Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Diplomová práce

2020 **Bc. Vilém Havel**

Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Mobilní aplikace jako součást grantu Photostruk – Analýza historických fotografií pro virtuální rekonstrukci kulturního dědictví v česko-bavorském příhraničí

Diplomová práce

Bc. Vilém Havel

Školitel: Ing. Jan Fesl Ph.D.

České Budějovice 2020

/

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL MAGISTERSKÉ PRÁCE

Student: Bc. Vilém Havel

Obor – zaměření studia: Aplikovaná informatika

Katedra/ústav PřF JU: Ústav aplikované informatiky

Školitel: Ing. Jan Fesl Ph.D.

Školitel – specialista, konzultant:
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma magisterské práce: Mobilní aplikace jako součást grantu Photostruk - Analýza historických fotografií pro virtuální rekonstrukci kulturního dědictví v česko-bavorském příhraničí

Cíle práce: Mobilní aplikace pro turisty v prostředí Bavorského lesa a Šumavy, která umožní:

- *prohlížet si historické fotografie a rekonstrukce krajiny v aktuálních lokalitách*
- *prohlížet si metadata spojená s historickými fotografiemi a rekonstrukcí krajiny*
- *aktivně přispívat k obsahu databáze Photostruk nově pořízenými fotografiemi i editací a vkládáním metadat*

Hlavní cíl:

- Návrh a realizace mobilní aplikace, která umožní uživatelům v oblasti Bavorského lesa a Šumavy prohlížet historické informace, fotografie a rekonstruované objekty v aktuální lokalitě, kde se bude uživatel nacházet.
- Zmapovat aktuální světové trendy a možnosti pro požadovanou mobilní aplikaci
- Navrhnout scénáře nasazení mobilní aplikace v prostředí různých platforem mobilních zařízení a různé úrovně signálu v terénu.
- Aplikace umožní uživateli také aktivní zpětnou vazbu, kdy bude moci uživatel sám pořizovat v terénu svým mobilním zařízením fotografie a připojovat je k dané lokalitě.
- Časové a lokalizační údaje k pořízeným fotografiím bude mobilní aplikace doplňovat sama.

Dílčí cíl:

- Mobilní aplikaci si bude možné instalovat a provádět update prostřednictvím standardních zdrojů pro instalaci mobilních aplikací pro příslušnou platformu.

- Aplikace bude vytvořena a otestována minimálně pro platformy IOS a Android.
- Zdrojem informací o lokalitě, kde se bude uživatel nacházet, bude databáze Photostruku
- Aplikace umožní uživateli autentizaci, ale bude schopná pracovat i s anonymním uživatelem.
- Aplikace bude schopna dočasně pracovat i v off-line režimu při nedostatečném nebo chybějícím signálu, přičemž nedojde ke ztrátě dat a k synchronizaci dat dojde po opětném navázání komunikace.
- Aplikace upozorní uživatele, pokud nastane potřeba přenosu dat prostřednictvím datového roamingu.

Funkce mobilní aplikace Photostruk:

- Prohlížení informací a fotografií z naší databáze/webu
- Prohlížení 3D vizualizací
- Automatické rozpoznání polohy a nabídnutí informací a fotek o místě (pokud je signál)
- Možnost tagovat, komentovat fotografie
- Možnost vyfotit fotografii ze stejného místa jako je historická fotka a přidat na web/nebo do databáze. S tím souvisí možnost georeferencovat historické fotografie = označit v terénu místo, kde stál fotograf nebo které je na fotce zachycené. Tak aby byl patrný rozdíl, zda označuji místo fotografa nebo foceného objektu.

Základní doporučená literatura:

VisualStudio dokumentace [online]. Redmond: Microsoft, 2019 [cit. 2019-01-09]. Dostupné z: https://visualstudio.microsoft.com
Microsoft dokumentace [online]. Redmond: Microsoft, 2019 [cit. 2019-01-09]. Dostupné z: https://docs.microsoft.com

Školitel práce ....podpis : ....

U externích vedoucích fakultní garant práce..podpis : ....

Garant oboru mag. studia ....podpis : ....

Vedoucí katedry/ústavu PřF JU, kde proběhne obhajoba...........................podpis : ....

Případný souhlas vedoucího ústavu AV ....podpis : ....

V Českých Budějovicích dne 8. 2. 2019 ....podpis studenta: ....

Bibliografické údaje

Havel V., 2020: Mobilní aplikace jako součást grantu Photostruk – Analýza historických fotografií pro virtuální rekonstrukci kulturního dědictví v česko-bavorském příhraničí

[Mobile app as a part of grant Photostruk – Analysis of historical photographs for virtual reconstruction of cultural heritage in Czech-Bavarian borders Mgr. Thesis, in Czech] – 70 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Diplomová práce pojednává o návrhu a následném vývoji multiplatformní mobilní aplikace pro operační systémy Android a iOS. Aplikace byla vytvořena jako doplněk pro webovou aplikaci Photostruk. Tento projekt byl navržen pro analýzu historických fotografií a pro virtuální rekonstrukci kulturního dědictví v česko-bavorském příhraničí. Pro vývoj je použit programovací jazyk C# a framework Xamarin Forms. Mobilní aplikace komunikuje s webovým serverem pomocí REST API rozhraní. Posílaná data jsou ve formátu JSON. Výsledná aplikace umožňuje uživateli zobrazit zapomenuté objekty na mapě. Jako mapový podklad bylo využito Open Street Map.

Klíčová slova

Photostruk, Xamarin Forms, C#, ASP.NET, Android, iOS, REST API, OSM

Annotation

This scientific thesis is about designing and developing multiplatform mobile application for Android and iOS operating systems. The application was created as an addition for web app Photostruk. This project was designed as an analysis of historical photographs and for virtual reconstruction of cultural heritage in Czech-Bavarian borders. The coding languange C# and Xamarin Forms framework were used during development stages. Mobile app communicates with the web server using REST API interface. Data generated by the app are in JSON format. The final application enables the user to view forgotten or lost objects on the map. Open Street Map was used as underlay.

Key words

Photostruk, Xamarin Forms, C#, ASP.NET, Android, iOS, REST API, OSM

Prohlášení

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ________________ ____________________

podpis

Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Janu Feslovi, Ph.D. za cenné rady a připomínky při našich společných konzultacích. Díky patří také mé rodině a přátelům za jejich podporu při studiu.

Obsah

1 Úvod ... 6

1.1 Funkce systému ... 6

1.2 Projekt Photostruk ... 7

2 Vývoj mobilních aplikací ... 9

2.1 Nativní vývoj ... 9

2.2 Progresivní webové aplikace ... 10

2.3 Multiplatformní frameworky ... 10

2.3.1 Xamarin ... 10

2.3.2 React Native ... 10

2.3.3 Flutter ... 11

2.4 Shrnutí ... 11

3 Návrh projektu ... 12

3.1 Architektura ... 12

3.2 Návrh databáze uživatelů ... 13

3.3 Návrh aplikačního rozhraní ... 14

3.3.1 Repository layer ... 16

3.3.2 Service layer ... 17

3.3.3 Controller layer ... 18

3.3.4 Autentizace ... 19

3.4 Návrh mobilní aplikace ... 20

3.4.1 Architektura MVVM ... 20

3.5 Diagram užití ... 21

3.6 Diagram tříd ... 23

3.7 Návrh grafického rozhraní 24
4 Implementace 25
4.1 Použité technologie 25
4.1.1 ASP.NET Core 25
4.1.2 Entity Framework Core 26
4.1.3 Xamarin 28
4.1.4 Realm 30
4.2 Použité knihovny 32
4.2.1 AutoMapper 32
4.2.2 LazyCache 33
4.2.3 Newtonsoft Json 33
4.2.4 Swagger 33
4.2.5 Xamarin Essentials 34
4.2.6 MapsUI 34
4.2.7 FFImageLoading 35
4.3 Aplikační rozhraní 36
4.3.1 Validace příchozích objektů 36
4.3.2 ASP.NET Core Middleware 37
4.3.3 Načítání obrázků 39
4.3.4 Načítání dat z databáze Photostruk 40
4.4 Mobilní aplikace 40
4.4.1 Multijazyčnost 40
4.4.2 Autentizace třetích stran 41
4.4.3 Lokální databáze 41
4.4.4 Komunikace s API 42

4.4.5 Určování polohy ... 42

4.4.6 Styly ... 43

4.4.7 3D modely ... 44

4.5 Mapové podklady ... 44

4.5.1 Tvorba mapových podkladů ... 45

4.5.2 Renderování map ... 46

5 Nasazení ... 47

5.1 Virtuální stroj ... 47

5.2 Nasazení na Nginx ... 47

5.2.1 Self-signed certifikát ... 49

5.3 Nasazení na Azure ... 50

5.4 Přesun fotografií na Cloudinary ... 50

5.5 Testování mobilní aplikace ... 52

6 Závěr ... 54

Seznam literatury ... 55

Seznam obrázků ... 58

Seznam tabulek ... 59

Příloha A – Uživatelské rozhraní ... 60

Příloha B – Použitý software ... 63

Seznam použitých zkratek

Zkratka	**Význam**
OSM	Open Street Map
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
MVVM	Model-View-ViewModel
XML	Extensible Markup Language
XAML	Extensible Application Markup Language
JSON	JavaScript Object Notation
GUI	Graphical User Interface
API	Application Programming Interface
REST	Representational State Transfer
ORM	Object-Relational Mapping
CRUD	Create Read Update Delete
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics
DTO	Data Transfer Object
JWT	JSON Web Tokens
IIS	Internet Information Services
SQL	Structured Query Language
OSM	Open Street Map
LINQ	Language Integrated Query
URL	Uniform Resource Locator
SSH	Secure Shell

Zkratka	**Význam**
LTS	Long-term Support
IP	Internet Protocol
SDK	Software Development Kit
BIOS	Basic Input/Output System

1 Úvod

V dnešní době vzestupu mobilních zařízení si každý rozsáhlejší projekt zaslouží mobilní aplikaci. Tato aplikace může překonat funkce webových stránek nebo tzv. těžkých klientů.

Hlavní cíl projektu je prezentace zaniklých lokalit Šumavy díky virtuálním rekonstrukcím z historických fotografií. Projekt reaguje na potřebu posilovat společnou identitu, která je tvořena historickou provázaností obyvatelstva. Snaží se udržitelným způsobem zachovat a zhodnotit společné dědictví. Dostupnými i inovovanými nástroji pomůže zvýšit atraktivitu dotačního území prostřednictvím zachování a zhodnocení společného kulturního a přírodního dědictví.

Cílem práce je návrh a implementace mobilní aplikace, která bude doplňovat již nasazenou webovou aplikaci Photostruk. Aplikace bude načítat data z REST API, která budou posléze částečně uložena v mobilu. Server bude komunikovat s mobilní aplikací pomocí REST API. Data budou ukládána v databázi. Mobilní aplikace bude vytvořena a otestována pro operační systémy Android a iOS. Budou navrženy scénáře nasazení mobilní aplikace v prostředí různých platforem.

1.1 Funkce systému

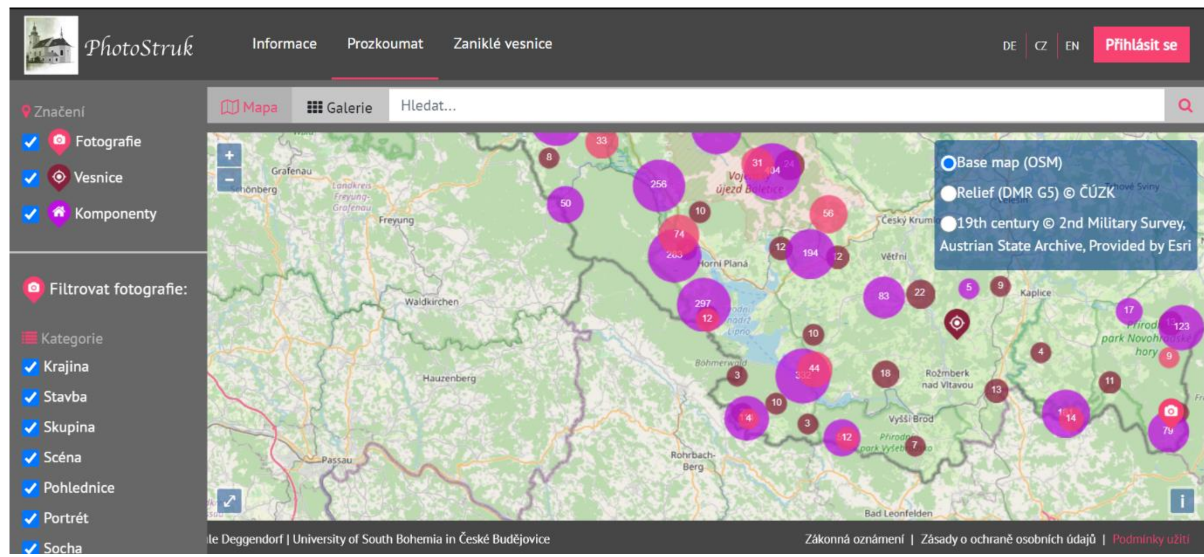
Funkce systému byly vytvořeny na základě zadání diplomové práce. Dále byly inspirovány webovou aplikací Photostruk, která již byla nasazena na začátku projektu. Kromě funkcí webové aplikace byla přidána množina funkcí specifických pro mobilní zařízení. Množina funkcí byla poté specifikována na konzultacích s Ing. Janem Feslem, Ph.D.

- Systém umožní uživateli v oblasti Bavorského lesa a Šumavy prohlížet historické informace, fotografie a rekonstruované objekty v aktuální lokalitě.
- Systém umožní uživateli pořizovat v terénu mobilním zařízením fotografie a připojovat je k dané lokalitě.
- Systém bude automaticky vyplňovat časové a lokalizační údaje k pořízeným fotografiím.

- Systém bude čerpat informace o lokalitě z databáze Photostruk.
- Systém umožní uživateli se autentizovat nebo pracovat v anonymním režimu.
- Systém umožní uživateli dočasně pracovat v offline režimu při nedostatečném signálu.
- Systém po opětovném navázání komunikace provede synchronizaci dat.
- Systém umožní tagovat a komentovat fotografie.
- Systém umožní prohlížet 3D vizualizace.

1.2 Projekt Photostruk

Inspirace pro diplomovou práci vychází z projektu PhotoStruk – Analýza historických fotografií pro virtuální rekonstrukci kulturního dědictví v česko-bavorském příhraničí. Projekt byl financován Evropskou unií v rámci Programu přeshraniční spolupráce Česká republika – Svobodný stát Bavorsko. V rámci projektu byla vytvořena databáze Photostruk, ve které se nacházejí data o zaniklých vesnicích a objektech v česko-bavorském příhraničí. Dále se v databázi nacházejí fotografie mapující tyto objekty. Jedná se o relační databázi MySQL. Databáze je nasazena na Jihočeské univerzitě a byla vytvořena Ing. Janem Feslem, Ph.D. Německá strana projektu vytvořila pro tuto databázi webovou aplikaci. Umožňuje zobrazení vesnic a objektů na mapě, také umožňuje prohlížení fotografií a také přidávání tagů a komentářů k fotografiím. Aplikace byla postavena na jazyce PHP a frameworku Laravel. Bavorská univerzita také vytvořila pro svoje účely vlastní databázi založenou na PostgreSQL. Data se mezi databázemi pravidelně synchronizují jedenkrát denně. Data jsou nejprve vyexportována do XML souborů. Ty jsou poté přístupné přes File Transfer Protocol (FTP). Německá strana poté provede import souborů do vlastní databáze. Jako podklad pro zobrazení map byl použit Open Street Map. Více informací o projektu se nachází na webu *https://photostruk.cz*.



