla vykreslována celá stránka znovu, React vypočítává pouze nutné změny pomocí reprezentace DOM v paměti.



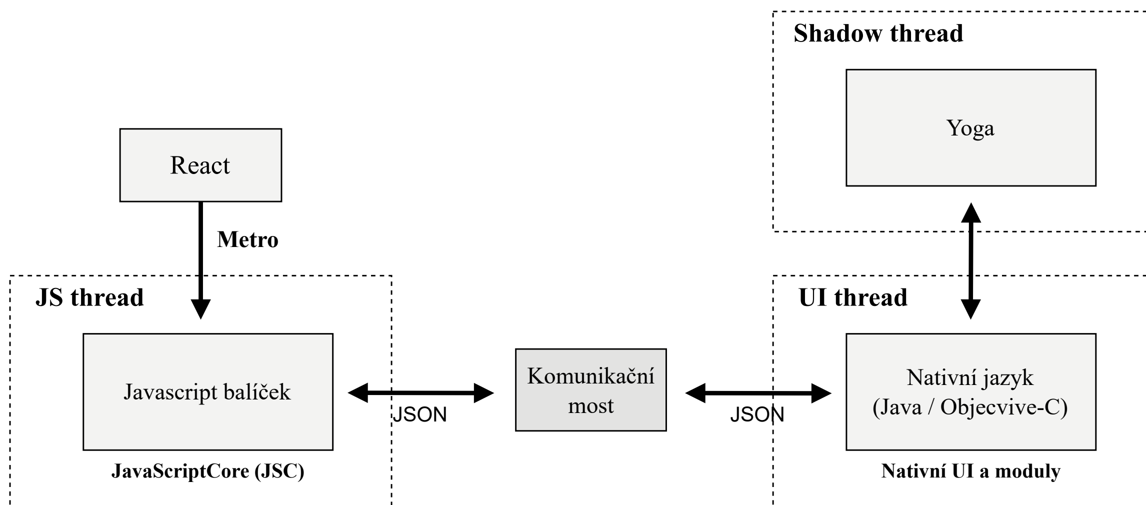
Obrázek 3.6: Způsob vykreslování v React Native. Převzato z: [9]

Obrázek 3.7 znázorňuje architekturu popsanou v článku [14], kterou tvoří tři paralelně běžící vlákna React Native aplikace.

- **JS thread** – probíhá zde většina aplikační logiky. Většina javascript kódu je zde čtena a kompilována. Technologie Metro zajišťuje překlad alternativních javascript syntaxí (TypeScript nebo JSX) a sjednocuje je do jednoho zdrojového souboru. Ten je dále spouštěn ve výchozím prostředí Hermes, alternativně v JavaScriptCore (JSC) nebo v prostředí V8 při ladění aplikace v prohlížeči Chrome [22].

- **Native thread** – dochází zde ke spouštění nativního kódu. Komunikace s vláknem JS je prováděna při změnách uživatelského rozhraní nebo při přístupu k nativním funkcím.

- **Shadow thread** – React Native zde používá vlastní technologii Yoga, která slouží pro výpočet CSS flexbox rozložení a k předání výsledku nativnímu UI.



Obrázek 3.7: Architektura React Native. Adaptováno z: [14]

Kniha [9] také uvádí, že React Native tvoří komunikační most (tzv. *bridge*) v jazyce Objective-C pro iOS a Java pro Android s nativním API pro vykreslování a pro přístup k nativním funkcím. Aplikace je díky tomu vykreslena nativními komponentami a ne ve WebView, proto vypadá stejně jako běžná nativní aplikace.

Komunikační most je tvořen asynchronní frontou, kterou prochází serializovaná data ve formátu JSON a při doručení jsou dekódována [14]. Taková komunikace mohla zapříčiňovat zpoždění a různé výkonnostní problémy, proto byla právě v tomto roce 2022 zveřejněna nová architektura React Native nahrazující komunikační most turbo moduly (tzv. *Turbo Modules*) umožňujícími synchronní komunikaci s JS [34].

3.4 Kompilované aplikace

Kompilovaný přístup aplikací spadá pod generativní přístup vývoje multiplatformních aplikací a je založený na kompilaci zdrojového kódu aplikace na kód cílené platformy (tzv. *byte code*) bez využití můstku jako u interpretovaných aplikací [1]. Přístup k nativním funkcím není tedy zajišťován žádnou vrstvou jako je můstek, ale namísto toho je poskytnut vývojáři aplikace přes SDK (Software Development Kit) aplikačního rámce, který mapuje funkcionalitu přímo na SDK dané platformy [5]. Další výhodou kompilovaných aplikací je použití nativních komponent uživatelského rozhraní. Mezi představitele lze zařadit aplikační rámec Flutter, který je kompilován z jazyka Dart přímo do nativního jazyka, a jak uvádí [36], [1] a [5], do této kategorie lze též zahrnout Xamarin a jeho nového nástupce .NET MAUI právě kvůli nezávislosti na interpretru, nicméně existují studie, které zařazují Xamarin do interpretovaného přístupu jako například [6].

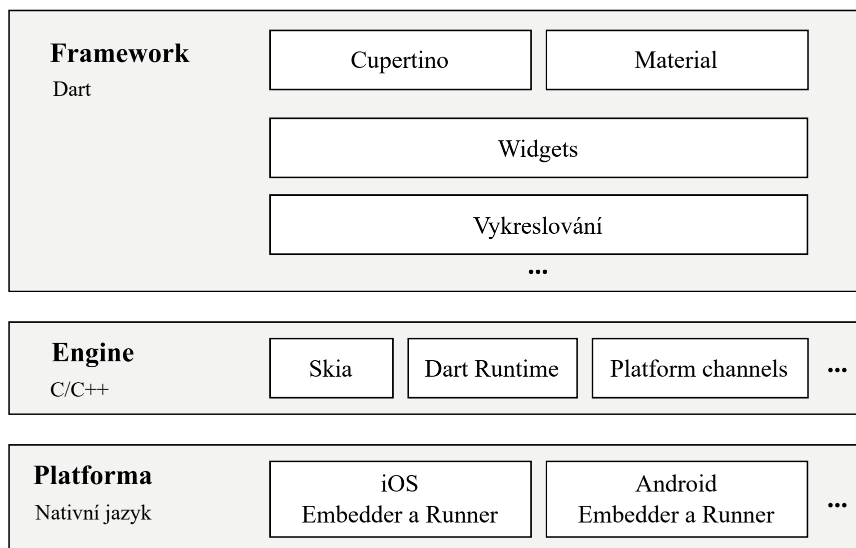
3.4.1 Flutter

V této podkapitole je čerpáno z knihy [4] a je zde rozebráno fungování rámce Flutter, jeho způsob kompilace, vykreslování a srovnání vůči ostatním populárním rámcům.

Mezi výhody rámce Flutter patří veřejná dostupnost, plná kontrola nad uživatelským rozhraním a podpora společností Google. Jedním z jeho hlavních cílů při vzniku bylo, stát se vysoce výkonnou alternativou ostatních existujících rámců sloužících k vývoji multiplat-

formních aplikací, proto se od nich způsob vykreslování zásadně liší. Vykreslování bylo vyvinuto s důrazem na vysokou snímkovou frekvenci a z toho důvodu byl použit grafický engine Skia[7], který se stará o vykreslování uživatelského rozhraní aplikace. Není díky tomu nutná tvorba dalšího rozhraní přistupujícího k API operačního systému pro vykreslování nativních UI komponent (tzv. *Original Equipment Manufacturer (OEM) widgets*). Nedochází ke ztrátě výkonu, ani k limitacím designu aplikace. Nevyužitím nativních komponent by mohlo dojít k porušení designu dané platformy, nicméně Flutter poskytuje balíčky poskytující nativní komponenty. Na platformě Android jde o Material[8] a na iOS o Cupertino[9]. Programovacím jazykem se stal Dart, který umožňuje deklarativní UI a poskytuje AOT (*ahead-of-time*) i JIT (*just-in-time*) kompilaci. Flutter používá AOT ke kompilaci release verzí aplikace a JIT během vývoje a ladění.

Architekturu Flutter tvoří tři vrstvy, sestupně – Framework, Engine a Platform [10]. Nejnižší vrstvu na úrovni platformy tvoří tzv. *Embedder* a *Runner*, které jsou specifické pro každou platformu. Poskytují přístup k nativnímu API, nativním službám a umožňují hostování aplikace v nativním prostředí včetně JIT kompilace. Druhá vrstva (Engine) je integrována s nižší vrstvou dané platformy. Poskytuje také například Dart runtime, Skia a Plaform Channels[10]. Probíhá zde rasterizace scény, která je sestavena ze stromu komponent v nadřazené vrstvě. Nejvyšší vrstva reprezentuje samotný Flutter framework, který spravuje tvorbu uživatelského rozhraní při vývoji aplikací. Popsané vrstvy jsou znázorněny na obrázku 3.8.



Obrázek 3.8: Schéma Flutter. Adaptováno z: [10]

3.4.2 Xamarin

Xamarin je otevřená platforma pro vývoj aplikací pro Android, iOS a Windows pomocí technologie .NET. Je popsána v oficiální dokumentaci [26]. Tvoří abstrakční vrstvu, která zajišťuje komunikaci sdíleného kódu a kódu cílené platformy.

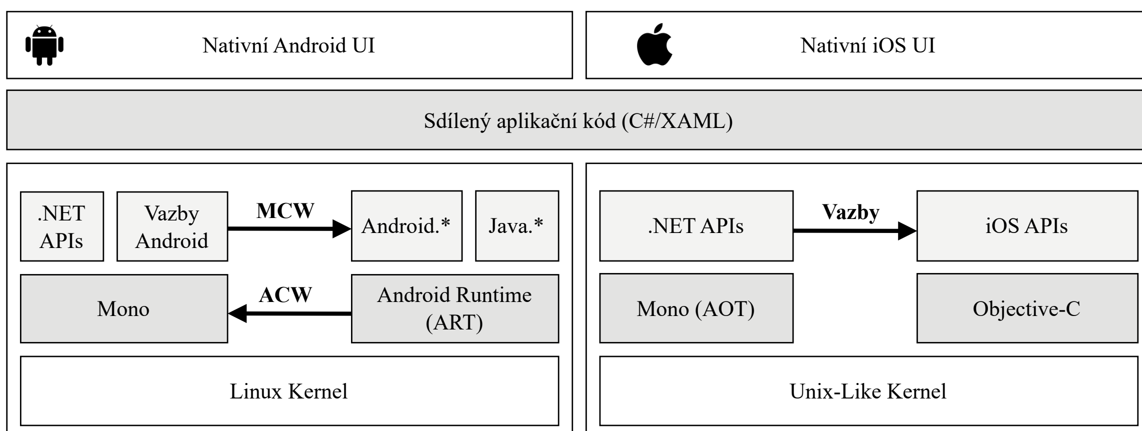
[7]Skia engine: https://skia.org/
[8]Material Design: https://docs.flutter.dev/development/ui/widgets/material
[9]Cupertino: https://docs.flutter.dev/development/ui/widgets/cupertino
[10]Plaform Channels: Kanály umožňují komunikaci mezi klientem (aplikací Flutter) a kódem hostující platformy.

Aplikace Xamarin Android jsou kompilovány z jazyka C# do jazyka IL (*Intermediate Language*), který je poté při spuštění aplikace kompilován způsobem JIT do nativního jazyka symbolických adres (*assembly language*) [24]. Běží uvnitř prostředí *Mono runtime* (napsaném v jazyce C) společně s ART (*Android Runtime*). Obě tyto prostředí běží nad linuxovým jádrem a poskytují vývojářům rozhraní pro přístup k systému.

Jak znázorňuje obrázek 3.9 a jak je popsáno v [23], aby byla možná komunikace ART s prostředím Mono (tedy se spravovaným kódem), je nutné rozhraní ACW (Android Callable Wrappers), jelikož není možná typová registrace za běhu programu pomocí ART. ACW je most JNI[11] (Java Native Interface) umožňující ART volat virtuální metody nebo metody rozhraní, které jsou implementovány ve spravovaném kódu.

Komunikace směrem ze spravovaného kódu do kódu Android je řešena pomocí rozhraní MCW (Managed Callable Wrappers), které je zodpovědné za převod spravovaných typů a Android typů a volání metod platformy Android pomocí nativního rozhraní JNI [24]. V praxi lze díky MCW používat ve zdrojovém kódu třídy z jmenného prostoru (tzv. *namespace*) Android.

Aplikace Xamarin iOS jsou kompletně kompilovány pomocí AOT z jazyka C# do nativního jazyka symbolických adres [26]. K vzájemné komunikaci mezi spravovaným kódem C# a Objective-C slouží vazby (tzv. Bindings), jak lze vidět na obrázku 3.9.