Obrázek 1 - Webová aplikace Photostruk, převzato z [1]

2 Vývoj mobilních aplikací

Poptávka po mobilních aplikacích stále roste. Jako následek toho vznikají nové možnosti vývoje v tomto odvětví. Vývoj se majoritně soustředí na dva nejpopulárnější mobilní operační systémy Android a iOS. Výběr správné technologie, může ovlivnit celkovou kvalitu produktu. Je nutné vzít v potaz externí faktory mimo vývoj softwaru, které projekt ovlivňují. Nejčastějšími jsou:

- Čas určený pro vývoj
- Cena vývoje aplikace
- Přijetí nových zaměstnanců
- Cena údržby aplikace
- Výkon aplikace
- Podpora technologie ze strany vývojářů
- Podpora knihoven třetích stran

2.1 Nativní vývoj

Vývoj v nativním prostředí probíhá pomocí nástrojů podporovaných samotnými vývojáři operačních systémů iOS a Android. Vývoj pro operační systém iOS je uskutečněn v programovacích jazycích Objective C nebo Swift. Oba jazyky jsou udržovány pod pevnou správou společnosti Apple. Android používá pro vývoj programovací jazyky Java a stále populárnější jazyk Kotlin, který je plně kompatibilní s jazykem Java. Nativní aplikace mohou být považovány za nejvýkonnější. Vše běží na nativních komponentách určených pro danou platformu. Vývojářské nástroje počítají s plnou podporou. Bohužel nativní vývoj pro každý systém samostatně může být pro malé týmy časově náročný a obtížnější z důvodu nutnosti znát obě technologie. Tento postup zvyšuje celkovou cenu vývoje a údržby aplikace.

2.2 Progresivní webové aplikace

Jedná se o webovou stránku, která běží lokálně v mobilním telefonu klienta [2]. Vývoj probíhá podobným způsobem jako při tvorbě webových aplikací. Velká výhoda spočívá ve vývojářských nástrojích, které jsou pro vývojáře mobilních aplikací známé nebo jednoduše naučitelné. Existují nástroje pro nejpopulárnější webové frameworky jako jsou Angular nebo React [3]. Další výhoda spočívá v možnosti instalace aplikaci pomocí webového prohlížeče. Toto umožňuje distribuci aplikace bez použití oficiálního obchodu pro danou platformu. Díky tomu, že se stále jedná o webovou aplikaci je výkon o něco nižší než u aplikací nativních. Jedním z hlavní nedostatků progresivních webových aplikací je omezený přístup ke kompletním funkcím mobilního telefonu.

2.3 Multiplatformní frameworky

Jedná se o frameworky umožňující vytvářet kompletně nativní aplikace. Umožňují přístup ke kompletním funkcím telefonu a dovolují vývojáři psát kód ve sdíleném programovacím jazyku. Nejpopulárnější z nich jsou popsány níže.

2.3.1 Xamarin

Jedná se o open-sourcový framework vyvíjený společností Microsoft [4]. Build aplikace probíhá pomocí Visual Studia. Programovací jazyk používaný pro vývoj je C# a framework .NET. Pro UI jsou využity nativní komponenty. Výkon aplikace je na srovnatelné úrovni s vývojem nativních aplikací.

2.3.2 React Native

React Native je multiplatformní framework od společnosti Facebook [5]. Pro vývoj aplikace se používá programovací jazyk javascript. Je založený na webovém frameworku React. Obsahuje nativní komponenty. Výkon je na podobné úrovni jako u frameworku Xamarin.

2.3.3 Flutter

Poslední multiplatformní framework se nazývá Flutter [6]. Byl vyvinut Google a jedná se o open-source. Hlavní rozdíl oproti předchozím dvěma je ve vlastním vykreslování komponent. Díky tomu jeho výhoda spočívá ve větším výkonu aplikace. Nevýhodou je, pokud dojde ke změně UI od strany Androidu nebo iOS budou se muset této změně přizpůsobit i jejich grafické knihovny. Dále se jedná o poměrně novou technologii a uživatelská základna ve formě komunitních knihoven není tak velká jako u předchozích. Vývoj pro frameworku Flutter probíhá také v poměrně novém programovacím jazyku Dart vyvíjeným Googlem.

2.4 Shrnutí

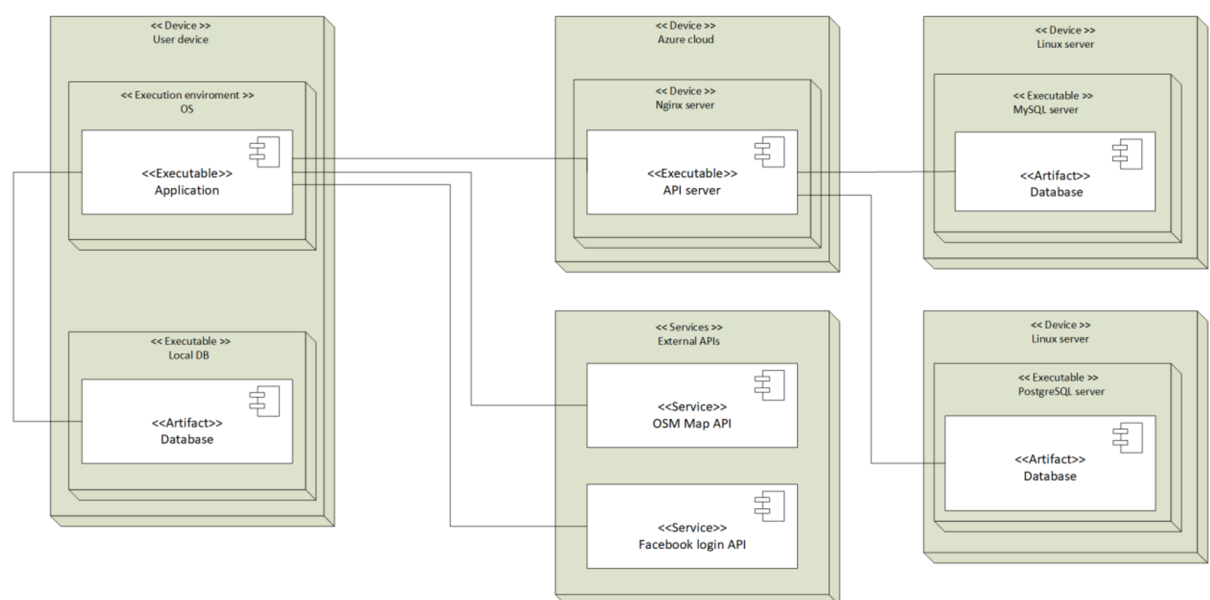
Při vývoji mobilní aplikace je nutné přesně definovat účel, pro jaký bude sloužit. Dále je nutné brát v potaz množství peněz a času má aplikace zabrat. Další podmínkou je znalost a schopnost vývojářského týmu. V diplomové práci bude probíhat vývoj aplikace jedním programátorem. Z tohoto důvodu odpadá tvorba aplikace v nativním prostředí. Dále by nebylo vhodné vést vývoj pomocí progresivních webových aplikací. Především kvůli vysokým nárokům na výkon aplikace a potřebě kompletního přístupu k funkcím mobilního telefonu. Jako nejlepší možné řešení bude tedy zvoleno použití multiplatformního frameworku. Flutter nebyl zvolen z důvodu poměrně nové a nerozvinuté technologie. Také nutnosti naučit se kompletně nový programovací jazyk Dart. Nakonec byl tedy zvolen framework Xamarin na základě programovacího jazyka C# a technologiím od Microsoftu.

3 Návrh projektu

V této kapitole bude probrán návrh architektury. Dále bude popsán návrh architektury aplikačního rozhraní a mobilní aplikace. Následně bude představen prvotní návrh grafického rozhraní.

3.1 Architektura

Architektura vychází ze zadání diplomové práce. Dále byl doplněn o informace z konzultací s vedoucím práce Ing. Janem Feslem, Ph.D. Architektura systému je založena na způsobu klient-server [7]. V této architektuře server poskytuje své zdroje jednomu nebo více klientům. Server nerozlišuje, o jaký druh zařízení se v klientské části jedná. Komunikace mezi zařízeními je prováděna pomocí smluveného protokolu a data jsou přenášena ve strukturovaném formátu. V diplomové práci bude použit pro přenos formát JSON. Jeho hlavní výhodou je snadná přehlednost posílaných dat a jednoduchý zápis. JSON je v moderních aplikacích považován za komunikační standard.



Obrázek 2 - Návrh architektury

V zadání práce je uvedeno pouze vytvoření mobilní aplikace. Zajištění aplikačního rozhraní měla zajistit německá strana. K tomu nedošlo a aplikační rozhraní muselo být vytvořeno autorem práce. Návrh je tvořen dvěma databázemi.

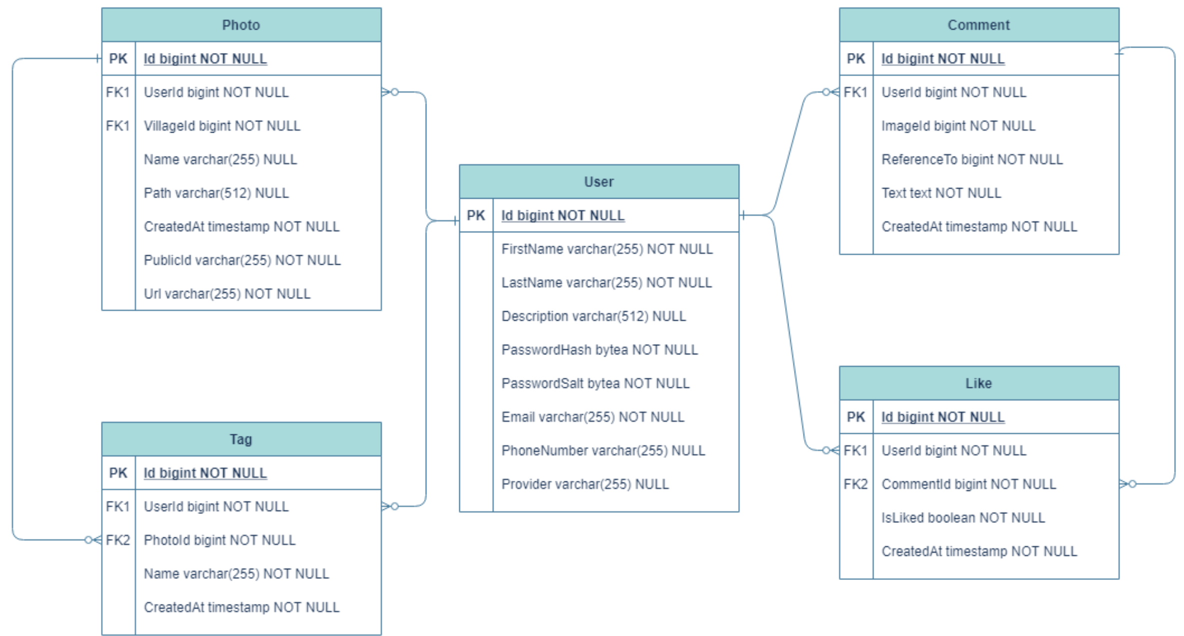
1. Databáze PhotoStruk – slouží pro ukládání dat o zaniklých objektech. Databáze byla již připravena předem. Data bude možno pouze čerpat. Úpravy nebudou povoleny z klientské aplikace.
2. Uživatelská databáze – slouží pro ukládání uživatelů, tagů, komentářů a lajků. Databáze bude vytvořena.

Tyto dvě databáze jsou propojené s aplikačním rozhraním viz obrázek 1. Hlavní úloha aplikačního serveru bude v získávání dat, uspořádání a zasílání výsledků na klientskou část. Dále bude zajištovat autentizaci uživatele. Po úspěšném přihlášení bude moci uživatel přidávat a upravovat data v uživatelské databázi. Mobilní aplikace se bude starat o zobrazení dat uživateli. Část dat bude trvale uložená v lokální databázi, aby se předešlo špatnému zobrazí při ztrátě internetového připojení v odlehlých oblastech. Data budou pravidelně synchronizována s databázemi přes aplikační rozhraní. Mobilní aplikace bude využívat komunikace s externími API.

1. API pro načítání map z Open Street Map serverů
2. API pro přihlášení přes Facebook

3.2 Návrh databáze uživatelů

Databáze bude sloužit k uložení dat o uživateli. Návrh je popsán entitně-relačním diagramem [8]. Diagram slouží pro zobrazení schéma databáze. Plné využití ER diagramu by mělo vést k vysoké kvalitě designu, managementu a údržby databáze. Klíčová databázová entita se jmenuje *User.* Obsahuje informace o registrovaném uživateli. Dále obsahuje přístupové heslo uživatele do aplikace. Heslo je do databáze uloženo ve formě hash. Do databáze je také přidána sůl pro zmatení potenciálního útočníka. Zbytek schématu zabírají entity *Photo, Tag, Comment* a *Like.* Budou obsahovat informace, kterými bude moci uživatel doplňovat dodatečné informace k fotografiím. Schéma databáze je vidět na obrázku níže.



Obrázek 3 - ER diagram databáze uživatelů

3.3 Návrh aplikačního rozhraní

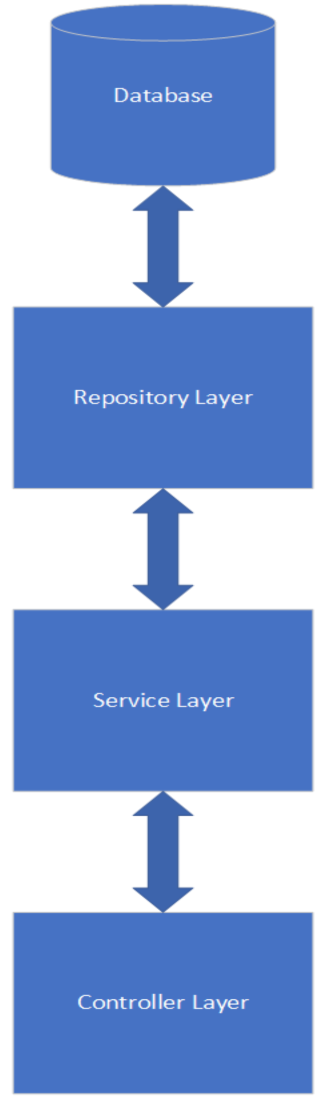
Application programming interface (API) slouží ke spojení databází s klientskou částí. Server bude postaven jako REST API. Webová služba REST dovoluje klientovi přistupovat a manipulovat se zdroji serveru. Bude sloužit hlavně pro načítání dat z PhotoStruk databáze. Aplikační rozhraní bude dále řídit autentizaci uživatelů u služeb, u kterých to bude vyžadováno. Dalším úkolem bude ukládat a načítat fotografie. V následující tabulce je uvedeno, jaké operace server bude umožňovat.