Obrázek 3.9: Architektura Xamarin u platforem Android a iOS. Adaptováno z [26].

K přístupu nativních funkcí slouží knihovna Xamarin Essentials, která tvoří jeho abstrakci a zjednodušuje tak celkový proces [26].

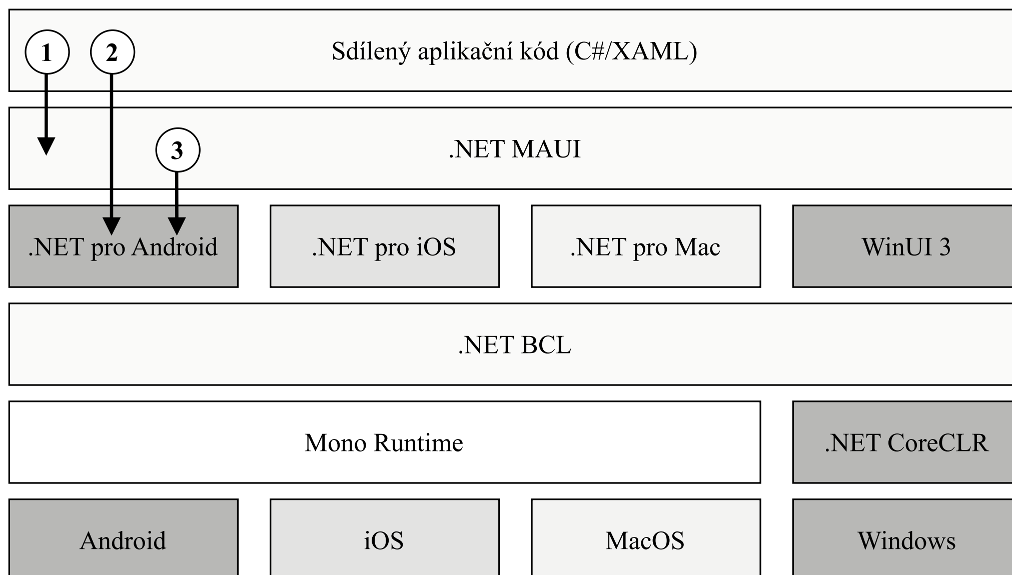
3.4.3 .NET MAUI

.NET MAUI (Multi-platform App UI) je nově vzniklá technologie sloužící ke multiplatformnímu vývoji aplikací a zároveň je nástupcem velmi známého využívaného frameworku Xamarin Forms[12]. Ten je rozšířen o vývoj desktopových aplikací, vyšší výkon a rozšiřitelnost UI komponent (tzv. *controls*).

[11] JNI: Nativní programovací rozhraní umožňující vzájemnou komunikaci mezi kódem napsaném v jazyce Java běžícím ve virtuálním stroji Java JVM (Java Virtual Machine) a aplikacemi napsanými v jiných programovacích jazycích [30].

[12] Xamarin Forms: https://dotnet.microsoft.com/en-us/apps/xamarin/xamarin-forms

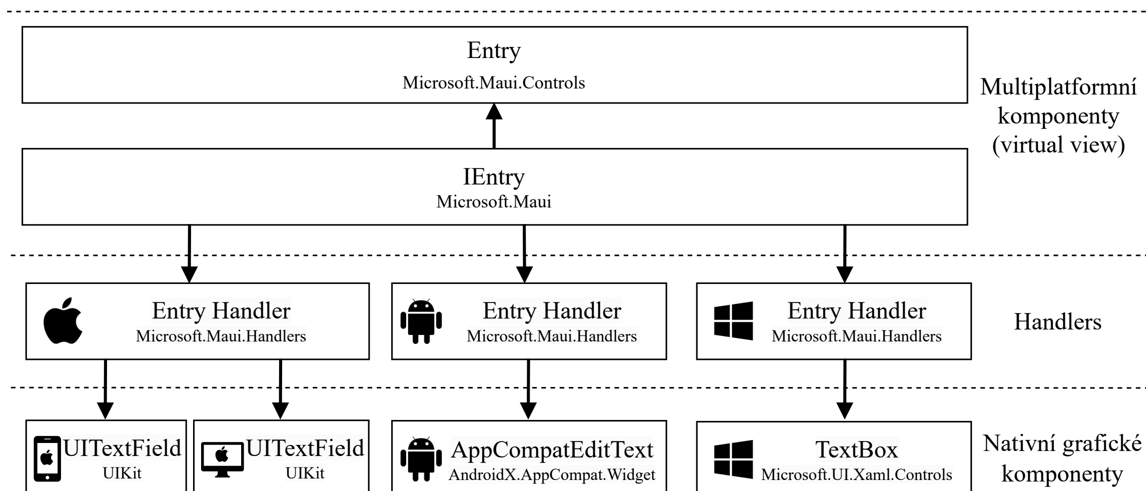
MAUI sjednocuje systémové rozhraní všech podporovaných platforem do jednoho API a umožňuje tak používat jeden sdílený kód [29]. Zároveň jsou poskytnuty knihovny pro specifické platformy, ke kterým může přistupovat aplikační kód přímo nebo přes sdílené API .NET MAUI, jak lze vidět u vyznačení mezi první, druhou a třetí vrstvou na schématu 3.10.



Obrázek 3.10: Architektura .NET MAUI. Adaptováno z: [27]

Všechny knihovny specifické pro konkrétní platformu sdílí stejnou knihovnu .NET BCL (*Base Class Library*), která abstrahuje přístup k daným platformám a odlučuje ho tak od kódu aplikace [29]. Knihovna BCL je závislá na běhovém prostředí .NET, kterým zajišťuje prostředí pro spustitelnost kódu (tzv. *execution environment*). U platforem Android, iOS a macOS se jedná o běhové prostředí Mono runtime, u Windows je využito prostředí .NET CoreCLR. Aplikace Android a iOS jsou kompilovány stejně jako u frameworku Xamarin, macOS využívá technologii Mac Catalyst od společnosti Apple a aplikace Windows používají knihovnu WinUI 3 pro tvorbu nativních komponent.

Podstatnou vlastností MAUI je rozšiřitelnost komponent a možnost tvorby nových funkcionalit. Existují proto tzv. *handlers*, které upravují vzhled a chování multiplatformních komponent pomocí jejich rozhraní, jak lze vidět na schématu 3.11. Logika musí být implementována pro každou platformu separátně, nicméně jde stále o kontext jazyka C#.



Obrázek 3.11: Schéma rozšiřitelnosti multiplatformních MAUI komponent.

V praxi se používá metoda `AddHandler`, mezi jejíž parametry se uvede typ (resp. rozhraní) komponenty definovaný v .NET MAUI frameworku a typ příslušného handleru. Příklad kódu je znázorněn v ukázce 3.2.

```
1 .ConfigureMauiHandlers(handlers =>
2 {
3     handlers.AddHandler(typeof(MapHandlerDemo.Maps.Map),typeof(MapHandler));
4 })
```

Výpis 3.2: Konfigurace chování multiplatformní komponenty v .NET MAUI.

Postup vytvoření vlastních multiplatformních komponent pomocí *handlerů* je dle dokumentace MAUI následovný [27]:

1. Vytvoření třídy pro multiplatformní komponentu dědící z komponenty *View*, která reprezentuje vizuální element. Komponenta by měla rovněž poskytovat veřejné rozhraní dostupné pro handler komponenty.

2. Vytvoření částečné (tzv. *partial*) třídy pro handler, který bude specifický pro každou z podporovaných platforem a bude určovat typ nativní vizuální komponenty.

3. Vydefinování akcí, které se mají vyvolat při změně vlastností multiplatformní komponenty a jejich následné přiřazení pomocí tzv. *Property Mapper*.

4. Tvorba částečné třídy handleru pro každou platformu separátně současně s tvorbou nativní vizuální komponenty implementující multiplatformní komponentu.

Stejně jako Xamarin Essentials u předchůdce Xamarin Forms, existuje sada rozšiřujících komponent zvaná MAUI Community Toolkit[13]. Jelikož se jedná o novou technologii, neobsahuje zatím mnoho komponent, nicméně jde o rozšiřující se open-source projekt[14].

[13]MAUI Community Toolkit: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/
[14]Community Toolkit projekt: https://github.com/CommunityToolkit/Maui

Kapitola 4

Analýza

Společnost Riganti se zabývá vývojem softwarových řešení na míru, a to zejména informačních systémů v technologiích od společnosti Microsoft, jako je .NET. Rovněž je autorem webového frameworku DotVVM, který běžně používá k vývoji komerčních aplikací pro svoje zákazníky a zároveň má sestaven tým spravující tuto technologii. V průběhu několikaletého používání frameworku v produkčním prostředí pravidelně vyplývají požadavky nových funkcionalit. Ty jsou postupně implementovány souběžně se zákaznickými projekty. Framework je zpřístupněn vývojářům a je veřejně k dispozici jako open source projekt na platformě GitHub, kde je průběžně aktualizován a rozvíjen o nové komponenty a funkcionalitu.

Riganti, a určitě i další vývojářské firmy, zabývající se tvorbou informačních systémů, se už někdy ocitli v situaci, kdy zákazník požadoval webovou aplikaci vycházející ze stávající desktopové aplikace. V takové situaci by ušetřila spoustu času tvorba webové aplikace, která by se využila v hybridním režimu jako desktopová, a případně i mobilní aplikace. Zároveň by byly zachovány složité komponenty uživatelského rozhraní, které by tímto způsobem nebylo nutné znovu implementovat pro separátní platformy.

Tato práce se zabývá poskytnutím rozšíření DotVVM takovým způsobem, aby mohla webová aplikace vyvinutá v DotVVM fungovat multiplatformně. K tomu může být využit multiplatformní framework, v němž bude webová aplikace hostována hybridním způsobem. Toho lze docílit vlastní modifikací komponenty WebView, která je typicky součástí multiplatformních frameworků, a následnou tvorbou komunikačního kanálu mezi webovým frameworkem, multiplatformním frameworkem a WebView. Aplikační rámec .NET MAUI se jeví jako vhodný kandidát pro integraci s DotVVM zejména kvůli platformě .NET, na které jsou oba frameworky postaveny a také použitím stejného návrhového vzoru MVVM.

Mezi existující řešení postavené na stejném principu se staví integrace technologií Blazor[1] a .NET MAUI od společnosti Microsoft, jehož zdrojový open-source kód byl v rámci analýzy prozkoumán a vychází z něj značná část navrhnutého řešení. Jedná se primárně o MAUI handler (sekce 3.4.3) a implementaci specifických komponent WebView pro dané platformy (popsaných v sekci 4.2). Řešení však postrádá dokumentaci a bylo nutné nahlížet pouze do zdrojových kódu obsažených v repozitáři[2], součástí této práce jsou tedy využité techniky objasněny a znázorněny. Co se týče možnosti integrace DotVVM a .NET MAUI, v článku [13] je nastíněn princip prototypu, z něhož část praktické práce také vychází. Jedná se pouze o důkaz konceptu, který demonstruje komunikaci na platformě Windows a funkčnost na mobilních zařízeních zde není řešena.

[1] Blazor: Framework pro tvorbu webových aplikací pomocí technologie .NET. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[2] Repozitář .NET MAUI: https://github.com/dotnet/maui/tree/main/src

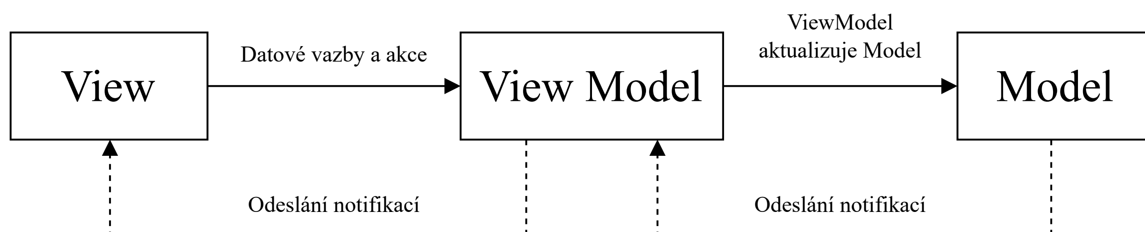
4.1 DotVVM

DotVVM je framework uživatelského rozhraní, který umožňuje tvorbu webových aplikací postavených na technologii ASP.NET (případně její varianty ASP.NET Core) pomocí návrhového vzoru MVVM. Přenos dat mezi serverem a klientem (prohlížečem) je zajištěn bez nutnosti tvorby a údržby REST API či mapování klientských a serverových dat. Tato kapitola představuje základní koncepty aplikačního rámce DotVVM, které jsou podrobněji popsány v dokumentaci [33].