URL	**HTTP metoda**	**Obsah dotazu**
/areal/{cultureInfo}	GET	cultureInfo
/areal/images	GET	
/auth/login	POST	loginDto
/auth/register	POST	registerDto
/auth/changePassword	POST	recoverPasswordDto
/comment	GET	
/comment/image/{imageId}	GET	imageId
/comment/{commentId}	GET	commentId
/comment	POST	commentInputDto
/comment/{commentId}	PUT	commentId, commentInputDto

/comment/{commentId}	DELETE	commentId
/component/{cultureInfo}	GET	cultureInfo
/component/images	GET	
/image/{imageId}	GET	imageId
/like/{commentId}	GET	commentId
/like	POST	likeInputDto
/like/{likeId}	PUT	likeId, likeInputDto
/photo	GET	
/photo/{photoId}	GET	photoId
/tag/{imageId}	GET	imageId
/tag	POST	tagInputDto
/user	GET	
/user/{userId}	GET	userId
/village/{cultureInfo}	GET	cultureInfo
/village/images	GET	

Tabulka 1 - REST API serveru

Architektura serveru bude rozdělena do několika vrstev. Rozdělením do vrstev získáváme kromě přehlednosti také možnost server lépe testovat. Každá vrstva slouží ke konkrétnímu účelu. Tato metoda také přidává možnost jednotlivé vrstvy přepracovat nebo nahradit, aniž by tím byl ovlivněn zbytek vrstev. Architektura bude rozdělena do tří na sobě nezávislých vrstev.



Obrázek 4 - Vrstvy aplikačního rozhraní

3.3.1 Repository layer

Repository layer je vývojový vzor přístupu k datům [9]. Data jsou z databáze předávána do objektu repository a ten řídí operace CRUD. Jedná se o základní databázové operace vytvoření, čtení, aktualizace a smazání. Repository dále centralizuje běžný přístup k datům a zajišťuje lepší udržitelnost. Také odděluje přístup k databázi od vrstvy doménových modelů. V API bude použit Object-relational mapping (ORM) framework, který implementování repository vrstvy výrazně simplifikuje. ORM je technika pro převod dat mezi dvěma nekompatibilními systémy. V následujícím kódu je vidět implementace metody, která vrací

všechna data z tabulky odpovídající entitě *TModel*. Klíčové slovo *virtual* značí v programovacím jazyce C#, že metoda bude moci být přepsána v poděděné třídě. Rozhraní *IQueryable* slouží k uchování dat získaných pomocí dotazů z databáze. Na takto uložený objekt lze použít dodatečné filtrování pomocí Language Integrated Query (LINQ) [10]. Jedná se o speciální syntaxi jazyka C# uzpůsobenou pro jednodušší práci s daty. Obyčejně je programátor nucen naučit se nové techniky pro přístup k rozdílným datovým zdrojům. LINQ tento problém řeší sjednocením syntaxe do jednoho společného jazyka. Pomocí příkazu *db.Set<TModel>* jsou vybrána všechna data z databáze třídy typu *TModel.*

```
public virtual IQueryable<TModel> GetAll()
        {
            return db.Set<TModel>();
        }
```

3.3.2 Service layer

V servisní vrstvě se nachází logika aplikace. Bude se starat o mapování modelů do Data Transfer Object (DTO) objektů posílaných na klienta. Dále se bude starat o načítání a ukládání souborů z disku. Také zde bude implementována logika autentizace. Společná logická část bude sdružena do abstraktních předků, u kterých bude využito genericity. Jedná se o možnost specifikovat datový typ až v momentě vytvoření instance. Ve třídě se poté pracuje s generickým typem. Generický typ je zapsán uvnitř špičatých závorek.

V následujícím kódu je vyobrazena metoda generického předka pro získání všech záznamů dané entity. Uvnitř metody je volána instanční proměnná mapper. Tato třída se stará o přemapování objektů z databáze na DTO objekty. V komerčním prostředí se jedná o běžnou praxi. Ve většině případech nechceme uživateli poslat celou databázovou tabulku, ale jenom její část. Ve výsledku se jedná o řešení zvyšující bezpečnost. Výsledný typ je posílán v rozhraní *IEnumerable,* ve které je generická třída uložena jako list.

```
public virtual IEnumerable<TListDto> GetAll()
        {
            return this.mapper.Map<IEnumerable<TListDto>>
(this.repository.GetAll());
        }
```

3.3.3 Controller layer

V této vrstvě budou vytvářeny endpointy viditelné ve webové stránce. Kontroléry slouží k ověření autentizace uživatele. Základní ověření dosáhneme přidáním atributu *[Autorize]* nad příslušný endpoint. Dále provádí validaci vstupních objektů. Tato vrstva u některých endpointů bude cachovat výsledky ze servisní vrstvy. V následujícím kódu je vidět příklad přidání komentáře.

```
[HttpPost]
        public async Task<CommentDto> Post([FromBody]
CommentInputDto comment)
        {
            return await
this.service.Add(comment).ConfigureAwait(false);
        }
```

Atribut *[HttpPost]* určuje, o jakou metodu HTTP dotazu se jedná. Identifikátor *async* říká, že metoda se bude provádět asynchronně. V metodě označené tímto identifikátorem se může nacházet i klíčové slovo *await*, které určuje, že aplikace bude vyčkávat na příslušné řádce kódu, dokud asynchronní metoda nebude vykonána. Pokud je v asynchronní metodě vrácen nějaký objekt, návratový typ musí být uzavřen třídou *Task*. Jedná se o generickou třídu určenou pro zpracování vláken. V uvedeném příkladu vrací generický typ *CommentDto*. Jako parametr metody je posílán objekt *CommentInputDto*. Při přijetí dotazu je automaticky provedena validace objektu. Atribut *[FromBody]* značí, že objekt bude poslán v těle dotazu. Objekt je poté předán jako parametr do metody *this.service.Add(comment)*, ve které je implementována logická část pro přidání nového komentáře. Metoda *ConfigureAwait(false)*

nakonfiguruje úlohu, aby pokračování po *await* nemuselo být spuštěnu v kontextu volajícího. Použitím této metody je zabráněno potenciálnímu deadlocku.

3.3.4 Autentizace

Ve webových aplikacích existuje mnoho variant způsobu autentizace. V této práci byla využita autentizace pomocí JSON Web Tokens (JWT) [11]. Jedná se o internetový standard založený na vytváření internetových tokenů ve formátu JSON. Token se skládá ze tří částí:

1. Hlavička – Obsahuje zkratku algoritmu, ve kterém je generován podpis a typ tokenu.
2. Payload – Nachází se zde doplňující informace o tokenu. Jedná se například o práva, jméno uživatele atd. Také se zde nachází časová známka platnosti tokenu.
3. Podpis – Kontroluje token. Podpis je vytvořen spočtením hlavičky s payload a zakódováním do Base64. Poté je zakódovaný text zašifrován pomocí algoritmu uvedeném v hlavičce tokenu.

Pokud se chce uživatel autentizovat, musí nejprve odeslat na server správné přihlašovací údaje. Po ověření těchto údajů, server vygeneruje nový JWT token a odešle ho klientovi. Přes tento token může uživatel přistupovat ke zdrojům na serveru podle toho, jaká mu byla přidělena práva. Token se zasílá v hlavičce. Po vypršení platnosti tokenu server odešle na klienta chybovou hlášku. Pokud uživatel bude chtít v relaci pokračovat, musí se znovu autentizovat. Tato metoda autentizace má několik nevýhod. Kdyby došlo k zachycení tokenu útočníkem, je schopný používat aplikaci pod právy uživatele, dokud platnost tokenu nevyprší. Další problém vzniká při odhlašování. Standardně probíhá smazáním tokenu z klientské části aplikace. Přesto token zůstává platný, dokud jeho platnost nevyprší. Z tohoto důvodu může být kýmkoliv využit. V kódu níže je vidět ukázka generování tokenu.

```
var secretKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes

(configuration.GetSection("JwtSettings:Secret").Value));
            var signinCredentials = new
SigningCredentials(secretKey, SecurityAlgorithms.HmacSha256);
```

```
            var tokeOptions = new JwtSecurityToken(
                claims: new List<Claim> {
                    new Claim(Const.Role, "User"),
                    new Claim(ClaimTypes.Name,
user.Id.ToString()),
                },
                expires: DateTime.Now.AddMinutes(60),
                signingCredentials: signinCredentials
            );
```

Nejprve dojde k načtení tajného klíče do proměnné *secretKey*. Poté je pomocí konstruktoru *SigningCredentials* vytvořen podpis tokenu. Hash bude vytvořen pomocí algoritmu HmacSha256. Nakonec je vytvořen konstruktorem *JwtSecurityToken* výsledný token.

3.4 Návrh mobilní aplikace

Mobilní aplikace bude sloužit pro zobrazování informací z databáze Photostruk. Metadata z databáze budou uložena v lokální databázi. Uživatel je bude moci na mapě prohlížet a stahovat k nim příslušné fotografie. Dále bude moci na server posílat data vlastní. V návrhu byly brány v potaz aktuální trendy používané při vývoji mobilních aplikací.

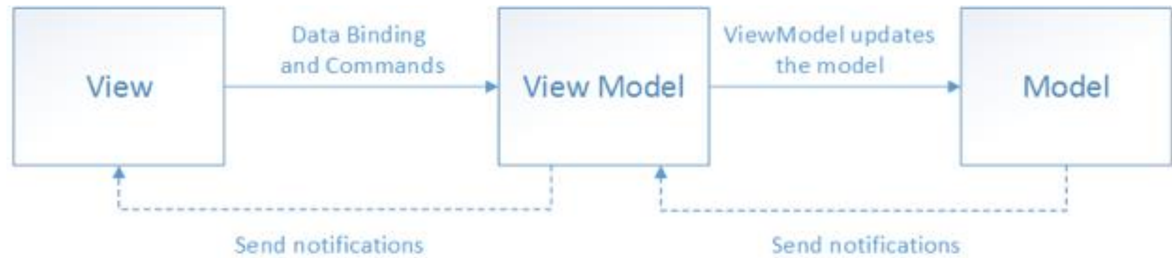
3.4.1 Architektura MVVM

Mobilní aplikace je postavena na návrhovém vzoru Model View View-Model (MVVM) [12]. Tato architektura se stala již standardem pro vývoj mobilních aplikací a každá nově napsaná aplikace by se měla tohoto návrhového vzoru držet. Jedná se o řešení, při kterém se oddělí bussiness a prezentační logika aplikace od uživatelského rozhraní. Na základě těchto principů vzniká kód lépe udržitelný, rozšiřitelný a testovatelný. Návrhový vzor se skládá ze tří vrstev:

- Model – popisuje data, se kterými aplikace pracuje. Nachází se zde také bussiness logika aplikace.
- View – reprezentuje uživatelské rozhraní aplikace

- View-Model – nachází se zde prezentační logika aplikace. Propojuje Model a View.

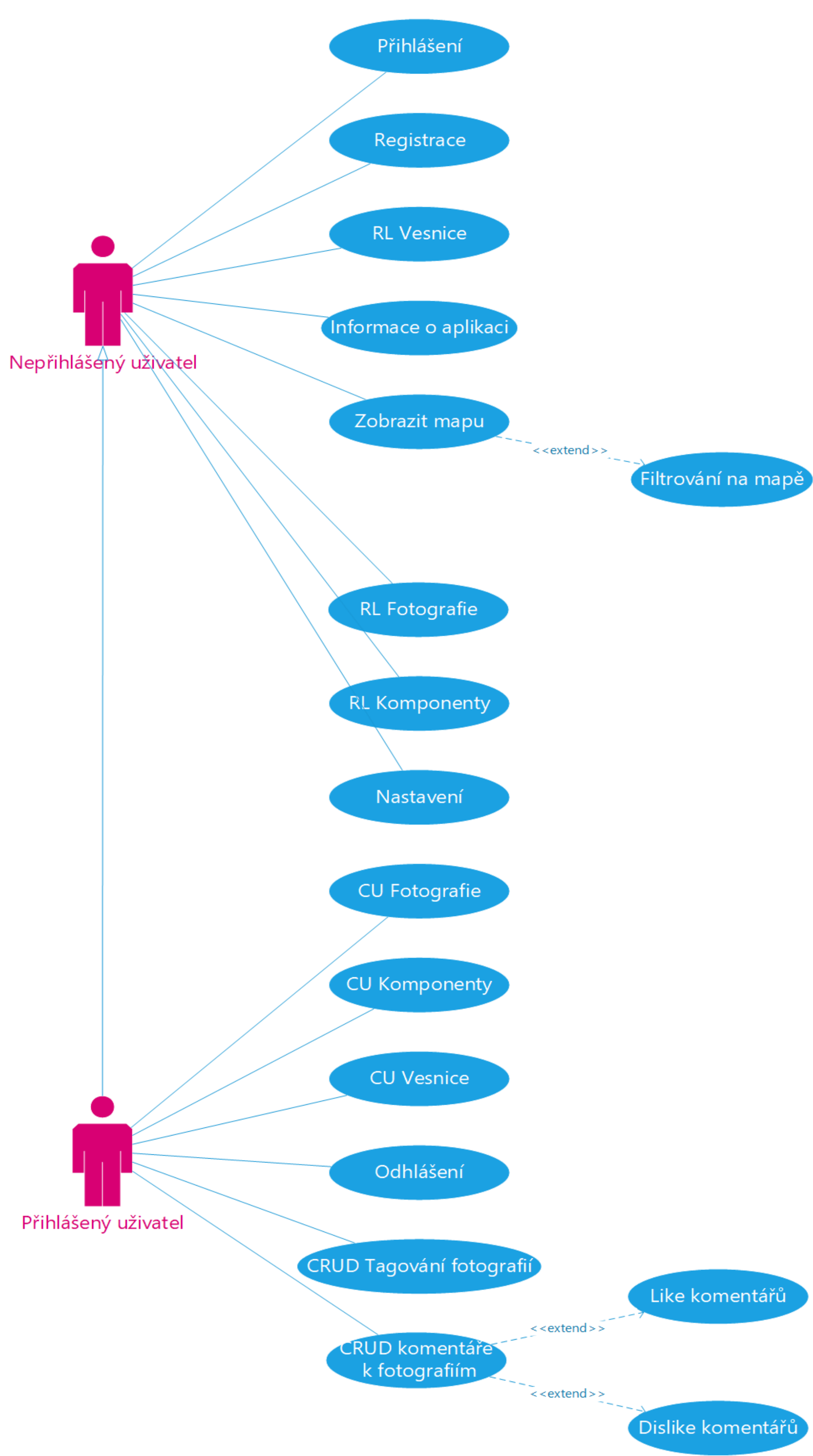
MVVM podporuje automatický update komponent mezi View a ViewModel pomocí metody DataBinding.



Obrázek 5 - MVVM diagram, převzato z [12]

3.5 Diagram užití

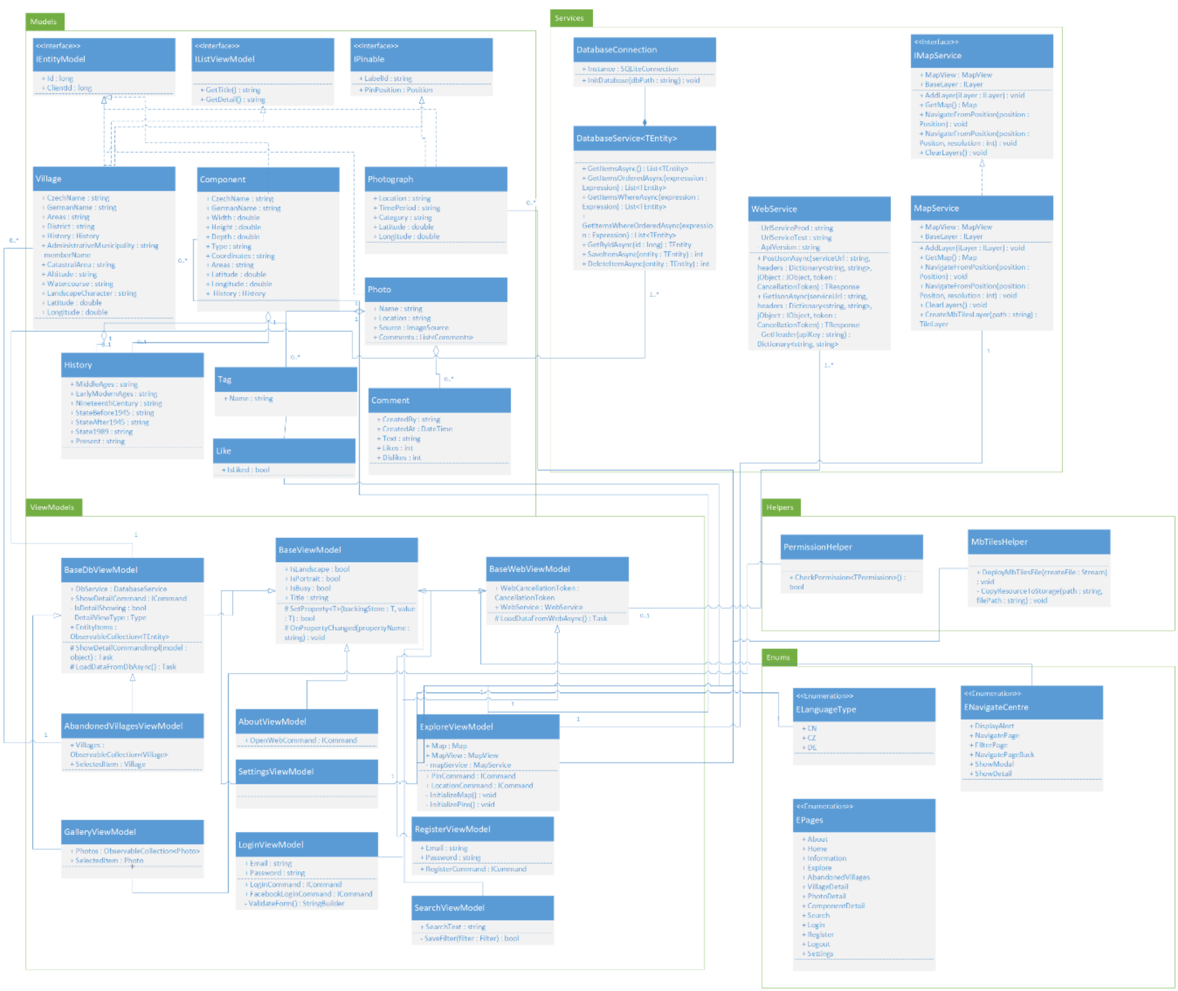
Účel diagramu užití je zachytit vnější pohled na modelovaný systém [13]. Pomáhá odhalit hranice sytému a slouží jako podklad pro odhad rozsahu systému. Měl by obsahovat všechny funkcionality, které by měl daný systém umět. Funkcionalita v sobě může mít návaznost na další akce. S jednotlivými případy užití komunikuje aktér, který zastává nějakou roli v systému. Roli může zaujmout uživatel nebo externí systém. Tento systém rozlišuje pouze role přihlášeného a nepřihlášeného uživatele.



Obrázek 6 - Diagram užití

3.6 Diagram tříd

Diagram tříd popisuje statickou strukturu systému, znázorňuje datové struktury a operace u objektů a souvislosti mezi objekty [14]. Využívá se pro běžné koncepční modelování, ale i pro detailní modelování nebo převod modelů do programového kódu. Obsahuje třídy systému a jejich atributy a metody. Také znázorňuje vztahy mezi jednotlivými objekty. Diagram je platformě závislý. Je připraven pro programovací jazyk C# a framework Xamarin Forms.



Obrázek 7 - Diagram tříd

3.7 Návrh grafického rozhraní

Sleduje aktuální trendy v oblasti návrhu grafického desingu. Při tvorbě bude cíleno hlavně na jednoduchost a intuitivnost. Budou použity koncepty pro uživatele známé z jiných aplikací. Design aplikace bude inspirován Material Desingem od společnosti Google. Grafický návrh uživatelského rozhraní se nalézá v příloze A.

Navigace v aplikaci bude probíhat pomocí dolního menu. Bude se skládat ze čtyř položek:

- **Informace** – Obrazovka se bude skládat z obrázků a informacích o projektu Photostruk.
- **Prozkoumat** – Obrazovka bude obsahovat mapu s objekty. Po kliknutí na pin se zobrazí náhled vybraného objektu. Z náhledu půjde přejít na jeho detail. Dále se bude na obrazovce nalézat tlačítko pro filtrování objektů na mapě.
- **Vesnice** – Obrazovka bude obsahovat list seznamu vesnic. Po označení položky bude možné se dostat na její detail.
- **Nastavení** – Obrazovka bude obsahovat položky, které půjdou v aplikaci nastavit, informace o uživateli a možnost odhlášení.

4 Implementace

Tato kapitola se zabývá implementací mobilní aplikace a aplikačního rozhraní. Technologie a knihovny, které byly využity při tvorbě aplikací jsou popsány v části první. Dále je popsána tvorba serverové části a také mobilní aplikace. Nakonec je charakterizován způsob vykreslování map v mobilní aplikaci a postup při renderování mapy vlastní.

4.1 Použité technologie

Tato část práce se zabývá představením jednotlivých technologií, které byly použity na vypracování praktické části. Technologie byly na vybrány na základě zkušeností autora a konzultací s vedoucím práce. Systém byl napsán ve vývojovém prostředí Visual Studio od společnosti Microsoft.

4.1.1 ASP.NET Core

Pro vývoj serverové části aplikace byl vybrán open-sourcový scriptovací framework ASP.NET Core [15]. Tento framework je vyvíjen společností Microsoft a je psán v programovacím jazyku C#. Vychází z dnes již skoro nepoužívaného frameworku ASP.NET [16]. Jedna z jeho hlavních výhod je jeho multiplatformost. Framework podporuje server IIS od firmy Microsoft i linuxové servery Nginx a Apache.

Server je spuštěn ve třídě *Program*.

```
Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
```

Zde se zavolá konvenčně pojmenovaná třída *Starup* skládající se z dvou metod *ConfigureServices* a *Configure.* V této třídě při spouštění serveru proběhnou veškerá nastavení. Nejprve bude spuštěna metoda *ConfigureServices*. Bude zde provedeno zaregistrování služeb, které mohou být poté vloženy do třid pomocí techniky dependency

injection. Posloupnost přidávání služeb v této třídě není závislá na pořadí. Metoda *Configure* specifikuje, jak bude aplikace odpovídat na dotaz od klienta. Každá přidaná komponenta vytváří novou mezivrstvu. Přidání komponent je zde závislé na pořadí. V kódu je vidět komponenta pro mapování kontrolérů.

```
app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
```

4.1.2 Entity Framework Core

V objektovém programování, které se snaží co nejvěrněji napodobit realitu, vzniká problém pracovat s daty z relační databáze. Tato rozdílnost mezi přístupem k datům vedla k vývoji frameworků zabývajících se objektově relačním mapování. ORM se stará o konverzi dat mezi relační databází a objekty. Programátor je odstíněn od přímé práce s databází a od používání jazyka SQL. Nemusí se zabývat nejednotností mezi jednotlivými verzemi jazyka SQL v relačních databázích. ORM celkově urychluje vývoj a srozumitelnost aplikací. Nevýhodou je nižší efektivnost dotazů do databáze.

Entity Framework Core využívá dva přístupy spojení s databází Database First a Code First. První způsob dovoluje využít již existující databázi a pomocí nástroje scaffolding vytvořit třídy, které budou využity v aplikaci. Druhá možnost využívá opačný přístup. Nejprve jsou vytvořeny třídy, ze kterých se vytvoří SQL příkaz, který upraví vybranou databázi. V diplomové práci byly využity oba přístupy.

Nejprve byl ve třídě *Startup* implementován databázový kontext, který zajištuje spojení do databáze.

```
services.AddDbContext<DataContext>(o =>
            o.UseNpgsql(Configuration
.GetConnectionString("DefaultConnection"),
            b =>
b.MigrationsAssembly("PhotostrukBE.WebAPI")));
```

V souboru *appsettings.json* jsou uložena data o nastavení serveru. Nachází se zde také *ConnectionString*, ve kterém je uložena adresa, port, název a přihlašovací údaje k databázi. ASP.NET Core rozlišuje dva typy souborů: *appsetting.json* a *appsetting.Development.json*, jeden pro vývoj a druhý pro nasazení. Toto zajišťuje vlastní nastavení pro každé vývojové prostředí.

4.1.2.1 Entity

Při přístupu Database First bylo nejprve nutné vytvořit třídy, které se poté konvertují na databázové tabulky.

```
public class Photo : BaseModel
    {
        public string Name { get; set; }
        public string Path { get; set; }
        public DateTimeOffset CreatedAt { get; set; }
        public long UserId { get; set; }
        public virtual User User { get; set; }
    }
```

V kódu můžete vidět třídu *Photo*, která představuje model pro Entity Framework. Ve třídě jsou znázorněné jednotlivé atributy, které bude databázová tabulka obsahovat. Také obsahuje třídu *User*. Tato třída znázorňuje referenci mezi třídami. Integritu dat zajišťuje třída *DataContext.* Tato třída obsahuje všechny entity existující v databázi a nastavení jejich atributů. V proměnné *modelBuilder* je vybrána entita *Photo*, která je konfigurována. Postupně jsou upravovány omezení pro atributy této entity. Na závěr je provedena konfigurace cizího klíče.

```
modelBuilder.Entity<Photo>(entity =>
            {
                entity.Property(e =>
e.Name).HasMaxLength(255);

                entity.Property(e =>
e.Path).HasMaxLength(512);
```

```
                entity.Property(e => e.CreatedAt)
                .HasColumnType("timestamp with time zone")
                .IsRequired();

                entity.HasOne(d => d.User)
                    .WithMany(p => p.Photo)
                    .HasForeignKey(d => d.UserId)
                    .HasConstraintName("fk_user");
            });
```

Poté, co byly vytvořeny třídy a kontext databáze, bylo nutné provést migraci databáze. Bez této operace by se změny provedené v aplikaci nepromítly v databázi. Migrace databáze bude provedena následujícími příkazy.

```
Add-Migration Init
Update-Database
```

4.1.3 Xamarin

Xamarin je open-sourcový framework vyvíjený společností Microsoft [4]. Byl vybrán na základě druhé kapitoly diplomové práce a zkušeností autora s touto technologií. Zde jsou popsány nejdůležitější vlastnosti frameworku Xamarin.

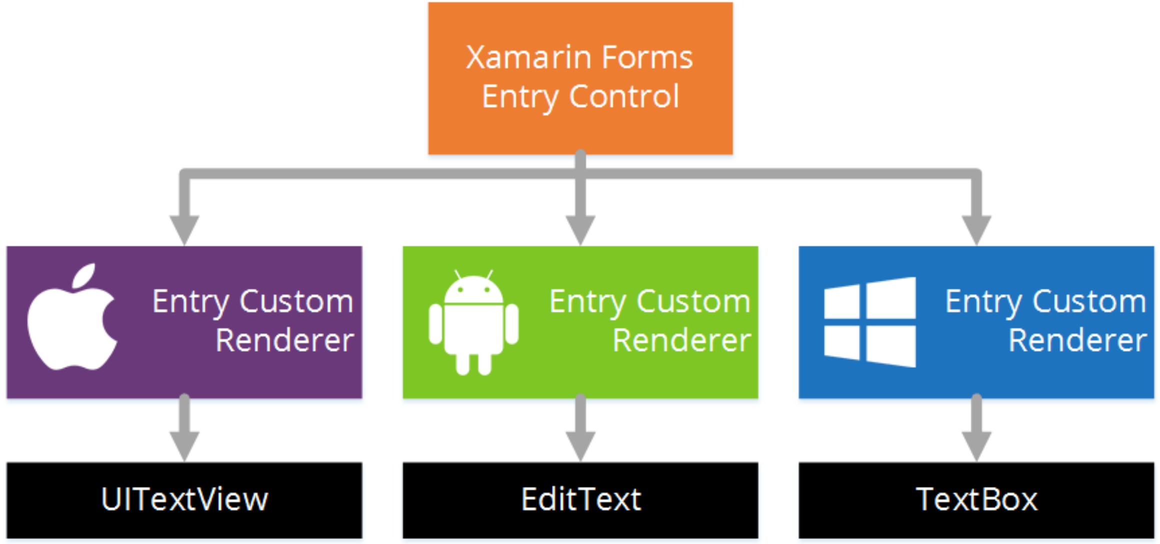
4.1.3.1 Xaml

Grafické rozhraní je v Xamarin aplikacích tvořeno jazykem Extensible Application Markup Language (XAML) [4]. Jedná se o jednoduchý a přesně strukturovaný jazyk založený na XML. Soubory způsobem zápisu připomínají validní XML soubor. Jedná se o hierarchickou strukturu připomínající strom. V následujícím kódu je vidět základní zápis stránky v XAML jazyku.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="PhotoStruk.Views.Page1">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Welcome to Xamarin.Forms!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

4.1.3.2 Vlastní vykreslovače

Vlastní vykreslovače umožňují přístup pro úpravu vzhledu a chování komponent používaných v Xamarin Forms [4]. Mohou být použity pro drobné úpravy vzhledu až po komplexní změny v logice komponenty. Xamarin Forms využívá pro zobrazení UI nativních komponent. Z toho vyplývá, že vzhled aplikace se liší podle toho, na jakém operačním systému je spuštěn. Pro dosažení úplné kontinuity UI mezi operačními systémy je nutné vytvořit vlastní vykreslovač.