Použití DotVVM je vhodné především u podnikových aplikací, jako jsou administrační stránky, informační systémy ERP[3] (*Enterprise Resource Planning*), CRM[4] (*Customer Relationship Management*) a zejména aplikace, které pracují s datovými tabulkami, formuláři s velkým množstvím polí, komplexními validacemi a logikou. Jelikož DotVVM dokáže běžet ve stejném procesu společně s jinými aplikačními rámci založenými na technologii ASP.NET (jako je například ASP.NET Web Forms), je možné využít DotVVM k postupné modernizaci zastaralých aplikací a následně je zmigrovat na nejnovější verzi .NET platformy. Během procesu přepisování aplikace je zachována její funkčnost, je tedy možné souběžně vyvíjet novou funkcionalitu.

4.1.1 Návrhový vzor MVVM

Návrhový vzor MVVM (*Model-View-ViewModel*) odděluje aplikační a prezentační logiku od uživatelského rozhraní. Jeho součástí jsou tři komponenty – Model, View a ViewModel. Jejich vztah lze vidět na diagramu 4.1.



Obrázek 4.1: Návrhový vzor MVVM a vztah mezi jeho komponentami. Adaptováno z: [37]

View je zodpovědný za definování struktury a vzhledu uživatelského rozhraní. U aplikací společnosti Microsoft jde typicky o formát XAML (Extensible Application Markup Language), ale může jít například u webových aplikacích i o HTML. Ve view je vyjádřena vazba na viewmodel a jeho vlastnosti a akce. Viewmodel implementuje vlastnosti a akce, ke kterým může být view navázán a zároveň ho informuje o změnách stavu dat. Model slouží k reprezentaci dat a neobsahuje žádnou aplikační logiku.

4.1.2 Architektura klienta a serveru

DotVVM pokrývá jak serverovou, tak klientskou část, a proto není nutná tvorba klientské části v jazyce javascript, ani REST API. Díky tomu je možné vytvářet komplexní webové

[3] Plánování podnikových zdrojů (ERP): „Typ softwarového systému, který organizacím pomáhá optimalizovat výkon pomocí automatizace a řízení základních obchodních procesů.“ [25]

[4] CRM software: Program, který umožňuje přehledně shromažďovat potřebné údaje o zákaznících firmy – kontaktní informace, celkové obraty apod. [35]

aplikace s použitím čistě jazyka C# a jazyka DotHTML, jehož syntax je pouze rozšířením běžného HTML. Jak uvádí [41], DotVVM spravuje běžné úkony, které musí vývojáři řešit, jako je například komunikace se serverem a mapování dat mezi klientem a serverem. Klientská část frameworku je založena na JS knihovně Knockout, jež je DotVVM značně rozšířena. Stránky v DotVVM jsou tvořeny dvěmi částmi – *view* strukturované v jazyce DotHTML a běžná C# třída reprezentující *viewmodel*.

ViewModel

DotVVM viewmodel reprezentuje aktuální stav stránky a reaguje na akce (tzv. *commands*) uživatelů pomocí metod, jeho formát lze vidět na příkladu 4.1. Ve své podstatě existují dva jeho stavy – jeden na serverové straně a druhý na klientské straně, který je spravován čistě DotVVM. Oba se při vzájemné komunikaci serializují/deserializují do formátu JSON, přičemž při výchozím nastavení se zasílá stav celého viewmodelu.

```
1  public class SampleViewModel : DotvvmViewModelBase
2  {
3      public int Number { get; set; }
4
5      public int Result { get; set; }
6
7      public void CalculateCommand()
8      {
9          Result = Number * Number;
10     }
11 }
```

Výpis 4.1: Příklad DotVVM viewmodelu v jazyce C#

Velikost viewmodelu může být s rostoucí komplexností velká, a proto dokáže DotVVM volat metody na serveru i jiným způsobem – statickými příkazy (tzv. *static commands*). Ty umožňují odesílat na server pouze identifikátor a argumenty metody, nikoliv celý viewmodel. Jako odpověď je serverem zaslána návratová hodnota metody, která může být použita k aktualizaci viewmodelu na klientské straně. Stejně tak mohou být statické příkazy využity k jednoduchým operacím, které jsou přeložitelné do jazyka javascript a mohou být provedeny lokálně bez jakékoliv komunikace se serverem.

View a datové vazby

V DotVVM je view strukturován v jazyce DotHTML, který rozšiřuje standardní HTML syntax o datové vazby vlastností a akcí ve viewmodelu, komponenty (tzv. *controls*) a definice direktiv (tzv. *directives*). Existují čtyři typy možných datových vazeb v jazyce DotHTML:

- ***Value*** – vyhodnocení datové vazby hodnoty vlastnosti viewmodelu na klientovi.
- ***Command*** – vyvolání akce viewmodelu na serveru.
- ***StaticCommand*** – vyvolání akce viewmodelu na klientovi, pokud jde o přeložitelný výraz do jazyka javascript. V ostatních případech jde o vyvolání akce na serveru, přičemž není zasílán celý viewmodel.
- ***Resource*** – vyhodnocení datové vazby hodnoty vlastnosti viewmodelu na serveru.

DotVVM generuje z formátu DotHTML standardní HTML obsahující speciální výrazy z JS knihovny Knockout, které zajišťují datovou vazbu mezi view viewmodelem na straně klienta. Taková vazba je reprezentována atributem `data-bind` a reaguje na uživatelské změny nebo externí změny v datech dynamickou aktualizací uživatelského rozhraní. Mimo vazby poskytuje Knockout návrhový vzor MVVM. Reprezentace stavu stránky je uchovávaná ve viewmodelu, který je uložen v prohlížeči jako tzv. *Knockout observable object*. Ten funguje na principu návrhového vzoru pozorovatel (*Observer pattern*) a dokáže tak informovat související objekty o změnách stavu, typicky voláním jejich metod.

Definice direktiv nacházejících se na začátku zdrojového kódu komponent slouží k importu objektů, které se nacházejí v jiném namespace. Jedna z povinných definic je reference na viewmodel příslušící danému view a lze ji vidět na prvním řádku v ukázce 4.2.

```
1  @viewModel DotvvmDemo.SampleViewModel, DotvvmDemo
2  @js sample-module
3
4  <p>
5      Zadejte hodnotu: <dot:TextBox Type="Number" Text="{value: Number}" />
6  </p>
7  <p>
8      <dot:Button Text="Umocnit" Click="{command: CalculateCommand()}" />
9  </p>
10 <p>
11     Druhá mocnina zadané hodnoty je: {{value: Result}}
12 </p>
```

Výpis 4.2: Příklad pohledu v jazyce DotHTML vázaného k viewmodelu 4.1.

Komponenty

Pro znovupoužitelnost repetetivních a dynamických částí DotHTML slouží komponenty (tzv. *Markup Controls*), které se vytváří obdobně jako stránky, ale je možné je použít opakovaně na různých stránkách bez duplicit kódu. Na ukázce 4.3 je deklarována komponenta, jejíž název DotHTML tagu je složen z prefixu nastaveného v konfiguraci DotVVM, dvojtečkou, názvem komponenty a následným výčtem atributů s možnou vazbou na viewmodel. Atributy mohou být specifické pro danou komponentu i běžné HTML atributy.

```
1  <div>
2      <cc:Player Text="{value: Name}" Time="{value: ElapsedTime}" class="player" />
3  </div>
```

Výpis 4.3: Příklad použití DotVVM komponenty v DotHTML stránce. Registrace komponenty je uvedena na ukázce 4.4.

```
1  config.Markup.AddMarkupControl("cc", "Player", "Components/Player.dotcontrol");
```

Výpis 4.4: Příklad registrace DotVVM komponenty v konfiguračním souboru `DotvvmStartup.cs`. Registrační metoda vyžaduje jako parametry prefix, název a fyzickou cestu komponenty.

Ke každé komponentě i stránce může být přiřazen nejen vlastní viewmodel, ale i javascript modul. Tento modul umožní definovat javascript funkce, které mohou být díky direktivě uvedené v ukázce 4.1 na druhém řádku volány ze stránky DotVVM pomocí proměnné `_js` v datové vazbě typu StaticCommand. Registrace modulu probíhá opět v konfi-

guračním souboru DotvvmStartup.cs, jak je naznačeno v ukázce 4.5, avšak je nutné uvést jako typ webového prostředku ScriptModuleResource.

```
1 config.Resources.Register("sample-module",
2     new ScriptModuleResource(new UrlResourceLocation("~/app/sample-module.js"))
3 {
4     Dependencies = new [] { ... }
5 });
```

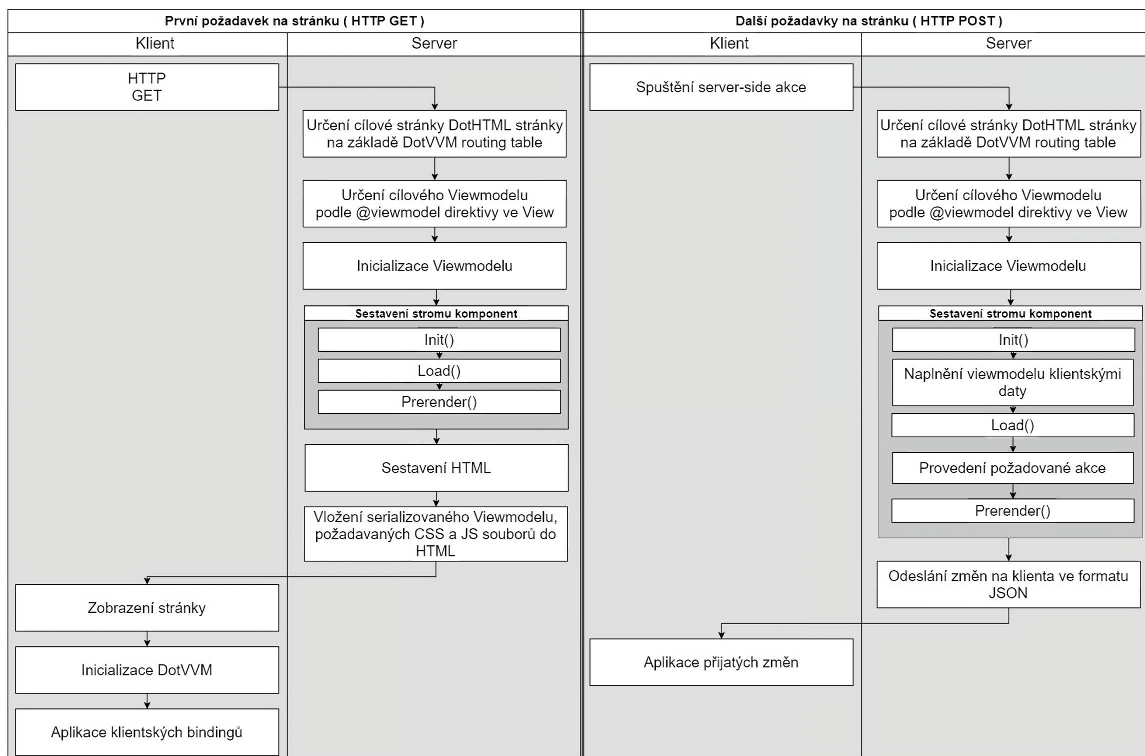
Výpis 4.5: Příklad registrace javascript modulu v konfiguračním souboru DotvvmStartup.cs. Registrační metoda vyžaduje jako parametry název a fyzickou cestu skriptu.

Životní cyklus HTTP požadavků

DotVVM rozlišuje 2 typy životního cyklu stránky dle metody HTTP požadavku [41]. Na obrázku 4.2 je znázorněna zjednodušená sekvence akcí vykonávaných na klientovi i na serveru během HTTP GET a POST požadavků.

Při iniciálním HTTP požadavku (při načtení stránky) je na serveru nejprve určena cílová stránka na základě DotVVM směrovací tabulky (tzv. *routing table*), následně je vytvořena instance viewmodelu a syntax DotHTML je přeložena do běžného HTML. Serverová část DotVVM serializuje C# viewmodel do formátu JSON, který zahrne v HTML stránce do skrytého elementu. Společně s ním jsou odeslány také skripty JS nutné pro fungování DotVVM a to během jednoho HTTP požadavku, jak lze vidět na obrázku 4.2.

Pokud je nutné při akci, jako je kliknutí tlačítka, zavolat serverovou metodu, DotVVM serializuje viewmodel do formátu JSON a odešle ho pomocí AJAX na server. Stejně jako u požadavku GET, se určí cílová stránka a její příslušný viewmodel. Serverová část DotVVM poté vytvoří instanci viewmodelu, který naplní obdrženými daty od klienta a zavolá požadovanou metodu. Z obrázku 4.2 lze vyčíst, že jsou po proběhnutí business logiky, vzniklé změny ve viewmodelu serializovány do JSON a zaslány zpět prohlížeči, v němž jsou aplikovány. Knockout zajistí aktualizaci uživatelského rozhraní a není tak nutná obnova celé stránky.