Obrázek 8 - Hierarchie vlastních vykreslovačů, převzato z [4]

Tvorba vykreslovačů se skládá ze tří kroků:

1. Podědění třídy vykreslovače komponenty, která bude vizuálně změněna
2. Přepsání metody *OnElementChanged*, která se stará o vykreslení a logiku vybrané komponenty
3. Přidání atributu *ExportRenderer* nad název třídy pro upřesnění, že vykreslovač bude využit jako komponenta Xamarin Forms

4.1.3.3 Data binding

Technika slouží ke svázání atributů ze dvou rozdílných tříd. Změny jednoho atributu se automaticky reflektují do druhého. Data binding je důležitá součást architektury MVVM [4]. Vždy dochází k provázání objektů mezi view a view-model. V aplikaci se nejprve nadefinuje atribut ve ViewModelu.

```
string title = string.Empty;
public string Title
{
    get { return title; }
    set { SetProperty(ref title, value); }
}
```

Třída, která využívá data binding, musí implementovat rozhraní *INotifyPropertyChanged.* Deklarace atributu *Title* probíhá standardně podle konvencí C#. Hlavní rozdíl spočívá v metodě *set*. Pokud má být atribut svázaný s jiným, musí být *value* přidáváno přes metodu *SetProperty.* Pokud má View obsahovat nabindované atributy, musí nejprve načíst *BindingContext* s ViewModelem, který bude provázán. Poté je atribut zavolán pomocí následujícího kódu.

```
Title="{Binding Title}"
```

4.1.4 Realm

Realm je objektová databáze používaná v mobilních aplikacích [17]. Je vytvářena při startu aplikace pomocí definovaných tříd.

```
public class Village : RealmObject, IEntityModel
    {
        [PrimaryKey]
        public long Id { get; set; }
        public string NameCzech { get; set; }
        public string NameGerman { get; set; }
        public DateTimeOffset? PeriodStart { get; set; }
        public DateTimeOffset? PeriodEnd { get; set; }
        public double? Latitude { get; set; }
        public double? Longitude { get; set; }
        public History History { get; set; }
    }
```

Každá třída, kterou chceme použít jako databázovou entitu, musí být poděděna ze třídy *RealmObject* [17]. Atributy entity odpovídají typům proměnných v jazyce C#. Proměnné končící otazníkem vyjadřují možnost nabývat hodnot *null*. Pokud má být proměnná použita jako primární klíč, je nad ní použit atribut *[PrimaryKey]*. Primární klíč jednoznačně identifikuje určitou instanci databázové entity. Dále musí dodržovat dvě základní vlastnosti:

- Jedinečnost
- Nesmí nabývat nulové hodnoty

Databáze Realm je schopna vytvářet vztahy mezi entitami 1-1 nebo 1-N. Implementace vazby je znázorněna v kódu výše třídou *History*. Zápis do databáze musí být proveden vždy pomocí transakce, jak je vidět v kódu níže. V instanční proměnné *_database* je zavolána metoda *Write*. Tento zápis určuje, že následují blok kódu bude vykonán jako transakce. Transakce je skupina příkazů, které převedou databázi z jednoho konzistentního stavu do druhého. Po úspěšném provedení všech příkazů v bloku bude transakce automaticky potvrzena. Kdyby došlo uvnitř transakce k chybě, všechny změny budou vráceny do původního stavu. V bloku transakce je vidět příkaz *foreach,* který projde pole *entities* a provede příkazy uvnitř bloku kódu. Zde je provede příkaz pro uložení položky do databáze *_database.Add(item, true).* Parametr *item* označuje entitu, která bude uložena. Parametr true určuje jestli má být provedena aktualizace pokud zvolená entita v databázi již existuje.

```
_database.Write(() =>
            {
                foreach (var item in entities)
                {
                    _database.Add(item, true);
                }
            });
```

Nevýhodou Realm databáze je její přístup k vláknům. Pokud bude instance databáze volána z jiného vlákna, než ve kterém byla data načtena, bude vyvolána výjimka pro čtení ze špatného vlákna. Aby se předešlo těmto chybám, je nutné pro asynchronní metody vytvářet dočasně novou instanci databáze a operace pro čtení dat provádět z tohoto vlákna.

4.2 Použité knihovny

Aplikace využívají celou řadu knihoven. V následující časti jsou popsány ty nejdůležitější. Instalace knihoven byla provedena přes nástroj NuGet, který je součástí Visual Studia.

4.2.1 AutoMapper

Knihovna AutoMapper slouží k automatické transformaci objekt na objekt [18]. Umožňuje nám jednoduchou konfiguraci typů a jednoduchý převod mezi objekty. V aplikačním rozhraní byl použit pro zredukování objektů, které budeme posílat do mobilní aplikace. Nejdříve byl nakonfigurován profil.

```
CreateMap<Comment, CommentDto>();
```

Podle tohoto profilu bylo nastaveno, z jakého objektu AutoMapper bude mapovat a jak výsledný objekt bude vypadat. Následně se zavolá třída Mapper, která je zavedena pomocí dependency injection, a vložený objekt převede na objekt nastavený v profilu.

4.2.2 LazyCache

Bylo předpokládáno, že na webovou službu bude posíláno velké množství stejných dotazů. Aby API nemuselo na každé volání služby vytvářet stále stejný dotaz do databáze, byla implementována do projektu cache, ve které se na několik minut uloží výsledek dotazu z databáze. Tímto způsobem byla snížena zátěž pro příslušný server a bylo urychleno vypracování výsledků. Přidání dotazu do cache je vidět v následujícím kódu.

```
cache.GetOrAdd("Village.GetAll", () =>
service.GetAll(cultureInfo));
```

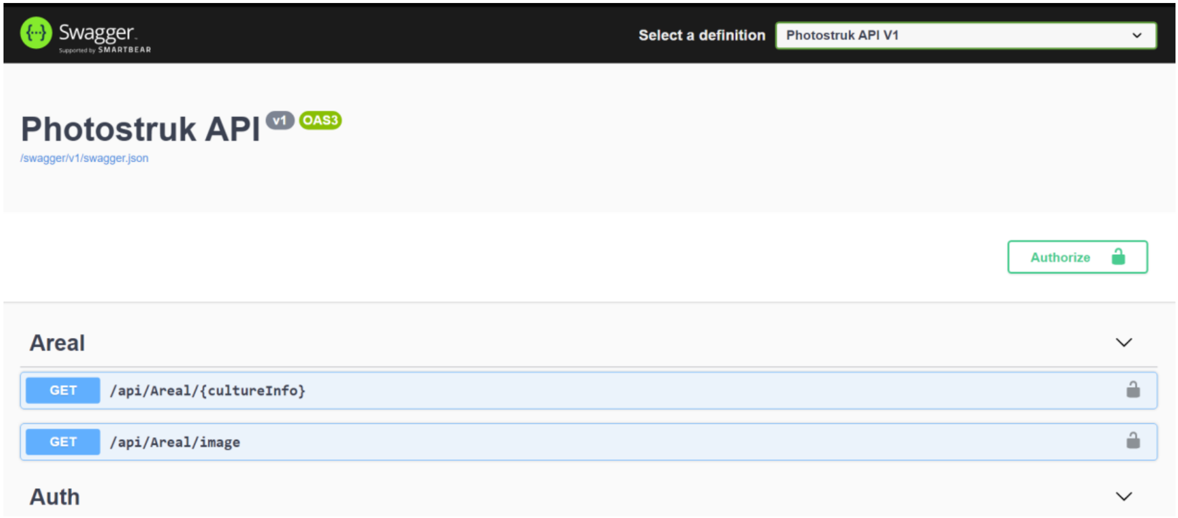
Test cache byl proveden pomocí programu Postman. Jedná se o populární program používaný, pro testování aplikačního rozhraní. Byla otestována služba vracející seznam všech vesnic v databázi. Na server bylo posláno celkem sto dotazů. První dotaz trval 1341 ms. Rychlost odpovědi u další dotazů byla v rozmezí od 810–1020 ms. U jiných služeb bylo také dosaženo zrychlení v průměru 300 ms.

4.2.3 Newtonsoft Json

Ke komunikaci mezi webovou a mobilní aplikací probíhá ve formátu JSON. Dotnet parsování formátu JSON podporuje nativně, ale celý proces neprobíhá tak intuitivně jako s knihovnou Newtonsoft Json.

4.2.4 Swagger

Pro testování aplikačního rozhraní byl použit obecně preferovaný nástroj Swagger. Tato knihovna umožňuje automaticky vygenerovat webovou stránku, do které, namapuje kontroléry vytvořené v aplikaci. Přes tuto webovou stránku je možné posílat dotazy na server bez nutnosti vytvářet dotazy manuálně. Swagger dále umožňuje přidat do dotazů autentizovanou hlavičku.



Obrázek 9 - Úvodní stránka Swaggeru

4.2.5 Xamarin Essentials

Jedná se o rozšíření původního frameworku Xamarin o API přes které je možné přistupovat k ovládacím prvkům, které se nachází v mobilním telefonu. Knihovna byla původně vyvíjena pouze komunitně, ale po počátečním úspěchu byl vývoj převzat Microsoftem a je přidávána nativně jako součást každého projektu. V aplikaci byly přes tuto knihovnu nejvíce využity možnosti kontroly internetového připojení, spouštění kódu na hlavním vlákně aplikace a zjišťování geografické polohy.

4.2.6 MapsUI

Jedním ze základních ze základních pilířů mobilní aplikace je využití map. Systémy Android a iOS mají v sobě zabudovanou rozsáhlou podporu map od společnosti Google. Toto řešení nebylo možné využít z důvodu nulové podpory offline režimu. Na trhu existuje více knihoven s podporou offline map. Bohužel většina jich je placených. Z tohoto důvodu byla pro práci s mapou použita knihovna MapsUI.

Jedná se o open-sourcové řešení vyvíjené komunitou. Druhá verze této knihovny vyšla dne 2. května 2020. Klíčová změna proběhla u změny závislosti na framework .NET Standart 2.0, který podporuje Xamarin. Nevýhodou této knihovny je podpora pouze rastrových offline map. Práce s knihovnou byla poměrně komplikovaná. Jedná se o komunitní řešení a

v důsledku toho neexistuje prakticky žádná dokumentaci. Obsluha knihovny byla zjištěna přes příklady uvedené na stránkách autora nebo analýzou zdrojového kódu.

Mapa je vytvořena inicializací konstruktoru *Map*. Dále je nutné načíst výchozí bod a poměr přiblížení, ve kterém se bude mapa zobrazovat.

```
Map Map = new Map
var startPoint = SphericalMercator.FromLonLat(14.7936, 48.7636);
Map.Home = n => n.NavigateTo(startPoint, Map.Resolutions[5]);
```

Ve výchozím stavu je mapový podklad prázdný. Nejprve aplikace ověří rychlost datového připojení. Na základě vyhodnocení rychlosti připojení se mapová vrstva bude stahovat přes OSM API nebo budou načteny mapové podklady uložené v telefonu. Podklady se dělí na vrstvy. Vrstev může mapa obsahovat hned několik. Přidání nové vrstvy provedeme následujícím kódem.

```
ILayer layer = OpenStreetMap.CreateTileLayer();
Map.Layers.Add(layer);
```

Dále je možné vytvořit na mapě kolekci pinů nebo vykreslit vektorové objekty. V aplikaci byli použity tři druhy pinů pro komponenty, vesnice a fotografie. Pinů se načítá na mapu několik stovek. Přidávání takového množství pinů značně omezuje načítání mapy. Bylo vyzkoušeno několik variant pro zrychlení načítané mapy.

4.2.7 FFImageLoading

V mobilních aplikacích se značně pracuje s obrázky. Xamarin umí pracovat s obrázky pouze ve formátu PNG. Tento nedostatek řeší knihovna FFImageLoading, která nám dovoluje zobrazit i obrázky ve formátu SVG. Dále umožnuje obrázky upravovat a cachovat a tím zrychluje celý proces zobrazování obrázků. Knihovna byla také použita k vytvoření vlastních tlačítek s SVG obrázky uvnitř.

4.3 Aplikační rozhraní

Tato část popisuje implementaci aplikačního rozhraní. Hlavní úloha serveru spočívá v načítání dat z databáze a jejich následnou úpravu a odeslání ve formátu JSON na klienskou část aplikace.

4.3.1 Validace příchozích objektů

Data od klienta mohou obsahovat chyby. Z tohoto důvodu je na serveru vždy nutné validovat příchozí objekty. V ASP.NET validace funguje automaticky přidáním speciálních validačních tagů, které jsou umístěny nad atributem v objektu.

```
[Required]
[EmailAddress]
public string Email { get; set; }
[Required]
[RegularExpression(@"^[!-@a-žA-Ž0-9]{9,30}$")]
public string Password { get; set; }
```

V kódu je vidět atribut *[Requider]*. Jeho úkolem je ověřit, jestli příslušný textový řetězec není prázdný. Validace e-mailové adresy je provedena speciálním atributem *[EmailAddress]*. Pokud aplikace vyžaduje komplikovanější validování textových řetězců bude použit atribut *[RegularExpression]*. Pokud nějaká příchozí proměnná neprojde validací, systém automaticky vygeneruje chybovou zprávu a odešle ji zpět na klienta.

V projektu bylo nutné vytvořit vlastní validační atributy pro validaci velikosti příchozích obrázků a rozeznání povolených přípon. Nejprve bylo nutné podědit třídu *ValidationAttribute*. Poté pro validaci povolených přípon přepsat metodu *IsValid* do podoby, která je vidět níže.

```
protected override ValidationResult IsValid(
        object value, ValidationContext validationContext)
        {
            var file = value as IFormFile;
            string extension = string.Empty;
            if (file != null)
            {
```

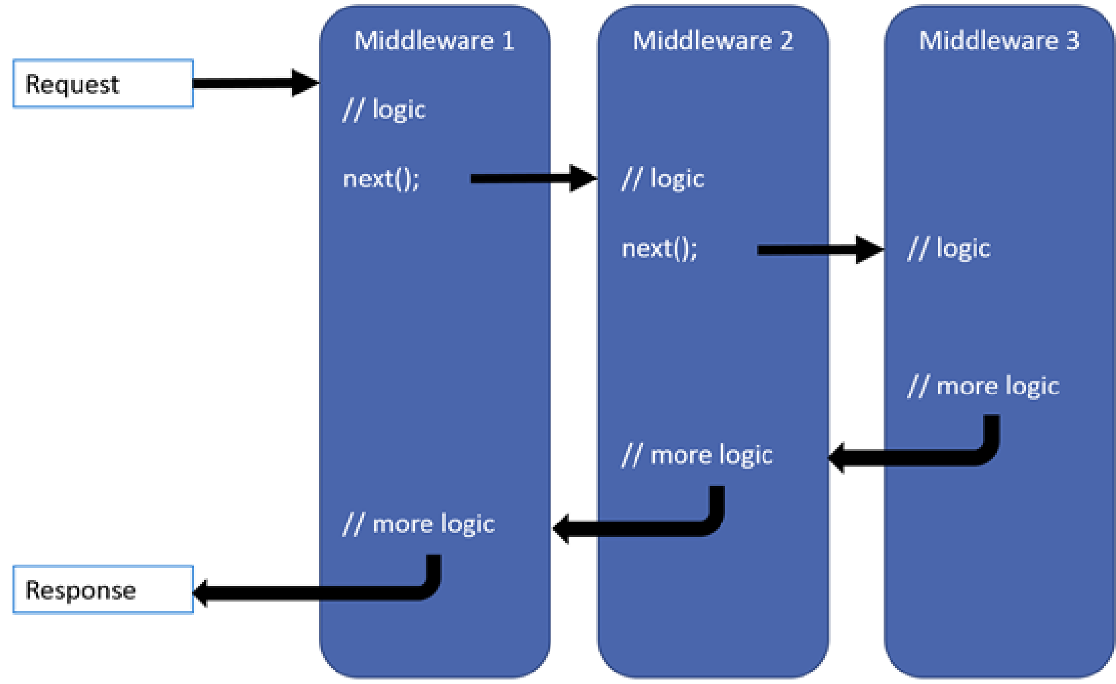
```
                extension = Path.GetExtension(file.FileName);

                if
(!_Extensions.Contains(extension.ToLower()))
                {
                    return new
ValidationResult(GetErrorMessage());
                }
            }

            return ValidationResult.Success;
        }
```

4.3.2 ASP.NET Core Middleware

Middleware je program, který sestaví pipeline pro zpracování dotazů [19]. Používá se pro:

- Rozhodování o tom, jaký dotaz bude poslán do další komponenty
- Vykonání kódu před nebo po přesměrováním do další komponenty

Middleware se skládá z posloupnosti volání delegovaných dotazů. Příklad je uveden na následujícím obrázku.



Obrázek 10 - Příklad middleware posloupnosti, převzato z [19]

Systém využívá middleware na rozeznání chybových dotazů. Pro tuto úlohu byla vytvořena třída *HttpStatusCodeExceptionMiddleware*. Zde byla v metodě *Invoke*, která je spuštěna při probuzení třídy, vytvořena logika pro zpracování HTTP kontextu.

```
public async Task Invoke(HttpContext context)
        {
            try
            {
                await _next(context).ConfigureAwait(false);
            }
            catch (HttpStatusCodeException ex)
            {
                if (context.Response.HasStarted)
                {
                    _logger.LogWarning("The response has
already started, the http status code middleware will not be
executed.");
                    throw;
                }
```

```
                context.Response.Clear();
                context.Response.StatusCode = ex.StatusCode;
                context.Response.ContentType = ex.ContentType;

                await
context.Response.WriteAsync(ex.Message).ConfigureAwait(false);

                return;
            }
        }
```

4.3.3 Načítání obrázků

Jednou z nejdůležitějších tabulek v databázi jsou fotografie historických budov. Fotografie jsou uloženy ve složkách na disku a v databázi byla uložena pouze cesta k nim. Aby bylo možné přistupovat k obrázkům v ASP.NET aplikacích, fotky byly překopírovány do statických souborů, do kterých je možné vzdáleně přistupovat. Statické soubory se standardně ukládají do složky *wwwroot*. Dále bylo nutné přidat do třídy *Startup* komponentu *StaticFiles*. Pro načtení souborů na disku byla použita třída *PhysicalFileProvider*. Do této třídy se jako parametr v kontruktoru použije cesta vybraného souboru.

```
if (!File.Exists(hostingEnvironment.WebRootPath + image.Path))
            {
                throw new HttpStatusCodeException("Image not
found on disk", StatusCodes.Status404NotFound);
            }


            IFileProvider provider = new
PhysicalFileProvider(hostingEnvironment.WebRootPath +
ImagePath);
            IFileInfo fileInfo =
provider.GetFileInfo(image.Name);
```

Nejprve aplikace zkontroluje, jestli se vybraný soubor na disku nachází. Poté se uloží výsledek ze třídy *PhysicalFileProvider* do interface *IFileProvider.* Z tohoto interface je načten detail souboru podle jeho názvu do interface *IFileInfo*. Obrázek je do DTO objektu uložen jako Stream pomoci metody *CreateReadStream.* Výsledek je poté odeslán v kontroleru jako třída *File*.

```
imageDto.FileStream = fileInfo.CreateReadStream();

return File(imageDto.FileStream, imageDto.ContentType);
```

4.3.4 Načítání dat z databáze Photostruk

Databáze Photostruk je poměrně rozsáhlá databáze obsahující všechna metadata a fotografie použitých v projektu. Databáze měla být snadno rozšiřitelná, proto se v ní nachází mnoho asociačních tabulek. Přístup k takovým datům je přes standartní LINQ, které podporuje C# pro načítání dat poměrně složité. Z tohoto důvodu musely být dotazy do databáze napsány ručně. Výsledek je poté ručně uložen do objektu posílaného na klienta.

4.4 Mobilní aplikace

V této části jsou popsány funkce mobilní aplikace. Její hlavní úloha spočívá v grafickém promítnutí dat uživateli. Dále jsou probrány některé funkce, které musela mobilní aplikace splnit.

4.4.1 Multijazyčnost

Aplikace je zaměřena spíše na české a německé občany. Z tohoto důvodu bylo nutné vydat aplikaci ve více jazycích. Jako doplňující jazyk byla zvolena angličtina. V Xamarinu je vytvoření multijazyčné aplikace poměrně jednoduché [4]. Nejprve se vytvoří jazykové soubory. Každý jazyk, který bude v aplikaci použit, musí mít svůj vlastní soubor. U souborů je důležité jejich pojmenování. Prostřední název musí být vždy zkratka dané země, pro kterou je soubor platný, například *Resource.cs.resx*. Poté se texty přidávají do kódu ve speciální

syntaxi popsané níže. Na základě tohoto způsobu se při startu aplikace vyberou texty příslušící jazyku, kterém běží operační systém.

```
Text="{x:Static translate:R.Description}"
```

4.4.2 Autentizace třetích stran

Pro povolení autentizace přes Facebook API byla nejprve aplikace zaregistrována na webové stránce Facebook pro developery. Poté byla povolena autentizace pro všechny platformy, na kterých bude ověřování běžet. Nastavení pro operační systém Android obsahuje Google Play Package Name, Class Name a Key Hashes. Google Play Package Name je název balíčku, který se skládá ze jména aplikace, země a organizace pod kterou byl projekt vyvíjen. V tomto případě *cz.jcu.photostruk.* Do textového pole Key Hashes byl vložen hash vytvořený z klíčenky, kterou se podepisuje vytvořená aplikace. Hash byl vytvořen následujícím příkazem v příkazové řádce.

```
keytool -exportcert -alias <KEY_STORE_ALIAS> -keystore
<KEY_STORE_PATH> | openssl sha1 -binary | openssl base64
```

4.4.3 Lokální databáze

Jak již bylo zmíněno v části výše, jako lokální databáze byla použita Realm. Pro usnadnění práce s databází byla vytvořena generická třída *DbService*. Třída řídí CRUD operace zvolené entity. Dále se stará o načtení instance databáze.

```
public TEntity GetById(long id)
        {
            return _database.Find<TEntity>(id);
        }
```

V kódu je vidět načtení entity pomocí *id.* Pro práci s daty je nejprve vybrána instance databáze *_database*. Budou hledána data pomocí primárního klíče. Z tohoto důvodu byla použita metoda *Find*. Pokud by bylo hledání vyvoláno pomocí jiných atributů, které nejsou indexované, bude použita metoda *Where*. V následujícím kódu je vidět načtení instance databáze a uložení vesnic do databázi.

```
DbService<Village> dbService = new DbService<Village>();
dbService.SaveItemsAsync(entities);
```

4.4.4 Komunikace s API

Pro centralizaci dotazů na server byla vytvořena třída *WebService.* Třída obsahuje metody pro všechny dotazy odesílané na server. Dále zajišťuje odchycení chyb a ověřuje internetové připojení. V následujícím kódu je vidět implementace metody přihlášení.

```
public async Task<LoginResponse> LoginAsync(string email,
string password, CancellationToken cancellationToken)
        {
            JObject signInData = JObject.FromObject(new
            {
                email,
                password
            });
            return await PostAsync<LoginResponse>(ApiUrl +
"auth/login", GetServiceHeader(), signInData,
cancellationToken);
        }
```

Parametry metody obsahují atributy, které bude mít odesílaný JSON. Dále se zde nachází *CancellationToken* pro případ ukončení dotazu, pokud by měl dotaz trvat příliš dlouho. V metodě se poté vytváří JSON objekt pomocí třídy *JObject*. Následně je volána metoda *PostAsync<LoginResponse>,* ve které se pomocí parametrů posílá URL API, hlavička dotazu, data a token. Generická třída označuje, v jakém objektu bude přicházet odpověď. Odeslání dotazu probíhá přes třídu *WebRequest*. Je zde implementováno nastavení dotazu a jeho následné zpracování.

4.4.5 Určování polohy

K funkci určování polohy byla využita knihovna Xamarin Essentials [4]. Pro určování polohy je nejprve nutné povolit ve složce *Android Manifest* hledání polohy. Aby tato funkce byla použitelná, musí uživatel povolit sledování na svém mobilu. V následujícím kódu je vidět

postup načtení polohy. Nejprve je ověřeno, jestli už bylo o načtení polohy dříve zažádáno pomocí metody *GetLastKnownLocationAsync.* Pokud předchozí poloha nebyla nalezela je zavolána metoda *GetLocationAsync*. Načítání polohy rozlišuje několik možností přesnosti.

```
var location = await Geolocation.GetLastKnownLocationAsync();
                if (location == null)
                {
                    location = await
Geolocation.GetLocationAsync(new GeolocationRequest
                    {
                        DesiredAccuracy =
GeolocationAccuracy.Medium,
                        Timeout = TimeSpan.FromSeconds(30)
                    });
                }
```

4.4.6 Styly

Framework Xamarin Forms podporuje možnost, která je známá z webových aplikací. Nastavení stylů na jednom centrálním místě. Tento přístup umožňuje vývojáři jednoduše změnit design komponenty, která daný styl obsahuje. Dále přidává na přehlednosti kódu. Jednotlivé styly je možné dědit a tím omezit redundanci výsledného kódu. Vývojář může přidat vlastní styl nebo nastavit parametry pro všechny komponenty stejného druhu, jak je vidět v následujícím kódu.

```
<Style TargetType="Label">
        <Setter Property="FontFamily" Value="{StaticResource
FontFamilyDefault}" />
        <Setter Property="FontSize" Value="{StaticResource
DefaultFontSize}" />
        <Setter Property="TextColor" Value="{StaticResource
PrimaryColor}" />
        <Setter Property="LineBreakMode" Value="NoWrap"/>
</Style>
```

Nový styl je přidán klíčovým tagem *Style.* Atribut *TargetType* určuje, že vybraná komponenta bude typu *Label*. Dále jsou pomocí tagu *Setter* nastaveny *Property* a *Value* pro jednotlivé atributy.

4.4.7 3D modely

Pro webovou aplikaci Photostruk bylo vytvořeno několik 3D modelů znázorňujících umístění objektů v původním prostředí. Zobrazení pokročilých grafických prvků není v multiplatformním vývoji primitivní záležitost. Framework Xamarin Forms je primárně uzpůsoben k vývoji formulářových aplikací. Nejvíce relevantním řešením byla knihovna UrhoSharp, která se danou problematikou zabývá. Naneštěstí vývoj tohoto projektu byl před dvěma lety ukončen. To je z pohledu vývoje mobilních zařízeních velké množství času. Knihovna stačila za tuto dobu zastarat a nedá se implementovat na nových zařízeních. Nakonec bylo implementováno alternativní řešení inspirované webovou aplikací Photostruk. Ta zobrazuje 3D modely pomocí videozáznamů. Výhodou tohoto řešení je jednodušší implementace, zobrazení prostředí a konzistentní zobrazení pro obě platformy. Na druhou stranu data zabírají více prostoru a vytrácí se možnost volně manipulovat s objektem.

4.5 Mapové podklady

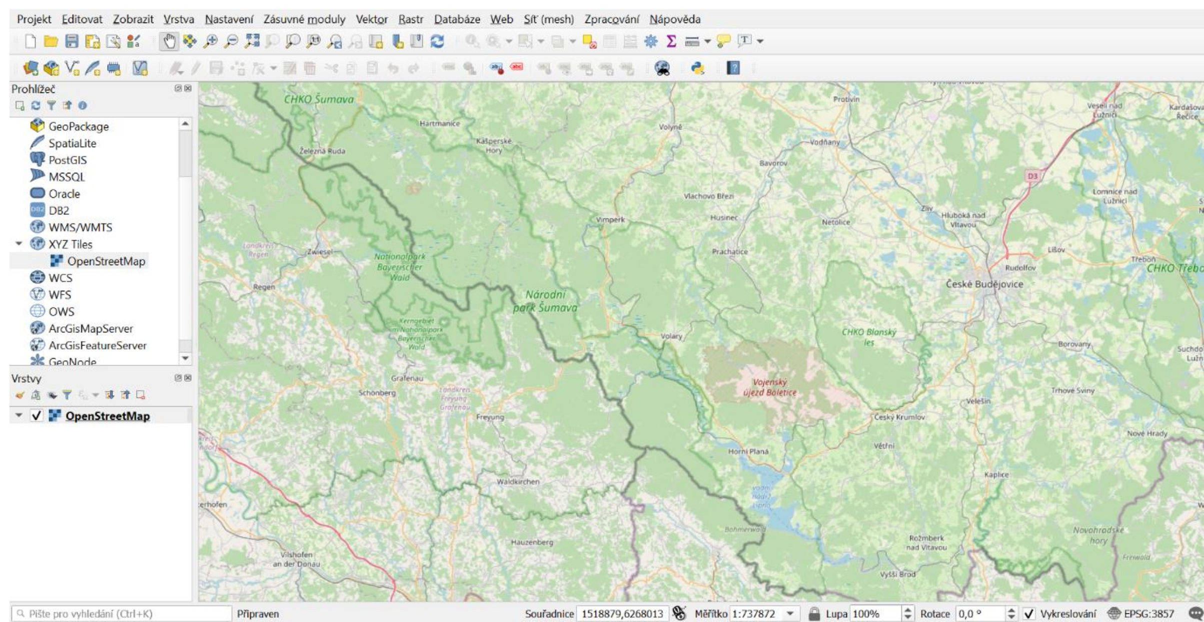
Existuje mnoho společností zabývajících se tvorbou map. Většina těchto společností tvoří mapy pro své soukromé účely nebo je nabízí v online verzi pouze dostupnou přes jejich aplikační rozhraní. Výjimkou je společnost Open Street Map (OSM) [20]. OSM projekt je tvořen komunitou. Její hlavní cíl je vytvořit otevřenou a editovatelnou mapu světa. Motivace pro vývoj a růst OSM je tvořena restrikcemi o používání nebo dostupnosti v různých částech světa. OSM je vyvíjen jako open-source. Data je možné používat pro libovolný účel, dokud jsou uvedena v projektu autorská práva. Projekt založil Steve Coast v roce 2004 ve Velké Británii. Prvotní myšlenka byla inspirována úspěchem wikipedie.

Projekt dnes již zmapoval prakticky celý svět. Jeho volně dostupné aplikační rozhraní využívá nejrůznější variace aplikací a knihoven. Rovněž lze stáhnout jejich offline verzi. Mapy se nacházejí na serveru ve vektorové podobě. Kompletní verze map světa zabírá 55 GB.

Naštěstí lze stáhnout mapy podle jednotlivých států. Česká republika zde zabírá 727 MB. Podklady z OSM byly použity jako zdroj v diplomové práci. Knihovna MapsUI podporuje komunikaci s jejich API pro online použití map.