Obrázek 4.2: Životní cyklus HTTP požadavků v DotVVM. Převzato z: [41].

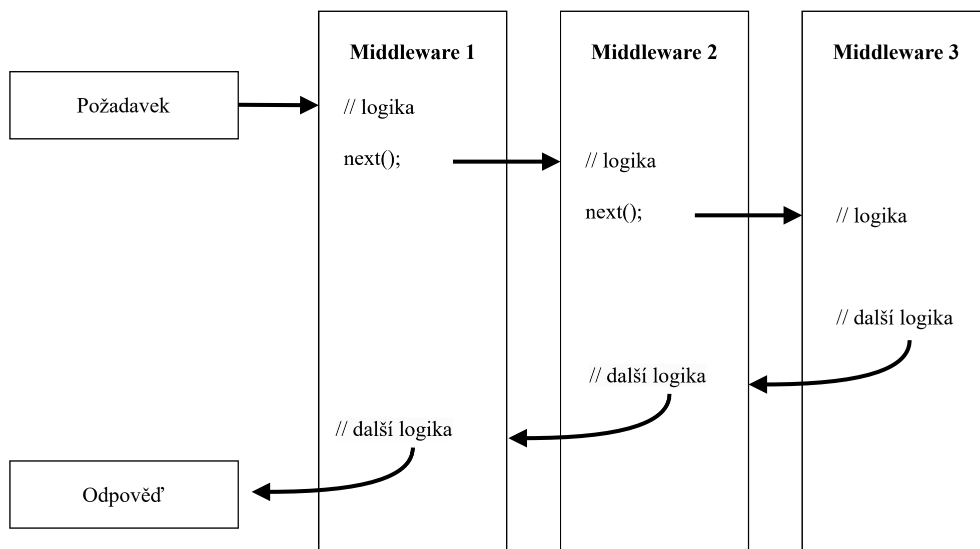
4.1.3 Konfigurace DotVVM

Nastavení a funkcionalita v DotVVM může být konfigurována a to typicky ve dvou souborech nacházejících se v kořenovém adresáři projektu:

- `Startup.cs` – jedná se o hlavní konfigurační soubor aplikací ASP.NET Core, kde se registrují služby DotVVM a tzv. *middleware*[5], který se zapojuje do procesu zpracování HTTP požadavků (tzv. *request pipeline*). Zřetězení middleware tvořící request pipeline je ukázáno na obrázku 4.3. Jeho součástí je rovněž `DotvvmMiddleware`, který slouží ke zpracování požadavku frameworkem, jež na stejném principu provolává vlastní middleware.

- `DotvvmStartup.cs` – v tomto souboru se vyskytuje konfigurace DotVVM zahrnující například směrování požadavků na dané stránky a registraci komponent.

[5]Middleware: Software zapojený do procesu zpracování HTTP požadavků a odpovědí, který může vykonat nějaké akce a předat požadavek další komponentě připojené v procesu [28].



Obrázek 4.3: Proces zpracování HTTP požadavků (tzv. *request pipeline*). Adaptováno z: [28].

Směrování požadavků

Každá stránka v DotVVM musí být explicitně registrována ve směrovací tabulce (tzv. *route table*) společně s náležitostmi jako jsou identifikační název cesty, URL včetně parametrů, cesta ke stránce a případně výchozí hodnoty parametrů. Slouží k určení požadované stránky z URL příchozího požadavku. Nastavení směrování je součástí konfiguračního souboru `DotvvmStartup.cs`, jehož ukázka je uvedena v příkladu 4.6.

```
1    config.RouteTable.Add("Page", "page/{id}", "Views/page.dothtml", new { id = 0 });
```

Výpis 4.6: Příklad registrace stránky do směrovací tabulky

Kompilace stránek

DotHTML stránky musí být kompilovány ještě před jejich prvním použitím a to za běhu aplikace, nikoliv při sestavení projektu. Je tomu tak kvůli potřebné konfiguraci, která je definována v kódu C#. Existují 3 možnosti určující moment překladu:

- ***Lazy mode*** – stránky jsou kompilovány při prvním uživatelském přístupu. Překlad je vhodný pro vývojářské prostředí a případné produkční prostředí s menším síťovým provozem.
- ***AfterApplicationStart mode*** – kompilace stránek začne v pozadí po spuštění aplikace. Během procesu může aplikace zpracovávat HTTP požadavky, nicméně je možné, že uživatel přistoupí na ještě nezkompilovanou stránku a ta je kompilována stejným způsobem jako u módu Lazy. Jedná se o výchozí způsob kompilace, jelikož je zachováno rychlé spuštění aplikace a je zvýšena šance, že uživatel nebude čekat na kompilaci stránky.
- ***DuringApplicationStart mode*** – ke kompilaci stránek dochází během spouštění aplikace, což může trvat výrazně déle a zpracování HTTP požadavků probíhá až po kompilaci všech stránek aplikace.

4.2 Komponenta WebView

V sekci 3.4.3 o .NET MAUI byl uveden postup tvorby multiplatformní komponenty a handleru. Tohoto přístupu využívá řešení integrace Blazor a .NET MAUI pro tvorbu multiplatformní komponenty WebView. Pro každou platformu existuje element reprezentující WebView, jehož podstatné implementační detaily jsou v této sekci uvedeny.

Windows WebView2

Windows `WebView2`[6] je komponenta využívající Microsoft Edge k vykreslování webových stránek v nativních aplikacích. Současně nabízí také integrované funkcionality ve formě API. Pro řešení této práce jsou nejdůležitější následující:

- `CoreWebView2.PostWebMessageAsJson`, `CoreWebView2.WebMessageReceived` – akce zajišťující interoperabilitu mezi webem (javascript) a nativním kódem pomocí jednoduchých zpráv, javascript kódu a nativních objektů. První metoda slouží k odesílání a druhá událost k přijímání komunikačních zpráv ve formátu JSON.
- `CoreWebView2.WebResourceRequested` – událost odchycující veškeré URL požadavky odpovídající nastavenému filtru metodou `AddWebResourceRequestedFilter`, která může filtrovat požadavky dle URL a typu webových prostředků (font, obrázek, skript, ...).
- `CoreWebView2.AddScriptToExecuteOnDocumentCreatedAsync` – asynchronní metoda přijímající řetězec kódu v jazyce javascript, který bude spuštěn před zpracováním HTML kódu a spuštěním jiných skriptů.
- `CoreWebView2.ExecuteScriptAsync` – asynchronní metoda spouštějící kód javascript na vstupu.
- `CoreWebView2.NavigationStarting` – událost nastávající při navigování na jinou URL.

Android WebView

Android`WebView`[7] využívá Chromium k zobrazení webových stránek. Některé podstatné funkce poskytuje přímo, jiné jsou zprostředkovány souvisejícími třídami `WebViewClient` a `WebChromeClient`, které je možné nastavit příslušnými metodami `setWebViewClient` a `setWebChromeClient`. Další potřebné metody třídy WebView jsou:

- `evaluateJavascript` – metoda sloužící ke spuštění kódu javascript.
- `createWebMessageChannel` – vytváří komunikační kanál s javascriptem a navrací koncové porty, které jsou již provázané a v zahájeném stavu. Na portu je možné nastavit tzv. *callback* pomocí metody `setWebMessageCallback`, jenž se vyvolá při obdržení zprávy ze strany jazyka javascript, nebo také zavolat na daném portu metodu `postWebMessage`, která odešle zprávu na druhý port webu.
- `loadUrl` – načte danou URL.

[6] WebView2: https://learn.microsoft.com/en-us/microsoft-edge/webview2
[7] WebView Android: https://developer.android.com/reference/android/webkit/WebView

Třída `WebViewClient` poskytuje následující využitelné metody:

- `shouldInterceptRequest` – metoda odchycující požadavky a nabízející vlastní implementaci jejich zpracování. Pokud je návratová hodnota `null`, bude požadavek zpracován obvyklým způsobem, jinak bude použit poskytnutá odpověď.
- `shouldOverrideUrlLoading` – metoda umožňující provedení akcí těsně před načtením nové URL. Pokud navrácena hodnota `false`, bude URL načtena běžným způsobem, jinak bude načtení přerušeno.
- `onPageFinished` – metoda vyvolána dokončením načtení stránky.

iOS WKWebView

iOS `WKWebView`[8] používá WebKit framework k vykreslování webového obsahu a umožňuje volání následujících metod:

- `evaluateJavaScript` – metoda sloužící ke spuštění kódu javascript.
- `loadRequest` – načte danou url.
- `navigationDelegate` – objekt sloužící ke správě navigačních akcí.

Další použitelné metody poskytuje třída `UserContentController`, která je dostupná přes konfiguraci `WKWebView`, jedná se o metody:

- `addScriptMessageHandler` – metoda registrující handler komunikačních zpráv, který zachycuje zprávy obdržené z kódu javascript.
- `AddUserScript` – metoda spouštějící kód v jazyce javascript ve formátu řetězce s možností nastavení momentu spuštění.

Odchycení požadavků je na platformě iOS složitější, jelikož není povoleno odchycení požadavků se schématem http/https. Je tedy nutné změnit schéma na jakýkoliv jiný řetězec, například „dotvvm“. Takové požadavky může poté odchytit třída implementující rozhraní `IWKUrlSchemeHandler` metodou `StartUrlSchemeTask`. Registrace třídy společně s žádaným schématem je možná metodou `SetUrlSchemeHandler`.

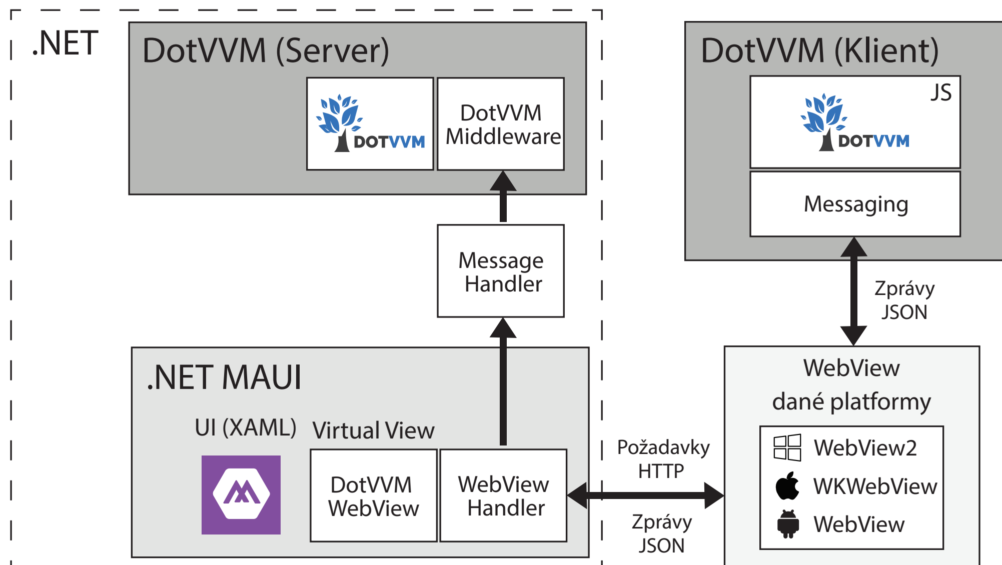
[8] WKWebView: https://developer.apple.com/documentation/webkit/wkwebview

Kapitola 5

Návrh

Hlavní součástí implementovaného řešení, popsaného diagramem 5.1, je komponenta *WebView*, která slouží k vykreslování webového obsahu a figuruje jako prostředník komunikace mezi .NET MAUI a klientské části DotVVM. Její specifická implementace pro každou z platforem by měla být schopna odchytávat požadavky tak, aby namísto odeslání požadavku na daný vzdálený webový server určený URL, zpracovala požadavky lokálně. Zpracování bude zajišťovat služba nazvaná v diagramu jako *Dotvvm Middleware*.

Každá implementace WebView pro danou platformu umožňuje nějakým způsobem spouštět skripty v jazyce javascript, odesílat do něj zprávy ve formátu JSON a také je přijímat. Těchto možností bude využito pro komunikaci s klientskou částí DotVVM, která běží v běhovém prostředí javascript, skrze rozhraní *Messaging*. Přeposílané zprávy z tohoto směru budou využity k volání metod na straně .NET MAUI pomocí modulu *WebView Handler* a k obsluze HTTP požadavků metody POST přes službu *Message Handler*. HTTP požadavky metody GET budou odchytávány komponentou WebView napřímo, nicméně to není možné u metod POST na platformě Android (resp. nelze přistoupit k tělu zprávy u odchyceného požadavku). Proto bude nutné tento typ požadavků zahrnout do komunikačního rozhraní a odesílat jej tak jako obsah v komunikační zprávě. Ze strany .NET MAUI bude sloužit komunikace k získání a modifikaci aktuálního stavu DotVVM skrze *WebView Handler*.