4.5.1 Tvorba mapových podkladů

Mobilní aplikace je zaměřená pouze na oblast česko-bavorského příhraničí, proto nebylo nutné přidat jako mapový podklad celou mapu České republiky. Z tohoto důvodu byla vytvořena mapa obsahující podklady zaměřující se pouze na tuto oblast. Díky využití knihovny MapsUI bylo nutné vytvořit rastrovou mapu. Rastrová data jsou tvořena maticemi pixelů. Obvykle jsou přizpůsobeny do čtverců o velikosti 256x256 pixelů, ale nemusí to být pravidlem. Z mřížky čtverců je poté utvořena celková mapa. Rastrové mapy se skládají z několika vrstev podle míry detailu, který je na mapě viditelný. Zdroj pro renderování rastrové mapy byl vytvořen přes program QGIS [21]. Jedná se o aplikaci pro vizualizaci, tvorbu a úpravu mapových podkladů. Projekt je řízen komunitou. Na obrázku níže je vidět uživatelské rozhraní aplikace.



Obrázek 11 - Aplikace QGIS

V aplikaci byl nejprve vybrán mapový podklad OSM. Cílová oblast nedosahovala takové velikosti, proto jednotlivé snímky mapy byly vytvářeny ručně. Tento výběr proběhl v několika vrstvách. Výsledek byl exportován do formátu PNG. Každý soubor obsahuje metadata se souřadnicemi. Tyto soubory budou sloužit jako podklad pro tvorbu map.

4.5.2 Renderování map

Renderování map proběhlo v programu MapTiler [22]. Program je vyvíjen společností sídlící v Brně. Dokáže vyčíst metadata z mapových fotografií a sestavit z nich kompletní mapový podklad. Výsledný soubor je ve formátu MBTiles [20], který se používá v mobilních aplikacích. Soubor je technicky SQLite databáze. Bohužel tento program je placený a byl použit pouze ve verzi zdarma. Hlavní omezení této verze je menší rozloha renderované mapy a přidání vodoznaků do mapy.

5 Nasazení

Kapitola se bude věnovat nasazení systému. V první části je popsáno nastavení virtuálního stroje. Dále se kapitola věnuje nasazení na webový server Nginx a poté přesunutí aplikačního rozhraní na Azure Cloud [23]. Následně je vysvětlena změna přesunutí datového uložiště pro fotografie. V závětu je popsán průběh testování aplikace.

5.1 Virtuální stroj

V rámci projektu byl pro nasazení aplikačního rozhraní zřízen virtuální stroj od Jihočeské univerzity. Operační systém, na kterém virtuální stroj běží je Ubuntu 20.04 LTS. Na zařízení byl povolen vzdálený přístup pomocí programu Remote Desktop Connection a SSH. Dále byla na Ubuntu nainstalována relační databáze PostgreSQL obsahující uživatelská data, a webový server Nginx. Používání virtuálního stroje sebou neslo jednu značnou nevýhodu. Jihočeská univerzita ve své síti blokuje dotazy na servery na portu 80 a 443. Díky tomuto problému nebylo možné získat platný certifikát od certifikační autority. Tento problém zredukoval možnosti využití virtuálu pouze pro testovací účely aplikačního rozhraní a na využití pro PostgreSQL databázi.

5.2 Nasazení na Nginx

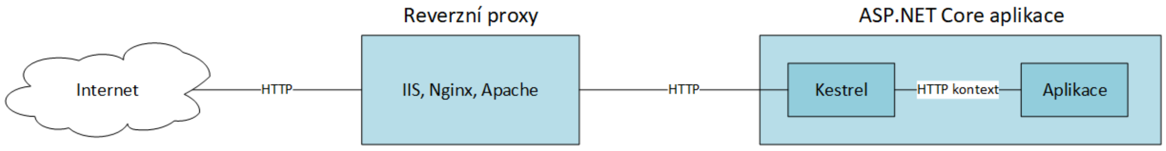
ASP.NET Core podporuje nasazení na linuxové servery Apache a Nginx [24]. V diplomové práci byl zvolen druhý jmenovaný. Nejprve byl na server nainstalován balíček .NET Core runtime pro instalaci knihoven na spuštění aplikace. Poté byla aplikace vydána pomocí následujícího příkazu.

```
dotnet publish --configuration Release
```

Publikované soubory byly přesunuty do složky, na kterou bude webový server odkazovat. ASP.NET Core aplikace v sobě obsahují multiplatformní webový server Kestrel. Je dostačující pro posílání dynamického webového kontentu v ASP.NET Core aplikacích. Bohužel oproti běžným webovým serverům, jako jsou IIS, Apache nebo Nginx nejsou, jeho možnosti tak bohaté. V důsledku toho byl webový server Nginx nastaven jako reverzní proxy.

Jedná se o běžné nastavení pro dynamické webové aplikace. Umístění reverzního proxy serveru může být přímo vedle webového serveru nebo může být nasazen na separátním zařízení. Toto jsou výhody zavedení reverzní proxy:

- Limituje veřejně vystavenou plochu aplikace
- Přidává další vrstvu ochrany
- Snižuje zátěž skutečného serveru
- Přidává možnost komprese a cache dotazů
- Zjednodušuje nastavení HTTPS komunikace. Pouze reverzní proxy server vyžaduje X.509 certifikát



Obrázek 12 - Schéma nastavení reverzní proxy

Protože aplikace bude schovaná za reverzní proxy, bylo nutné implementovat middleware pro přeposílání hlaviček dotazu. Middleware upravuje schéma dotazu přidáním do hlavičky atribut *X-Forwaded-Proto* a *X-Forwaded-For.*

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto
});
```

Po instalaci webového serveru Nginx bylo nutné upravit konfigurační soubor do podoby viditelné níže. Nejprve je nastaven port, na kterém bude server poslouchat pomocí příkazu *listen*. Poté je nastaveno serveru doménové jméno *test.photostruk.tk.* Doména byla vytvořena na serveru *freenom.com*. Tato společnost poskytuje své domény na rok zdarma. V bloku *location* je provedeno nastavení serveru jako reverzní proxy. Dále je zde vidět adresa *localhost:5001*, na které běží ASP.NET Core aplikace.

```
server
{
     listen 0.0.0.0:8443 ssl deafault_server
     listen [::]:8443 ssl default_server
     server_name test.photostruk.tk

     location / {
          proxy_pass          https://localhost:5001;
          proxy_http_version  1.1;
          proxy_set_header    Upgrade $http_upgrade;
          proxy_set_header    Connection keep-alive;
          proxy_set_header    Host $host;

proxy_cache_bypass  $http_upgrade;
          proxy_set_header    X-Forwarded-For
$proxy_add_x_forwarded_for;
          proxy_set_header    X-Forwarded-Proto $scheme;
}
}
```

Do nastavení serveru bylo také přidáno přesměrování z portu 8080 na 8443.

```
server {
     listen 8080;
     listen [::]:8080;
     server_name test.photostruk.tk:8443;
     return 302 https:// $server_name$request_uri;
}
```

5.2.1 Self-signed certifikát

Poté, co nemohl být obdržen certifikát od autorizované certifikační autority, byl v zájmu větší bezpečnosti vytvořen alespoň certifikát vlastní. Použitím takového certifikátu vzniká riziko podvržení ze strany útočníka. Pro testovací účely by měl být ale vlastní certifikát dostačující [25].

Vytvoření vlastního certifikátu bylo provedeno v programu OpenSSL pomocí následujícího příkazu.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout /etc/ssl/private/nginx.key -out
/etc/ssl/certs/nginx.crt
```

Při tvorbě byly zadány informace o organizaci, která certifikát vytváří. Po úspěšné tvorbě certifikátu byl přidán do konfiguračního souboru Nginx.

5.3 Nasazení na Azure

Po neúspěšném nasazení aplikačního rozhraní na virtuální stroj byla vymyšlena alternativa v podobě Azure [23]. Jedná se o cloudovou službu podporující velké množství programovacích jazyků, nástrojů a frameworků. Byl vybrán z důvodu silné provázanosti s programovacím jazykem C# a frameworkem ASP.NET Core. Jihočeská univerzita poskytuje pro studenty testovací licenci na 12 měsíců zdarma.

Po založení účtu byla založena skupina prostředků. Do této skupiny jsou ukládány všechny služby související s danou aplikací. Dále byla pod touto skupinou prostředků vytvořena aplikační služba. V nastavení služby bylo vybráno, jaký bude použit framework a kolik hardwarových prostředků bude služba vyžadovat. V testovací licenci jsou možnosti dosti omezené, ale na zkoušku funkčnosti systému jsou dostačující. Přístup k aplikačnímu rozhraní byl zatím omezen na povolené IP adresy. Poté, co byla aplikační služba spuštěna, bylo provedeno propojení s Visual Studiem a následně nahrání API na Azure. Aplikační rozhraní běží na adrese *https://photostruk.azurewebsites.net*. Azure při založení webové služby automaticky založí validní certifikát. Bez ověřeného certifikátu nebylo možné otestovat komunikaci s mobilním zařízením. Nešifrované nebo neověřené připojení vyvolávalo chybu v komunikaci.

5.4 Přesun fotografií na Cloudinary

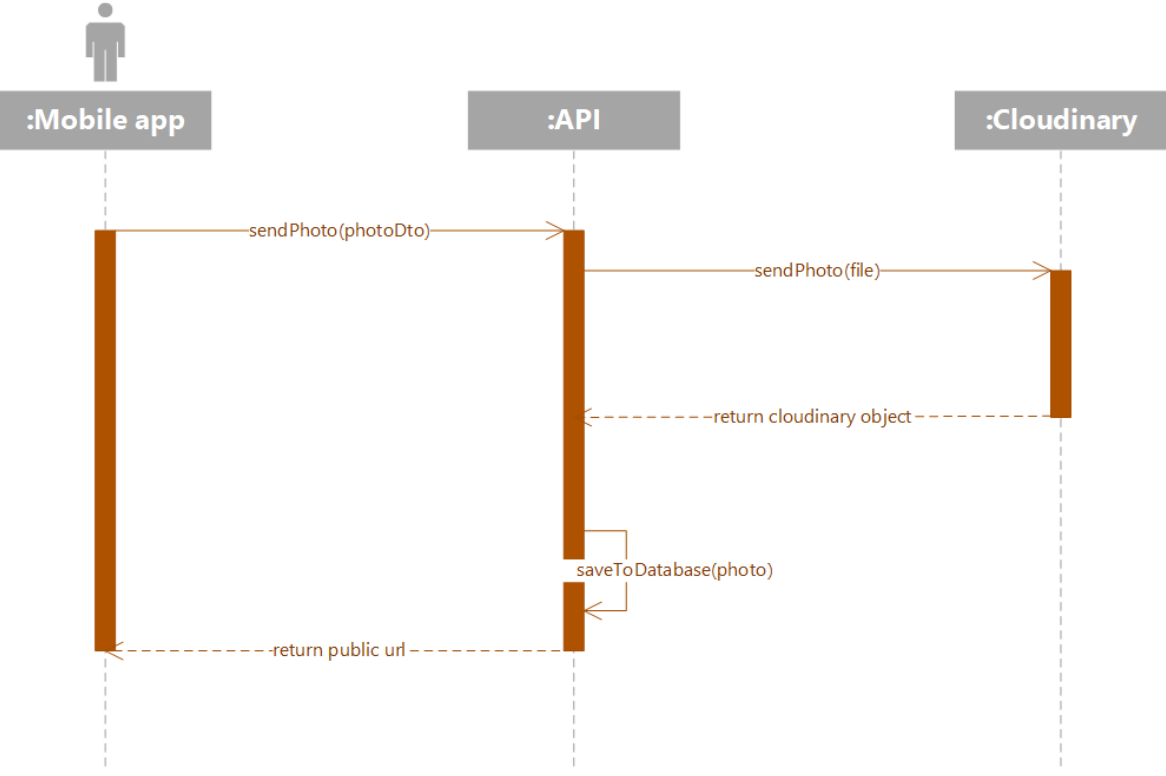
V prvotním návrhu systému bylo počítáno s ukládáním fotografií na disk. Tento způsob byl využit i u původní databáze Photostruk. V databázi se nachází pouze cesta k hledanému

souboru. Výhoda tohoto řešení spočívá v menší zátěži databáze a menšímu objemu dat umístěných v databázi. Naopak problém může nastat při přesunutí souboru nebo úpravě nasazeného prostředí.

Po neúspěšném získání certifikátu od certifikační autority nemohlo být aplikační rozhraní nasazeno na virtuální stroj od JCU. Byl proveden přechod API na Azure Cloud, na kterém nebylo možné uchovávat datové soubory. Musela být vymyšlena alternativa k ukládání souborů. Touto alternativou je platforma Cloudinary [26]. Jedná se o populární cloud uzpůsobený pro ukládání médií. Disponuje velmi kvalitním kompresním mechanizmem. Práce se systémem byla vždy rychlá a intuitivní. Také umožňuje celou řadu možností transformací u vložených medií. Výběru této služby také pomohlo existující SDK pro .NET systémy. V budoucnu by také mohla být využita možnost připojení Cloudinary na API na rozpoznávání objektů na fotografii od společnosti Google nebo Amazon.

Pro přidání Cloudinary API do aplikačního rozhraní bylo nejprve nutné na jejich webových stránkách založit nový účet. Založením účtu uživatel získá plán zdarma, ve kterém může využít 25 kreditů měsíčně. 1 kredit lze využít na 1000 transformací, 1 GB úložného prostoru nebo 1 GB šířky pásma. Díky vysoké možnosti komprese, která se pohybuje u fotografií v rámci desítek KB, by měla být kapacita plánu zdarma pro účely systému dostačující.

Pro implementaci spojení s Cloudinary bylo nejprve nutné získat z webové stránky Cloud name, API Key a API Secret. Tyto údaje slouží k ověření přístupu do aplikace. Jednotlivé kroky, které aplikace provede, jsou vidět na následujícím sekvenčním diagramu.



Obrázek 13 - Sekvenční diagram nahrání fotografie

- Nejprve proběhne v klientské části aplikace odeslání fotografie
- Fotografie je zpracována v aplikačním rozhraní
- Ve třídě *ImageUploadParams* mohou být nastaveny parametry pro uložení fotografie na Cloudinary
- Fotografie je odeslána a uložena na Cloudinary
- Coudinary pošle zpět objekt, který obsahuje veřejné URL
- Údaje z objektu se uloží do databáze
- Výsledný objekt se pošle na klientkou část aplikace

5.5 Testování mobilní aplikace

Testování mobilní aplikace pro operační systém Android bylo provedeno na simulátorech i reálných zařízeních. Simulátor byl vytvořen přes Android Device Manager zakomponovaný ve Visual Studiu. Při tvorbě jsou nejprve nastaveny parametry. Je vybrán typ mobilního

zařízení, na kterém bude simulace provedena. Dále je vybrán typ procesoru, verze operačního systému a dodatečné hardwarové nastavení. Poté je simulátor stažen, nakonfigurován a připraven ke spuštění. Aby mohl být simulátor spuštěn, je také nutné povolit virtualizaci v nastavení BIOS.

Na zařízeních s operačním systémem iOS je situace komplikovanější. Apple přistupuje k problematice odlišně a snaží se být oproti operačnímu systému Android více uzavřený. Pro testovací účely je nutné mít platný vývojářský účet a ověřené zařízení od Applu. Poté může uživatel stáhnout platné certifikáty z vývojářského účtu a spárovat Visual Studio se zařízením. Následně je možné provést simulaci na simulačním zařízení nebo nasazení na skutečné zařízení s iOS. Bohužel autor diplomové práce nedisponuje vývojářským účtem ani zařízením od Applu a proto nemohlo být testování provedeno.