Obrázek 5.1: Návrh komunikace mezi DotVVM a MAUI pomocí komponenty WebView.

Sestavení požadavku a odpovědi do formátu, jenž je schopno DotVVM zpracovat bude probíhat v modulu značeném *Message Handler*. Zde bude také zajištěno zpracování, deserializace a provedení adekvátních akcí na základě typu obdržených zpráv komponentou WebView na komunikačním rozhraní s MAUI nebo klientskou částí DotVVM. Zprávy se budou předávat ve formátu JSON a budou mít strukturu uvedenou na ukázce 5.1.

```
1 {
2     "type": "PatchViewModel",
3     "messageId": 3,
4     "payload": {
5         { "Title": "Patched page" }
6     },
7     "errorMessage": null
8 }
```

Výpis 5.1: Schéma komunikační zprávy pro přenos dat mezi komunikačními rozhraními MAUI, WebView a DotVVM.

Typ zprávy bude vyjadřovat jednu z možných akcí, která je vyvolána klientskou částí DotVVM nebo MAUI, a číslo zprávy bude identifikátor pro odlišení jejich odpovědí ze strany jazyka javascript. Typy akcí a jejich příslušné obsahy (tzv. *payload*) budou následující:

- ***HttpRequest*** – Zpracování HTTP požadavků typu POST přicházejících ze strany jazyka javascript, které nedokáže WebView na platformě Android odchytit. Obsah příchozí zprávy bude v tomto případě obsahovat veškeré náležitosti běžného HTTP požadavku, tedy URL, metodu, hlavičky a tělo zprávy. Odpovídat bude formátu na ukázce 5.2.

```
{
    "url": "/",
    "method": "POST",
    "headers": [
        {
            "key": "content-type",
            "value": "application/json"
        },
        {
            "key": "x-dotvvm-postback",
            "value": "true"
        }
    ],
    "bodyString": { ... }
}
```

Výpis 5.2: Příklad formátu obsahu příchozího HTTP POST požadavku ze strany klienta DotVVM.

Obdobnou strukturu (na ukázce 5.3) bude mít i odpověď požadavku, která bude odesílána zpět klientské části DotVVM.

```
{
    "statusCode": "200",
    "headers": [ ... ],
    "bodyString": { ... }
}
```

Výpis 5.3: Příklad formátu obsahu odpovědi na příchozí HTTP POST požadavek.

- ***GetViewModelSnapshot*** – Získání stavu viewmodelu DotVVM stránky ze strany MAUI. Formát obsahu zprávy se dynamicky mění, bude se tedy jednat čistě o řetězec ve formátu JSON.
- ***PatchViewModel*** – Modifikace stavu viewmodelu DotVVM stránky ze strany MAUI. Lze využít k aplikaci změn jednotlivých vlastností. Obsah zprávy bude reprezentovat pouze úspěšnost provedené operace.
- ***InitCompleted*** – Informuje MAUI o dokončení inicializace klientské části DotVVM a názvu aktuální cesty stránky.
- ***PageNotification*** – Volání metod viewmodelu stránky MAUI ze strany DotVVM. Může být použito například k přístupu a modifikaci MAUI komponent. Formát obsahu komunikační zprávy je uveden v následující ukázce 5.4:

```
{
    "methodName": "UpdateMauiViewmodel",
    "arguments": [ { ... } ]
}
```

Výpis 5.4: Příklad formátu obsahu zprávy udávající provedení akce na straně MAUI.

- ***ErrorOccurred*** – Informuje o selhání zpracování akce na straně DotVVM.

Kapitola 6

Implementace

Framework .NET MAUI slouží k vývoji klientských multiplatformních aplikací a usnadňuje přístup k nativním funkcionalitám platforem, zatímco DotVVM nabízí možnost tvorby webových aplikací, a to jak serverové, tak klientské části, pomocí návrhového vzoru MVVM, jež není pro webový vývoj běžný. Po prozkoumání open-source repozitáře DotVVM[1] je patrné, že se zdrojový kód člení na následující, pro tuto práci důležité, projekty:

- `DotVVM.Framework` – obsahuje většinu interní klientské i serverové logiky jako je například parser a kompilátor stránek, viewmodelů, implementované základní webové komponenty a směrovací logika. Nedílnou součástí jsou také middleware služby (sekce 4.1.3) podílející se na obsluze požadavků a v diagramu 5.1 je lze abstrahovat jako serverovou část DotVVM. Výchozím bodem u klientské části, implementované v jazyce TypeScript, je soubor `dotvvm-root.ts`. Jde v podstatě o rozhraní poskytující užitečné funkce, mezi něž patří získání klientského stavu viewmodelu (funkce `state`) a také modifikace pomocí funkce `patchState`, která na vstupu přijímá rozdíl stavu ve formátu JSON.

- `DotVVM.Hosting.AspNetCore` – projekt je závislý na projektu `DotVVM.Framework`, ale přidává k němu balíček *ASP.NET Core*, který poskytuje navíc autorizaci a další přídavné služby. Projekt slouží k přímé referenci z webového projektu (tedy vyvíjené aplikace) a zároveň poskytuje metodu pro registraci frameworku do DI[2] kontejneru.

Jelikož jsou oba frameworky postaveny na platformě .NET, byla by předpokladem bezproblémová integrace těchto technologií, nicméně *ASP.NET Core* není pro běhové prostředí na platformě Android podporováno[3]. Je tedy nutné nejprve přeimplementovat projekt `DotVVM.Hosting.AspNetCore` na nový projekt `DotVVM.Hosting.Maui` takovým způsobem, aby nebyl závislý na *ASP.NET Core*.

6.1 Registrace DotVVM

Nejprve je třeba zajistit registraci nezbytných služeb DotVVM do aplikace .NET MAUI. Tu lze provést registrací služeb do kolekce služeb `IServiceCollection` dostupné ve třídě

[1] DotVVM open-source repozitář: https://github.com/riganti/dotvvm/tree/main/src/Framework
[2] DI (Dependency Injection): https://learn.microsoft.com/en-us/dotnet/architecture/maui/dependency-injection
[3] Chybějící podpora ASP.NET Core na platformě Android: https://github.com/dotnet/aspnetcore/issues/35077

`MauiProgram`, jež slouží jako vstupní bod MAUI aplikace. Výňatek kódu 6.1 reprezentuje žádaný způsob registrace.

```
builder.AddMauiDotvvmWebView<DotvvmStartup>(applicationPath, debug: true,
    configure: config =>
    {
        config.Markup.ViewCompilation.Mode = ViewCompilationMode.Lazy;
    });
```

Výpis 6.1: Žádaný způsob registrace služeb DotVVM je přes rozšiřující metodu (tzv. *extension method*) třídy `MauiAppBuilder`. Jako parametr je uvedena kořenová cesta k DotVVM aplikaci, nastavení ladícího módu a možnost modifikace konfigurace. V případě tohoto projektu je žádané nastavení kompilačního módu stránek (konfigurace DotVVM 4.1.3) na mód `Lazy`, jelikož by výchozí způsob kompilace velice zpomalil spouštění aplikace.

Analyzovaný projekt `DotVVM.Hosting.AspNetCore` již obsahuje metodu registrující potřebné služby, avšak některé služby jsou závislé na *ASP.NET Core* a nelze je tak v MAUI projektu použít. Zbylé nezávislé služby jsou registrovány interní rozšiřující metodou frameworku `RegisterDotVVMServices`. Veškeré služby, na nichž je DotVVM závislé, mají vlastní rozhraní, díky němuž je možné vytvořit vlastní implementaci. Bylo nutné přeimplementovat následující služby:

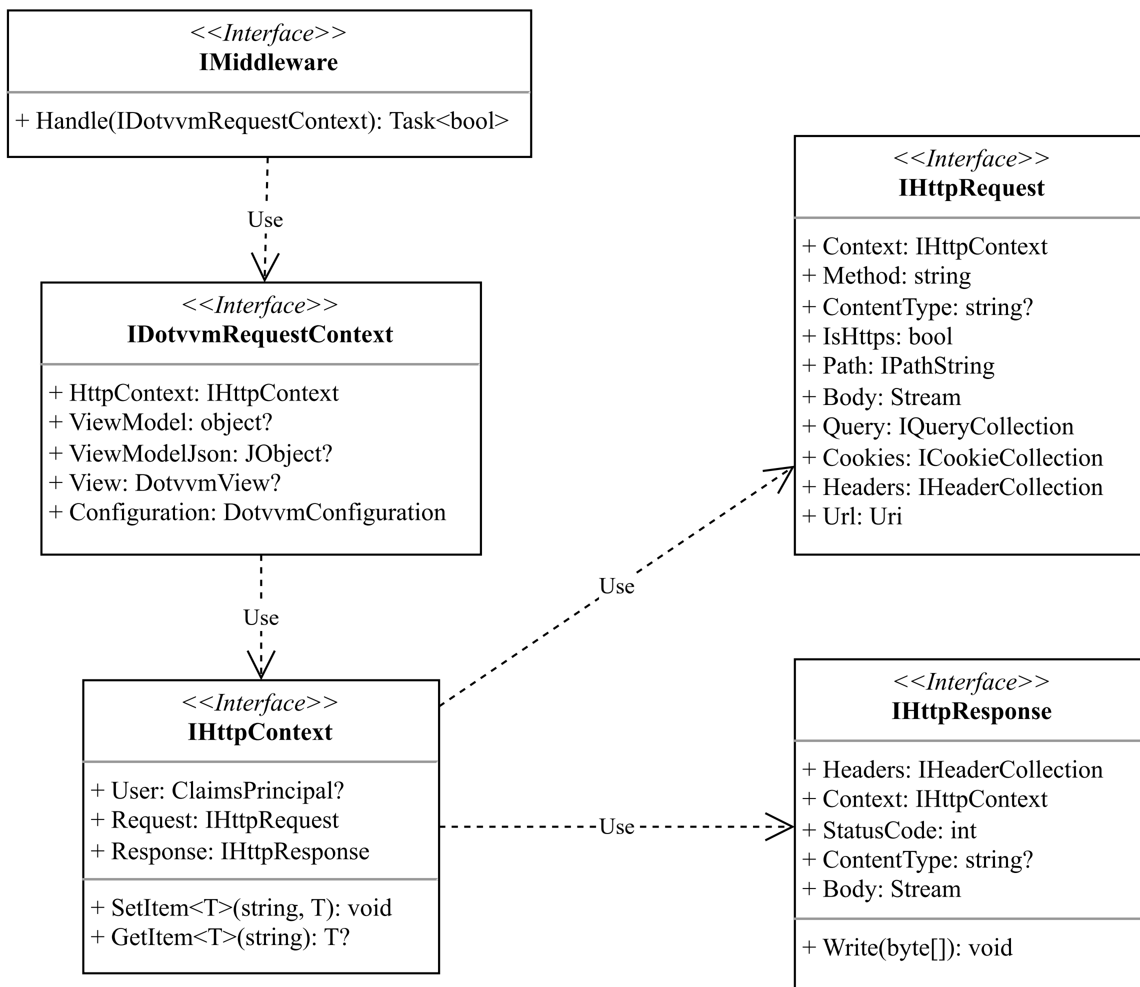
- `ICsrfProtector` – služba zabraňující CSFR útokům. Pro tento projekt je zanedbatelná, jelikož veškerá komunikace probíhá lokálně a není tak možné, aby k útoku došlo. Implementace je stále nutná, proto služba `WebViewCsrfProtector` poskytuje konstatní token, který je vždy úspěšně ověřen.
- `IViewModelProtector` – další služba zajišťující bezpečnost, konkrétně šifrování serializovaného viewmodelu, jež je při lokálním použití redundantní. Z toho důvodu služba `WebViewViewModelProtector` navrací nedotknutý viewmodel.
- `WebViewEnvironmentNameProvider` – jde čistě o poskytování názvu prostředí, v kterém webová aplikace běží. Slouží pouze k interní komponentě vykreslující jiný obsah stránky při různých typech prostředí. Provedená implementace poskytuje pouze produkční a vývojové prostředí.

6.2 Služba DotVVM Middleware

Jelikož byl stávající `DotvvmMiddleware` použit v ASP.NET Core pipeline jako middleware, automaticky mu byl HTTP požadavek předán při jeho samotném zpracování ostatními službami ASP.NET middleware. Nyní musí poskytovat instanční metodu, která zajistí zpracování požadavku všemi DotVVM middleware a navrátí odpověď. Signatura této metody nazvané `Invoke` je uvedena na ukázce 6.2.

```
public async Task<IHttpContext> Invoke(
    Uri requestUri,
    string method,
    IEnumerable<KeyValuePair<string, string>> headers,
    Stream contentStream);
```

Výpis 6.2: Signatury metody obsluhující DotVVM (HTTP) požadavek s následujícími parametry: URL požadavku, metoda HTTP požadavku, HTTP hlavičky a tělo požadavku.



Obrázek 6.1: UML diagram znázorňující návazné závislosti rozhraní `IMiddleware`, které bylo třeba implementovat.

Služby DotVVM middleware typicky implementují rozhraní `IMiddleware` (ukázka 6.3) a při jejich provolání (ukázka 6.4) je nutný parametr `IDotvvmRequestContext`, který slouží k reprezentaci kontextu daného požadavku. Jelikož jeho existující implementace vyžaduje `IHttpContext` pro vytvoření instance, v rámci implementace je vytvořena nová třída `DotvvmHttpContext` implementující toto rozhraní. Ta je nicméně závislá na dalších 2 neimplementovaných třídách – `DotvvmHttpRequest` a `DotvvmHttpResponse`. Bylo tedy nutné naimplementovat převodník mezi vstupními parametry metody `Invoke` 6.2 a všemi závislostmi `IDotvvmRequestContext` ukázány na diagramu 6.1. Šlo především o rozložení URL požadavku a hlaviček na jejich jednotlivé části v určeném formátu a také sestavení odpovědi, jež je následně jedním ze služeb middleware naplněna případnými daty. Pokud nedokáže žádný middleware daný požadavek obsloužit, je nastaven stavový kód na 404.

```
public interface IMiddleware
{
    Task<bool> Handle(IDotvvmRequestContext request);
}
```

Výpis 6.3: Rozhraní pro DotVVM middleware, jehož metody návratová hodnota udává, zda byl požadavek obsloužen nebo má být předán další službě middleware.

```
var context = CreateDotvvmContext(requestUri, method, headers, contentStream, scope);

foreach (var middleware in middlewares)
{
    if (await middleware.Handle(context))
    {
        return context.HttpContext;
    }
}
dotvvmContext.HttpContext.Response.StatusCode = 404;
```

Výpis 6.4: Ukázka části kódu uvnitř metody `Invoke`, jejíž vstupní parametry jsou použity k volání metody `CreateDotvvmContext` zajišťující vytvoření kontextu `IDotvvmRequestContext`. Ten slouží jako vstup jednotlivým službám middleware v uvedeném cyklu.

V tuto chvíli je možné zpracovávat požadavky v předepsaném formátu pomocí DotVVM voláním instanční metody `Invoke` třídy `DotvvmMiddleware`. Následujícím krokem je implementace služby umožňující práci se souborovým systémem.

6.3 Přístup k lokálním souborům

Jelikož DotVVM kompiluje stránky za běhu aplikace a zároveň čte webové prostředky ze souborového systému, je nutné tuto záležitost ošetřit. Důvodem je odlišnost souborových systémů na různých platformách a nemožnost vytvářet soubory nebo rekurzivně procházet adresáře. Zde přichází na řadu použití MAUI `MAUI FileSystem`[4]. Jde o pomocnou třídu, která je součástí .NET MAUI a díky ní je možné přistupovat k souborům zabaleným v aplikaci, které slouží pouze pro čtení, a ke kořenové cestě aplikace multiplatformně.

Aby bylo možné číst soubory přibalené k aplikaci, je nutné nastavit v projektu C# jejich akci sestavení (*build action*) na typ `MauiAsset`. Tohoto přístupu je využito v implementované službě middleware `WebViewFileSystemMiddleware` při poskytování webových prostředků jako jsou: kaskádové styly, skripty, obrázky apod. Vstupem služby je požadavek, který odkazuje na URL webového prostředku, díky němuž je obsah požadovaného souboru překopírován do výstupní odpovědi, nicméně musí být také přiřazen typ internetového média (tzv. *Mime type*[5]). Ten je prováděn knihovnou *MimeTypesMap*[6], která podle přípony souboru navrátí standardizovaný typ internetového média a naopak.

Dalším případem užití překopírování souborů je čtení stránek s příponami `.dothtml`, `.dotcontrol` a `.dotmaster` pro účel kompilace při běhu aplikace. Jelikož velká část DotVVM služeb pracuje se statickou třídou `File` z jmenného prostoru `System.IO`, bylo by velmi

[4]FileSystem: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/file-system-helpers
[5]Mime type: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
[6]Knihovna MimeTypesMap: https://github.com/hey-red/MimeTypesMap

náročné její nahrazení. Proto byla naimplementována třída `MauiDotvvmFileProvider`, která využívá způsobu přibalení stránek do projektu a jejich následné překopírování pomocí metod `FileSystem.OpenAppPackageFileAsync` a `File.WriteAllTextAsync` s kombinací cesty do kořenového adresáře aplikace.

6.4 Integrace komunikace

V této fázi je možné zachycovat požadavky díky metodám uvedených v podkapitole 4.2 a zpracovávat je pomocí služby implementované v sekci 6.2. Následující část se věnuje implementaci komunikace klientské části DotVVM s WebView a jejími pomocnými metodami. Odesílání zpráv z klientské strany je ukázáno na výňatku 6.5, přijímání na 6.6.

```
1 export function sendMessage(message: WebViewMessageEnvelope) {
2     (window.external as any).sendMessage(JSON.stringify(message));
3 }
```

Výpis 6.5: Ukázka vstupního bodu pro odesílání komunikačních zpráv v jazyce TypeScript. Volání této metody vyvolá událost příslušné WebView třídy. Například na platformě Windows bude vyvolána událost `CoreWebView2.WebMessageReceived` s konkrétní zpávou.

```
1 (window.external as any).receiveMessage(async (json: any) => {
2     const envelope = <WebViewMessageEnvelope>JSON.parse(json);
3     const response = await processRequestOrResponse(envelope);
```

Výpis 6.6: Ukázka vstupního bodu přijímání komunikačních zpráv v jazyce TypeScript. Uvedená anonymní metoda (na ukázce je pouze její část) je vyvolána například metodou `CoreWebView2.PostWebMessageAsJson` na platformě Windows.

V návrhu (kapitola 5) byla definována struktura komunikačních zpráv, které jsou v implementaci reprezentovány ekvivalentními třídami a to, jak v jazyce C#, tak v jazyce TypeScript. Serializace a deserializace z/do formátu JSON je tedy podstatně přívětivější.

DotVVM používá na klientovi funkci `window.fetch`[7], která však musela být nahrazena z objasněného důvodu na začátku této kapitoly. Funkce je zachována pro požadavky metody GET, nicméně pro POST je využita interoperabilita nabízená komponentou WebView. Nová implementace se nachází ve funkci `webMessageFetch`, kde je sestavena komunikační zpráva s typem akce *HttpRequest* a následně je odeslána funkcí `sendMessage` (ukázka 6.5). V tomto okamžiku čeká (pomocí tzv. *promise*[8]) zpráva na zpracování serverovou částí DotVVM a její odpověď, kterou dále zpracuje klientská část interně.

Anonymní metoda uvedená na ukázce 6.6 obdržuje zprávu, u které je zjištěn typ zprávy, a pokud se jedná o *HttpRequest*, jde o odpověď požadavku POST zmíněného výše. V dalších případech může jít o *GetViewModelSnapshot*, jemuž je zaslána zpráva zpět se serializovaným stavem DotVVM (globální objekt `dotvvm.state`), nebo *PatchViewModel*, jehož zpracování je uvedeno na výpisu 6.7.

[7]window.fetch: Funkce sloužící pro odesílání HTTP požadavků.
[8]Promise: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

```
if (envelope.type == "PatchViewModel") {
    try {
        dotvvm.patchState(envelope.payload);
    }
```

Výpis 6.7: Ukázka v jazyce typescript. Typ zprávy *PatchViewModel* aplikující změny na DotVVM viewmodel.

Poslední částí řešení je navrhovaný `WebViewMessageHandler`, který deserializuje komunikační zprávy přijaté z klienta a dle jejich typu provede adekvátní akce. Pokud se jedná o typ *HttpRequest*, jde o POST požadavek, na který je aplikována služba `DotvvmMiddleware` (sekce 6.2) a odpověď je odeslána zpět pomocí metody `PostWebMessageAsJson`. Jestliže se jedná o *PageNotification*, je přes vyvolání události ve WebView informováno MAUI. V případech *GetViewModelSnapshot* a *PatchViewModel* jde o odpověď zprávy, která je určena MAUI stránce, jelikož jí byla vyvolána.

Kapitola 7

Testování

Výsledkem implementace navrženého řešení je knihovna umožňující hostování frameworku DotVVM v aplikaci MAUI. Testování dosažených výsledků bylo provedeno implementací webové aplikace a následné tvorby MAUI aplikace, do níž byla za pomocí naimplementované knihovny zahostována.

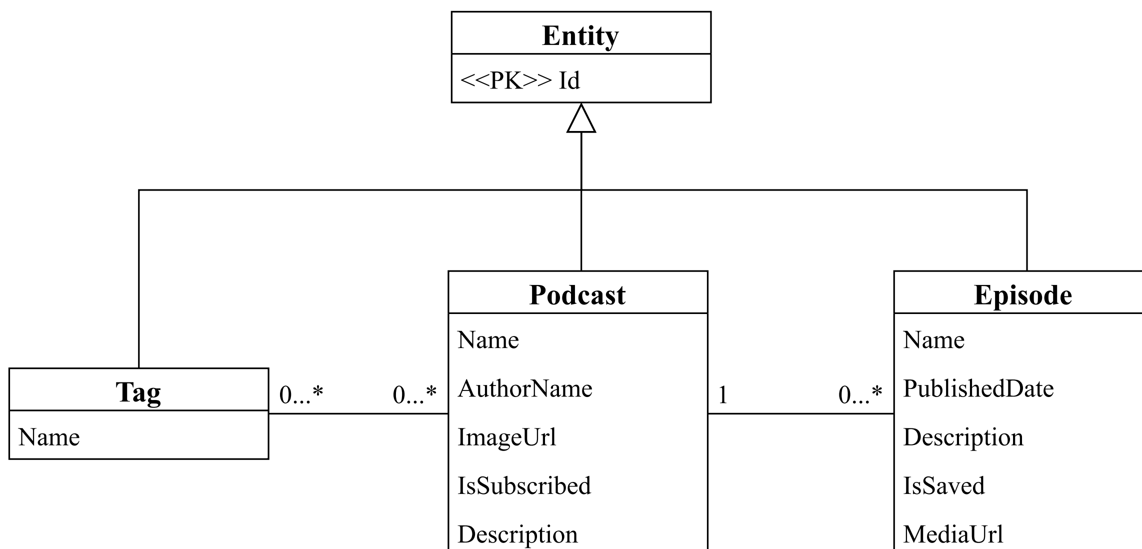
7.1 Návrh vzorové aplikace

Pro demonstraci hybridního přístupu byla navržena aplikace, jejíž uživatelské rozhraní bylo inspirováno existující webovou aplikací[1]. Bude umožňovat následující funkcionalitu:

- Přehrávání epizod podcastu – přehrávaná epizoda zobrazuje celkový čas, aktuální čas a nabízí zapnutí, pozastavení i nastavení úrovně hlasitosti.
- Zobrazení seznamu podcastů – seznam podcastů obsahuje základní informace a je možné jej otevřít pro detail.
- Zobrazení detailu podcastu – podrobné informace o podcastu včetně názvu autora, obrázku, popisu a možnosti přidání podcastu do seznamu oblíbených. Stránka obsahuje také seznam epizod, které mohou být uloženy na později.
- Zobrazení seznamu oblíbených podcastů.
- Zobrazení seznamu epizod uložených na později.
- Možnost spuštění funkce tzv. *seeding* pro uživatelské nahrání vzorových dat do lokální databáze aplikace.

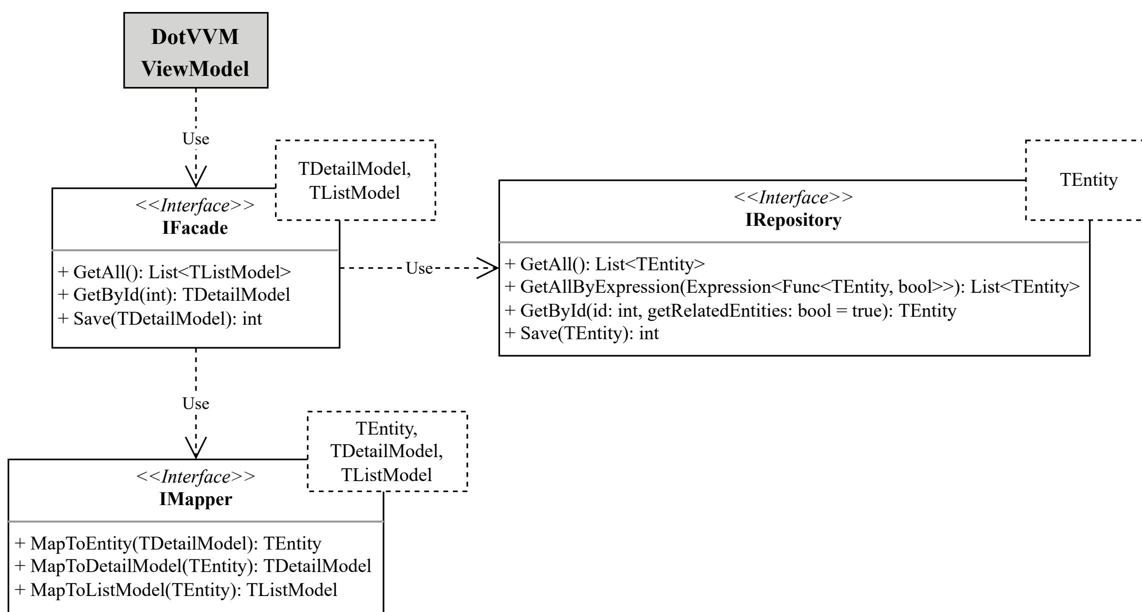
Pro ukládání dat byla využita relační databáze *SQLite*[2] a její rozšíření *SQLiteNetExtensions*[3] přidávající anotaci relace *ManyToMany*. Navrhnuté entity a vazby mezi nimi jsou uvedeny na diagramu 7.1.

[1] DotNetPodcasts: https://dotnetpodcasts.azurewebsites.net/discover
[2] SQLite: https://github.com/praeclarum/sqlite-net
[3] SQLiteNetExtensions: https://bitbucket.org/twincoders/sqlite-net-extensions



Obrázek 7.1: ER diagram databáze vzorové aplikace.

Architektura aplikace je znázorněna na diagramu 7.2. Přístup viewmodelu stránky k repozitáři a mapování entity na model stránky je zastíněn fasádou. Je podporováno načtení seznamu modelů určených pro zobrazení v seznamu, modelu pro stránku s detailem a také uložení pro aktualizaci dat. Mazání není rozhraním poskytováno, jelikož pro něj není v této aplikaci způsob užití.



Obrázek 7.2: Hlavní část architektury znázorňující závislosti vycházející z viewmodelu DotVVM stránky reprezentována jazykem UML. Typy `TDetailModel`, `TListModel` a `TEntity` jsou generické typy pro jednotlivé entity a jejich příslušné modely pro stránky detailu entity a seznamu entit, jež jsou pomocí implementovaných rozhraní `IMapper` mezi sebou mapovány.

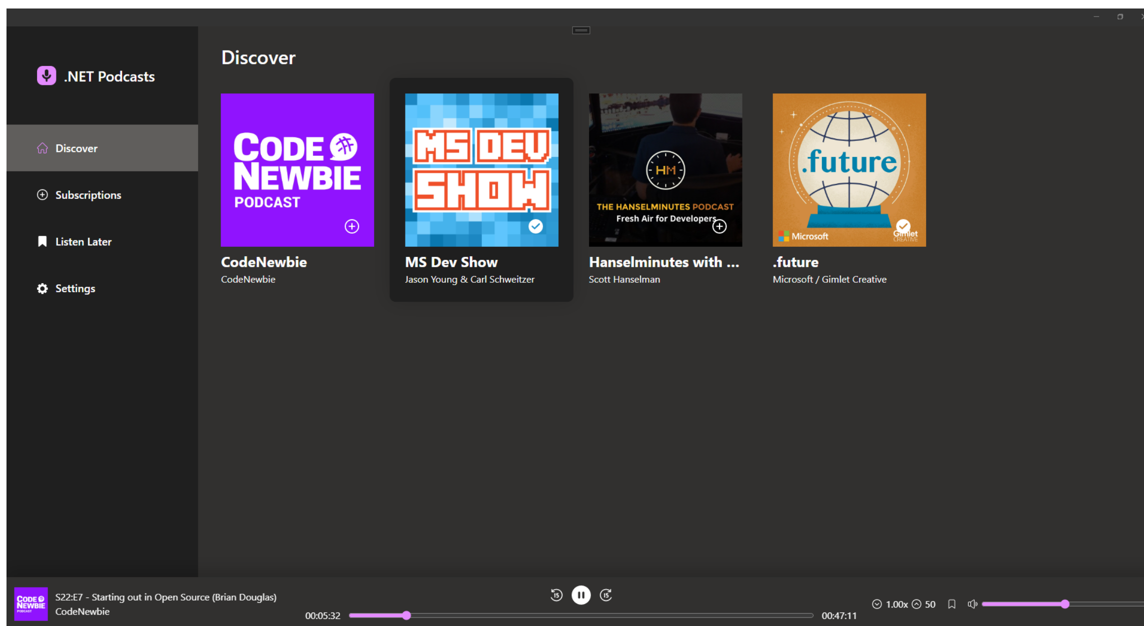
7.2 Implementace vzorové aplikace

Prvním krokem bylo vytvoření webových stránek v jazyce DotHTML a jejich nastylování pomocí syntaxe SCSS a preprocesoru Saas[4]. Byly naimplementovány jednotlivé stránky a komponenty (sekce 4.1.2):

- `MasterPage` – Kořenová stránka aplikace, do jejíž dynamické části jsou zanořeny zbylé stránky a komponenty. Je tvořena sdílenými části pohledu jako je navigační menu a přehrávač epizod (komponenta `EpisodePlayer`, kterou lze vidět na spodní části obrázku 7.3).
- `Default.dothtml` – Výchozí stránka (obrázek 7.3), která je načtena při spuštění aplikace. Je zde vykreslen seznam podcastů (komponenty `PodcastCard`) způsobem uvedeným na ukázce 7.1.

```
1 <h1>Discover</h1>
2 <dot:Repeater DataSource="{value: Podcasts}" class="podcast-list">
3     <ItemTemplate>
4         <cc:PodcastCard Podcast="{value: _this}"
5                         IconClicked="{command: _root.ToggleSubscribe(_this)}" />
6     </ItemTemplate>
7 </dot:Repeater>
```

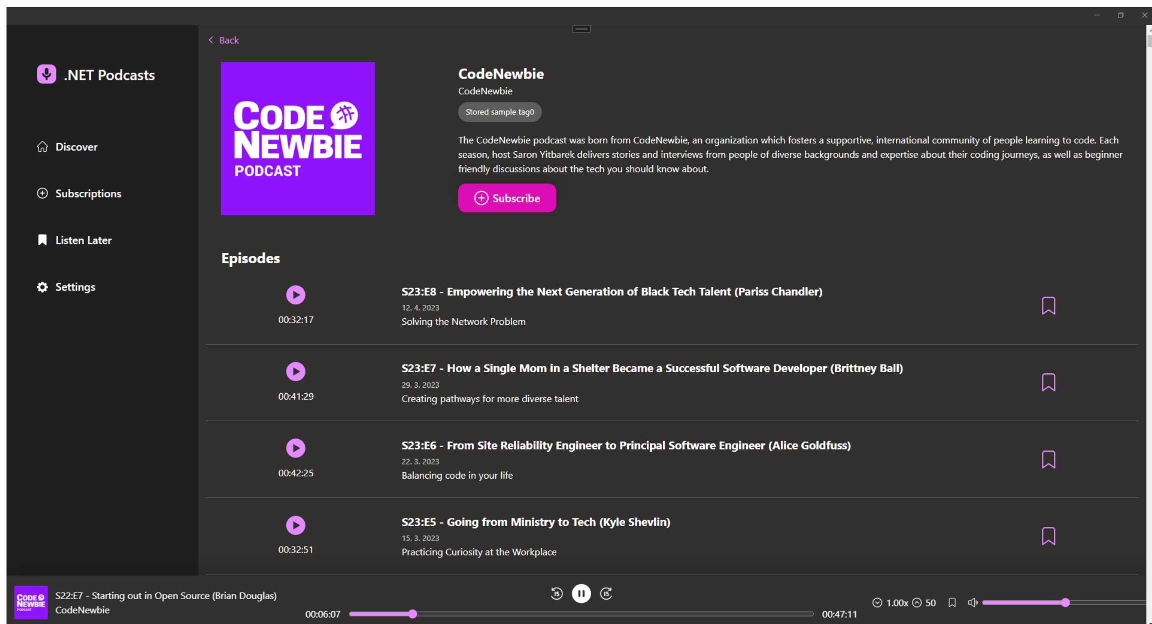
Výpis 7.1: Ukázka vykreslení jednotlivých prvků dynamického seznamu podcastů pomocí DotVVM komponenty `Repeater`. Jednotlivé podcasty jsou reprezentovány šablonou `ItemTemplate` a tvořeny komponentou `PodcastCard`.



Obrázek 7.3: Nastylovaná výchozí stránka vzorové aplikace.

- `PodcastDetail` – Slouží k zobrazení detailu podcastu a je možné se na tuto stránku navigovat přes výchozí stránku. Naimplementovanou stránku lze vidět na obrázku 7.4.

[4] Sass: https://sass-lang.com/documentation/syntax



Obrázek 7.4: Nastylovaná stránka detailu podcastu obsahující seznam přehrávatelných epizod.

- `SubscribedPodcasts`, `ListenLater.dothtml` – Stránky zobrazující seznam podcastů/epizod. Využita je metoda `GetAllByExpression` pro získání všech entit odpovídajících určitému výrazu, v tomto případě všech epizod, pokud je pravdivostní hodnota jejich atributu `IsSaved` pravda.

- `Options` – Slouží k naplnění lokální databáze reálnými dynamickými daty použitím služby `Seeder`, která z vybraných Rss zdrojů stáhne pomocí HTTP klienta[5] jednotlivé epizody.

- `Episode` – Jedná se o reprezentaci položky seznamu podcastů, která pomocí vlastního viewmodelu ukládá epizodu na pozdější přehrání. Provázání viewmodelu ze stránky je možné vidět na ukázce 7.2.

```
1 <cc:Icon Name="save"
2         ShowIcon="{value: !IsSaved}"
3         Events.Click="{command: _root.ToggleEpisodeBookmark(_this)}" />
4 <cc:Icon Name="save-active"
5         ShowIcon="{value: IsSaved}"
6         Events.Click="{command: _root.ToggleEpisodeBookmark(_this)}" />
```

Výpis 7.2: Ukázka volání metody pro uložení epizody na pozdější přehrání včetně dynamického zobrazení ikon. Pokud je epizoda uložena, zobrazí se ikona uložení s výplní (aktivní), pokud ne, je zobrazena ikona bez výplně (neaktivní).

- `EpisodePlayer` – Komponenta komunikující s MAUI pomocí zprávy typu *PageNotification* (kapitola 5) a nabízející možnost uložit epizodu stejným způsobem jak komponenta `Episode`. Aby bylo možné perzistovat nastavení aktuálně přehrávaného média,

[5] https://learn.microsoft.com/cs-cz/dotnet/api/system.net.http.httpclient

je použita třída Preferences[6], která slouží k ukládání aplikačních preferencí ve formě klíč-hodnota. Jde například o hlasitost a rychlost přehrávání média.

- PodcastCard – Reprezentuje jednotlivé položky seznamu podcastů na výchozí stránce. Kromě zobrazení základních dat o podcastu včetně obrázku, propaguje událost kliknutí (atribut IconClicked na ukázce 7.1) na komponentu do výchozí stránky, která na základě ní uloží podcast do seznamu oblíbených.

- Icon.dotcontrol – Komponenta repreztentuje HTML element svg, který pomocí elementu use[7] dynamicky zobrazuje různý obsah (ikony).

Důležitým prvkem vzorové aplikace je multiplatformní element MediaElement[8], který umožňuje přehrávání audia lokálně i vzdáleně pomocí URI (HTTP/HTTPS). Je součástí knihovny Community Toolkit (sekce 3.4.3) a využívá se ve stránce XAML, jak je ukázáno na příkladu 7.3. Po přidělení atributu x:Name, je dostupný přehrávač pod nastaveným názvem v C# třídě dané stránky.

```
1 <toolkit:MediaElement x:Name="mediaElement"
2                       ShouldAutoPlay="False"
3                       IsVisible="false" />
```

Výpis 7.3: Ukázka multiplatformního elementu v XAML zajišťujícího přehrávání médií.

Stejným způsobem (ukázka 7.4) je deklarována implementovaná komponenta WebView, u které je možné nastavit událost PageNotificationReceived, kterou bude MAUI aplikace získávat komunikační zprávy od DotVVM. Přiřazená událost je naznačena v ukázce 7.5.

```
1 <dotvvm:DotvvmWebView x:Name="DotvvmPage"
2                       RouteName="{Binding RouteName}"
3                       IsPageLoaded="{Binding IsPageLoaded}"
4                       PageNotificationReceived="DotvvmPage_PageNotificationReceived" />
```

Výpis 7.4: Ukázka multiplatformního elementu v XAML zajišťujícího vykreslování DotVVM stránek ve WebView.

```
1 void DotvvmPage_PageNotificationReceived(object sender, PageNotificationEventArgs e)
2 {
3     var method = GetType().GetMethod(e.MethodName);
4     if (method != null)
5     {
6         method.Invoke(this, e.Arguments);
7     }
8 }
```

Výpis 7.5: Událost obdržení komunikační zprávy z DotVVM, která pomocí reflexe volá danou metodu s danými argumenty. Pro tento projekt je k dispozici volání několika metod – Play, Pause, SetSpeed, SetVolume a Stop. Ty jsou implementovány pomocí elementu mediaElement a jeho příslušných metod.

[6] Preferences: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/preferences

[7] Element use: https://developer.mozilla.org/en-US/docs/Web/SVG/Element/use

[8] MediaElement: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/views/mediaelement

Rozhraní elementu `mediaElement` poskytuje události, po jejichž vyvolání je nutné kontaktovat DotVVM pomocí metody `PatchViewModel`. Jde o události:

- `MediaOpened` – Vyvolána při otevření nového zdroje média. V tento moment je známa délka přehrávaného obsahu, která je předaná DotVVM.
- `PositionChanged` – Vyvolávaná při průběžné i explicitní změně pozice přehrávaného média.
- `StateChanged` – Nastává při změně stavu přehrávaného média, např. při pozastavení.

Implementací vzorové aplikace byla ověřena použitelnost a funkčnost integrace DotVVM a .NET MAUI na platformách Windows a Android. Z klientské strany aplikačního rámce DotVVM je možné volat metody pohledu .NET MAUI, jehož součástí je komponenta DotVVM WebView. Naopak ze strany .NET MAUI lze získat a aktualizovat stav DotVVM. Současně byla demonstrována možnost použití multiplatformních komponent, konkrétně *Media Element* sloužící pro přehrávání médií, s kterým DotVVM komunikuje. Na platformě iOS bylo možné aplikaci sestavit i nasadit, nicméně nebyl v komponentě WebView vykreslen žádný obsah a nepodařilo se zjistit příčinu.

Kapitola 8

Závěr

Cílem této práce bylo zajistit integraci technologií DotVVM a .NET MAUI takovým způsobem, aby spolu dokázaly vzájemně komunikovat, a aby mohla být webová aplikace vyvinutá v DotVVM vykreslena pomocí komponenty WebView v multiplatformní aplikaci .NET MAUI. Předcházela tomu analýza existujících přístupů tvorby mobilních aplikací, interní fungování aplikačního rámce DotVVM a principiálně blízkého řešení integrace frameworku Blazor s .NET MAUI.

Pro otestování funkčnosti řešení byla naimplementovaná responzivní vzorová aplikace pomocí jazyků DotHTML, TypeScript, SCSS a C#. Aplikace umožňuje zobrazení seznamu podcastů, z nichž je možné přejít na stránku detailu podcastu s dalšími informacemi a seznamem přehrávatelných epizod. Součástí uživatelského rozhraní byla vytvořena komponenta reprezentující přehrávač audia, který umožňuje zapnutí, pozastavení, změnu hlasitosti a rychlosti právě přehrávané epizody. Epizody i podcasty je možné uložit do seznamu oblíbených a zobrazit si je na separátní stránce. Rovněž je umožněno získat reálná dynamická data z několika vybraných zdrojů RSS. Vzorová aplikace byla otestovaná na platformách Windows i Android, kde byla rovněž prokázaná použitelnost pro vývojáře webových aplikací v aplikačním rámci DotVVM.

Jelikož jsou veškeré zdrojové kódy obou integrovaných aplikačních rámců open-source pod licencí Apache License 2.0 a MIT, je projekt s řešením a vzorovou aplikací rovněž zveřejněn na platformě GitHub[1].

Součástí této práce nebylo provedení výkonnostního měření běhu aplikace na jednotlivých platformách, což se jeví jako vhodné rozšíření této práce. Další prací do budoucna je řádné otestovaní výsledné aplikace na platformě iOS, jelikož se zde vyskytla komplikace při vykreslování obsahu a nebylo tedy možné ověřit případné nedostatky v uživatelském rozhraní aplikace na této platformě.

[1] https://github.com/tomasmikes/dotvvm-maui-podcast

Literatura

[1] Alwakeel, L. a Lano, K. Model Driven Development of Mobile Applications. In: Open, J. Q. a Access, J. R., ed. *Doctoral Symposium, ECOOP 2020.* 2020.

[2] Amatya, S. *Cross-Platform Mobile Development: An Alternative to Native Mobile Development.* Växjö, SE, 2013. Diplomová práce. Linnaeus University, Department of Computer Science. Dostupné z: http://www.diva-portal.org/smash/get/diva2:664680/fulltext01.pdf.

[3] Beasley, R. E. *Essential ASP.NET Web Forms Development.* 1. vyd. Apress Berkeley, CA, 2020. ISBN 978-1-4842-5784-5.

[4] Biessek, A. *Flutter for Beginners.* 2. vyd. Packt Publishing, 2019. ISBN 978-1-78899-608-2.

[5] Biørn Hansen, A., Grønli, T.-M. a Ghinea, G. A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. *ACM Comput. Surv.* Association for Computing Machinery. 2018, sv. 51, č. 5. DOI: 10.1145/3241739. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3241739.

[6] Biørn Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A. a Ghinea, G. An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering.* 2020, sv. 25, č. 4. DOI: 10.1007/s10664-020-09827-6. ISSN 1573-7616.

[7] Cloudflare, I. *What are cookies? | Cookies definition* [online]. 2022 [cit. 2022-10-20]. Dostupné z: https://www.cloudflare.com/learning/privacy/what-are-cookies.

[8] Denko, B., Pecnik, S. a Fister jr, I. A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks. *International Journal of Security and Privacy in Pervasive Computing.* Leden 2021, sv. 13, č. 1, s. 78–90. DOI: 10.4018/IJSPPC.2021010105.

[9] Eisenman, B. *Learning React Native.* 2. vyd. O'Reilly Media, Inc., 2017. ISBN 9781491989142.

[10] Google LLC. *Flutter architectural overview* [online]. 2023 [cit. 2023-01-06]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview.

[11] Griffith, C. *Mobile App Development with Ionic.* Revised. O'Reilly Media, Inc., 2017. ISBN 9781491998120.

[12] Heitkötter, H., Hanschke, S. a Majchrzak, T. A. Evaluating Cross-Platform Development Approaches for Mobile Applications. In: Cordeiro, J. a Krempels, K.-H., ed. *Web Information Systems and Technologies*. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36608-6.

[13] Herceg, T. *DotVVM + MAUI integration* [online]. 2022 [cit. 2023-04-26]. Dostupné z: https://tomasherceg.com/blog/post/dotvvm-maui-integration.

[14] Ho, T. *React Native will be re-architecture in 2020* [online]. 2020 [cit. 2022-12-07]. Dostupné z: https://itzone.com.vn/en/article/react-native-will-be-re-architecture-in-2020/.

[15] Internet Assigned Numbers Authority. *Internet Assigned Numbers Authority* [online]. 2022 [cit. 2022-12-29]. Dostupné z: https://www.iana.org/.

[16] Jednotliví přispěvatelé mozilla.org. *Identifying resources on the Web* [online]. 2022 [cit. 2022-10-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web.

[17] Kromer, F. *Ionic Angular in a nutshell* [online]. 2021 [cit. 2022-12-29]. Dostupné z: https://florian-kromer.medium.com/ionic-angular-in-a-nutshell-b451d69c0924.

[18] Leon, S. a Rosen, R. *Web Application Architecture : Principles Protocols and Practices*. 2. vyd. Wiley, 2009. ISBN 978-0-470-51860-1.

[19] Lynch, M. *Ionic isn't Cordova Anymore* [online]. 2021 [cit. 2022-11-30]. Dostupné z: https://ionic.io/blog/ionic-isnt-cordova-anymore.

[20] Management, R. D. *OPEN OR PROPRIETARY PROTOCOLS?* [online]. 2018 [cit. 2022-09-18]. Dostupné z: https://www.resourcedm.com/resources/blog/are-open-protocols-better.

[21] Meta Platforms, Inc.. *Virtual DOM and Internals* [online]. 2022 [cit. 2022-12-06]. Dostupné z: https://reactjs.org/docs/faq-internals.html.

[22] Meta Platforms, Inc.. *JavaScript Environment* [online]. 2023 [cit. 2023-01-23]. Dostupné z: https://reactnative.dev/docs/javascript-environment.

[23] Microsoft. *Android Callable Wrappers for Xamarin.Android* [online]. 2022 [cit. 2022-12-17]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/android/platform/java-integration/android-callable-wrappers.

[24] Microsoft. *Architecture - Xamarin* [online]. 2022 [cit. 2022-12-17]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/android/internals/architecture.

[25] Microsoft. *Co je plánování podnikových zdrojů (ERP)? – Microsoft Dynamics 365* [online]. 2022 [cit. 2022-12-26]. Dostupné z: https://dynamics.microsoft.com/cs-cz/erp/what-is-erp/.

[26] Microsoft. *What is Xamarin?* [online]. 2022 [cit. 2022-12-17]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

[27] Microsoft. *What is Xamarin?* [online]. 2022 [cit. 2023-04-11]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/handlers/create.

[28] Microsoft. *ASP.NET Core Middleware* [online]. 2023 [cit. 2023-04-11]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware.

[29] Microsoft. *.NET Multi-platform App UI documentation* [online]. 2023 [cit. 2023-04-11]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui/.

[30] Oracle. *Introduction JNI* [online]. 2020 [cit. 2022-12-25]. Dostupné z: https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html.

[31] R. Fielding, Ed., M. Nottingham, Ed., J. Reschke, Ed.. *HTTP Semantics* [Internet Requests for Comments]. RFC 9110. RFC Editor, červen 2022. Dostupné z: https://www.rfc-editor.org/rfc/rfc9110.html.

[32] Red Hat, Inc.. *What is an IDE?* [online]. 2019 [cit. 2022-11-29]. Dostupné z: https://www.redhat.com/en/topics/middleware/what-is-ide.

[33] RIGANTI s.r.o.. *Introduction | DotVVM Documentation* [online]. 2022 [cit. 2022-12-25]. Dostupné z: https://www.dotvvm.com/docs/4.0/pages/introduction.

[34] Sajjad, Z. *React Native in 2022 and Beyond* [online]. 2022 [cit. 2022-12-07]. Dostupné z: https://semaphoreci.com/blog/react-native.

[35] SMIT Services s.r.o.. *Co je CRM systém?* [online]. 2022 [cit. 2022-12-27]. Dostupné z: https://www.vyber-crm.cz/co-je-crm-system.

[36] Stanojević, J., Šošević, U., Minović, M. a Milovanović, M. An Overview of Modern Cross-platform Mobile Development Frameworks. In: Faculty of Organization and Informatics Varazdin. *Central European Conference on Information and Intelligent Systems*. 2022.

[37] Stonis, M. *Enterprise Application Patterns using .NET MAUI*. 1. vyd. Microsoft Developer Division, 2022. Dostupné z: https://dotnet.microsoft.com/en-us/download/e-book/maui/pdf.

[38] Sulyman, S. Client-Server Model. *IOSR Journal of Computer Engineering*. 1. vyd. Leden 2014, sv. 16, č. 1, s. 57–71. DOI: 10.9790/0661-16195771.

[39] The Apache Software Foundation. *Architectural overview of Cordova platform* [online]. 2022 [cit. 2022-11-27]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[40] The Apache Software Foundation. *Plugin Development Guide* [online]. 2022 [cit. 2022-11-27]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/hybrid/plugins/index.html.

[41] Tichý, M. *Překlad DotVVM stránek do Accelerated Mobile Pages*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22954/.

[42] W3C. *Web Application Manifest* [online]. 2022 [cit. 2022-12-29]. Dostupné z: https://www.w3.org/TR/appmanifest/.

[43] Wargo, J. *Learning Progressive Web Apps*. 1. vyd. Addison-Wesley Professional, 2020. ISBN 978-0136484226.

[44] Wargo, J. M. *Apache Cordova 4 Programming*. 1. vyd. Addison-Wesley Professional, 2015. ISBN 978-0134048192.