6 Závěr

Hlavním cílem diplomové práce byl návrh a implementace mobilní aplikace pro turisty v prostředí Bavorského lesa a Šumavy.

Po úvodní kapitole byla práce věnována obecné analýze vývoje mobilních aplikací. Byly popsány možnosti existujících technologií a byl vybrán nejvhodnější framework pro řešení daného problému.

Třetí kapitola byla věnovaná návrhu řešení. Nejprve bylo vytvořeno schéma architektury systému založené na návrhu klient-server. Poté byl proveden návrh architektury aplikačního rozhraní. Následně byl popsán návrh mobilní aplikace založené na architektuře MVVM. Návrh byl doplněn o diagram užití a diagram tříd. V závěru kapitoly byl popsán grafický návrh obrazovek mobilní aplikace. Inspirací byla již existující webová aplikace.

Na základě návrhu z předchozí kapitoly byla provedena implementace systému. Na začátku kapitoly byl proveden výčet sady technologií, které byly v práci použity. Technologie byly vybrány hlavně podle zkušeností autora a jsou založené na programovacím jazyku C#. Dále byly popsány jednotlivé knihovny, které byly pro implementaci systému použity. Následně byla popsána implementace aplikačního rozhraní založeného na frameworku ASP.NET Core a implementace mobilní aplikace založené na frameworku Xamarin Forms. Konec kapitoly byl zaměřen na získávání mapových podkladů z API Open Street Map a také na tvorbu vlastních map pro účely mobilní aplikace.

V páté kapitole byl popsán postup nasazení aplikačního rozhraní. Nejprve bylo popsáno nastavení virtuálního stroje, který autor obdržel od Jihočeské univerzity pro testovací účely. Poté je čtenář seznámen s nasazením aplikačního rozhraní na virtuální stroj a jeho následným přesunem na Azure Cloud. V následující části je vysvětlen postup propojení aplikačního rozhraní s cloudem Cloudinary, na který bylo přesunuto ukládání fotografií. V závěru kapitoly byl popsán průběh testování mobilní aplikace.

Vzhledem k výše uvedenému lze funkce systému považovat za splněné.

Seznam literatury

[1] *Photostruk* [online]. Deggendorf: Technische Hochschule Deggendorf, 2020 [cit. 2020-09-08]. Dostupné z: https://photostruk.cz/

[2] *Progressive Web Apps* [online]. online: web.dev, 2020 [cit. 2020-04-09]. Dostupné z: https://web.dev/

[3] Progressive Web Apps with Angular. *Modus* [online]. online: Ahsan Ayaz, 2017 [cit. 2020-04-09]. Dostupné z: https://moduscreate.com/blog/creating-progressive-web-apps-using-angular/

[4] Xamarin. *Microsoft* [online]. Redmond: Microsoft, 2012 [cit. 2020-11-09]. Dostupné z: https://dotnet.microsoft.com/apps/xamarin

[5] *React Native* [online]. Menlo Park: Facebook, 2015 [cit. 2020-04-11]. Dostupné z: https://reactnative.dev/

[6] *Flutter* [online]. Mountain View: Google, 2017 [cit. 2020-04-11]. Dostupné z: https://flutter.dev/

[7] Klient-Server model. *TechTerms* [online]. online: TechTerms, 2016 [cit. 2020-11-23]. Dostupné z: https://techterms.com/definition/client-server_model

[8] Entity Relationship Diagram. *Visual Paradigm* [online]. Hong Kong: Visual Paradigm, 2005 [cit. 2020-10-19]. Dostupné z: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

[9] Repository pattern. *Microsoft* [online]. Redmond: Microsoft, 2018 [cit. 2020-11-24]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[10] LINQ. *Microsoft* [online]. Redmond: Microsoft, 2017 [cit. 2020-11-18]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

[11] *JWT* [online]. online: authO, 2013 [cit. 2020-11-24]. Dostupné z: https://jwt.io/

[12] The Model-View-ViewModel Pattern. *Microsoft* [online]. Redmond: Microsoft, 2017 [cit. 2020-11-14]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[13] Use case diagram. *Visual Paradigm* [online]. Hong Kong: Visual Paradigm, 2005 [cit. 2020-11-01]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

[14] Class diagram. *Visual Paradigm* [online]. Hong Kong: Visual Paradigm, 2005 [cit. 2020-11-01]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

[15] ASP.NET Core. *Microsoft* [online]. Redmond: Microsoft, 2017 [cit. 2017-12-09]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/

[16] *ASP.NET* [online]. Redmond: Microsoft, 2007 [cit. 2020-12-09]. Dostupné z: https://www.asp.net/

[17] Realm. *MongoDB* [online]. New York: MongoDB, 2017 [cit. 2020-11-10]. Dostupné z: https://www.mongodb.com/realm/mobile/database

[18] *Automapper* [online]. Austin: Jimmy Bogard Consulting, 2017 [cit. 2020-11-09]. Dostupné z: https://automapper.org/

[19] ASP.NET Core Middleware. *Microsoft* [online]. Redmond: Microsoft, 2017 [cit. 2020-11-11]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0

[20] *Open Street Map* [online]. Cambridge: Open Street Map Foundation, 2020 [cit. 2020-11-03]. Dostupné z: https://www.openstreetmap.org/

[21] *QGIS* [online]. online: QGIS, QGIS [cit. 2020-11-04]. Dostupné z: https://qgis.org/

[22] *MapTiler* [online]. Brno: MapTiler, 2020 [cit. 2020-11-03]. Dostupné z: https://www.maptiler.com/

[23] *Azure* [online]. Redmond: Microsoft, 2010 [cit. 2020-11-04]. Dostupné z: https://azure.microsoft.com/

[24] ASP.NET Core on Linux with Nginx. *Microsoft* [online]. Redmond: Microsoft, 2020 [cit. 2020-11-24]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-5.0

[25] Self-Signed SSL. *Cloud savvy IT* [online]. online: Anthony Heddings, 2020 [cit. 2020-11-05]. Dostupné z: https://www.cloudsavvyit.com/1306/how-to-create-and-use-self-signed-ssl-on-nginx/

[26] *Cloudinary* [online]. London: Cloudinary, 2011 [cit. 2020-11-02]. Dostupné z: https://cloudinary.com/

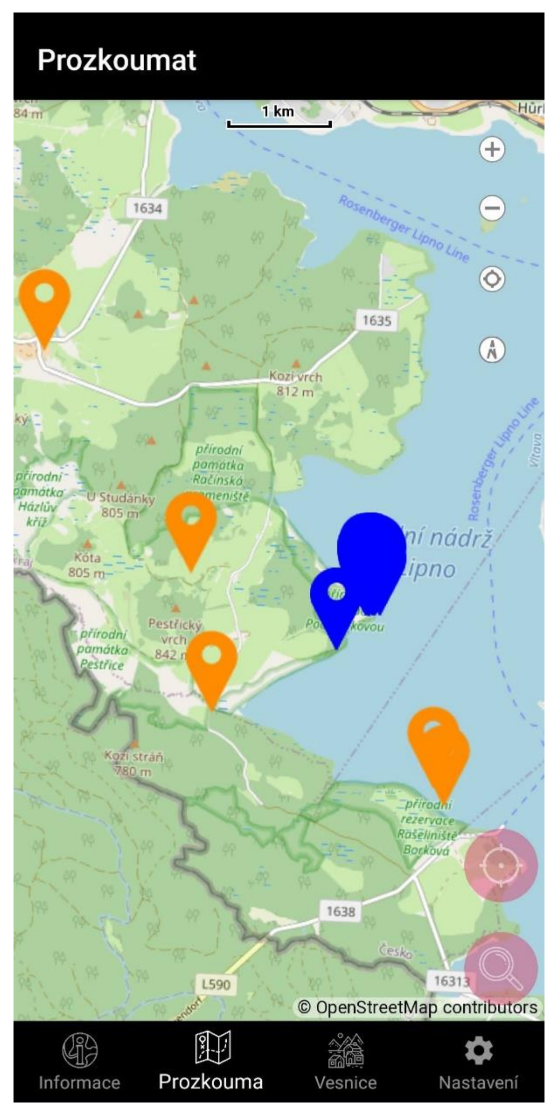
Seznam obrázků

Obrázek 1 - Webová aplikace Photostruk, převzato z [1] 8
Obrázek 2 - Návrh architektury 12
Obrázek 3 - ER diagram databáze uživatelů 14
Obrázek 4 - Vrstvy aplikačního rozhraní 16
Obrázek 5 - MVVM diagram, převzato z [12] 21
Obrázek 6 - Diagram užití 22
Obrázek 7 - Diagram tříd 23
Obrázek 8 - Hierarchie vlastních vykreslovačů, převzato z [4] 29
Obrázek 9 - Úvodní stránka Swaggeru 34
Obrázek 10 - Příklad middleware posloupnosti, převzato z [19] 38
Obrázek 11 - Aplikace QGIS 45
Obrázek 12 - Schéma nastavení reverzní proxy 48
Obrázek 13 - Sekvenční diagram nahrání fotografie 52
Obrázek 14 - Obrazovka mapa 60
Obrázek 15 - Obrazovka seznam vesnic 60
Obrázek 16 - Obrazovka filtr 61
Obrázek 17 - Obrazovka vesnice přehled 61
Obrázek 18 - Obrazovka vesnice historie 62
Obrázek 19 - Obrazovka vesnice fotografie 62

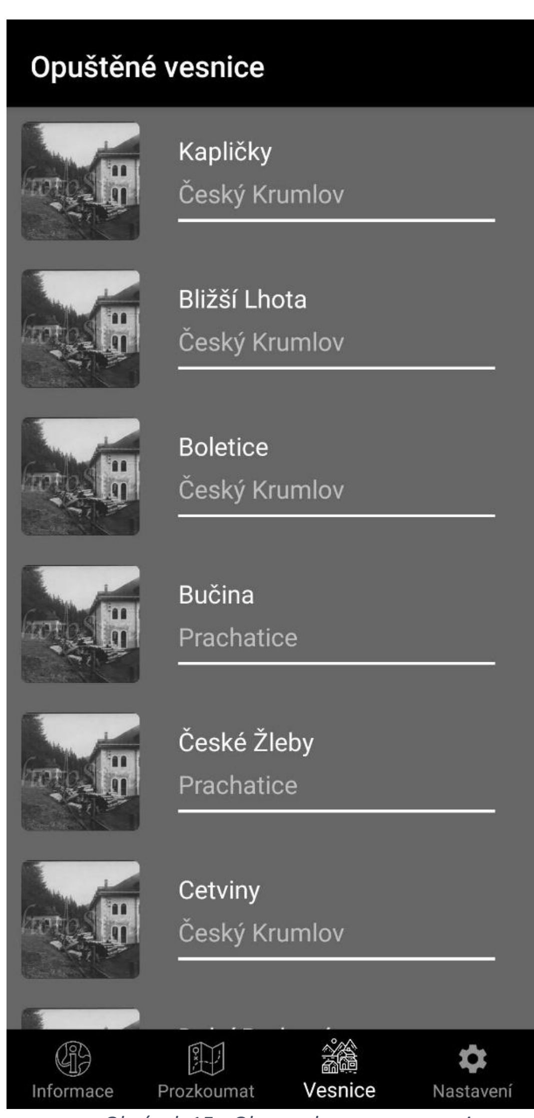
Seznam tabulek

Tabulka 1 - REST API serveru 15

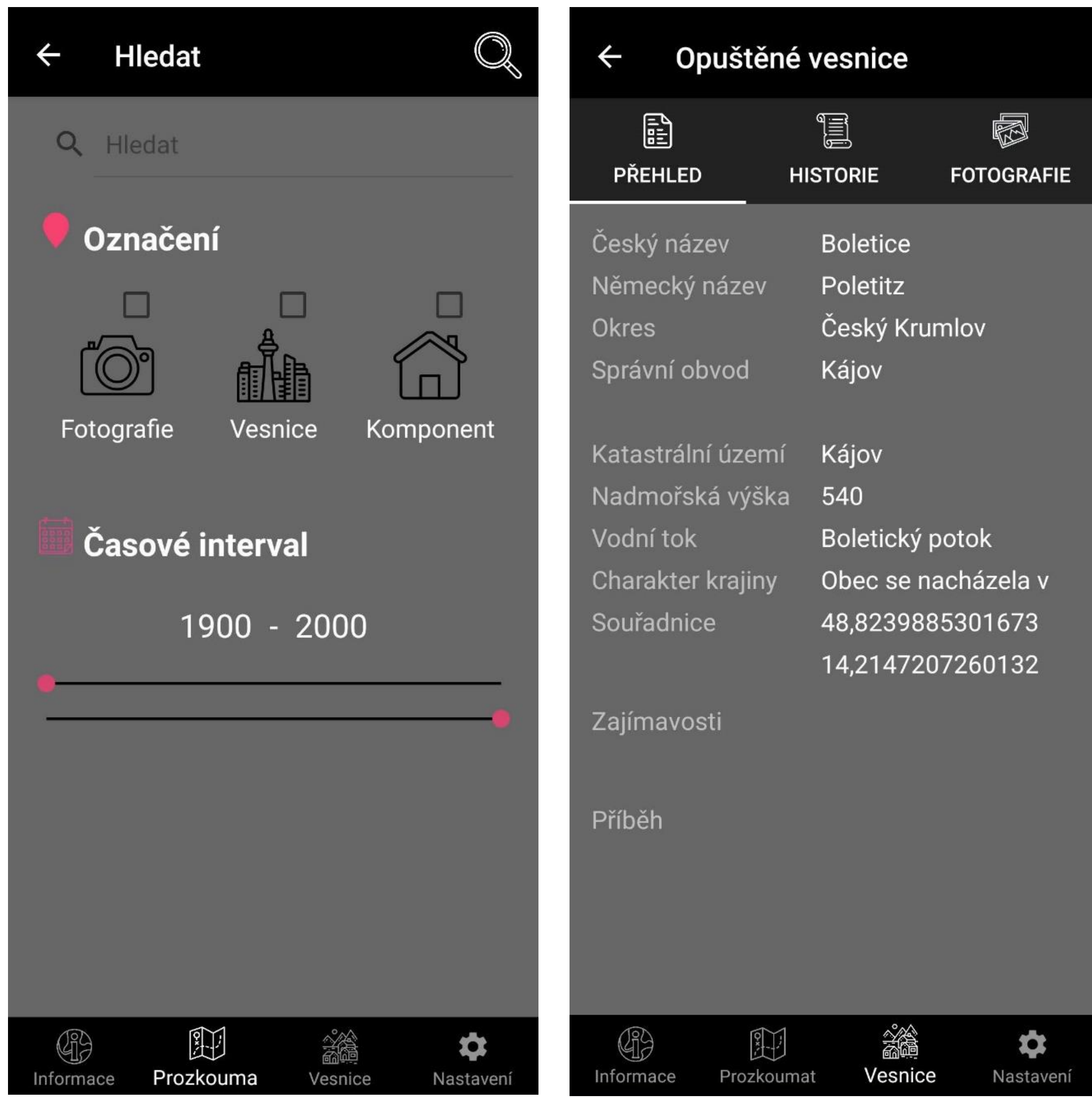
Příloha A – Uživatelské rozhraní



Obrázek 14 - Obrazovka mapa

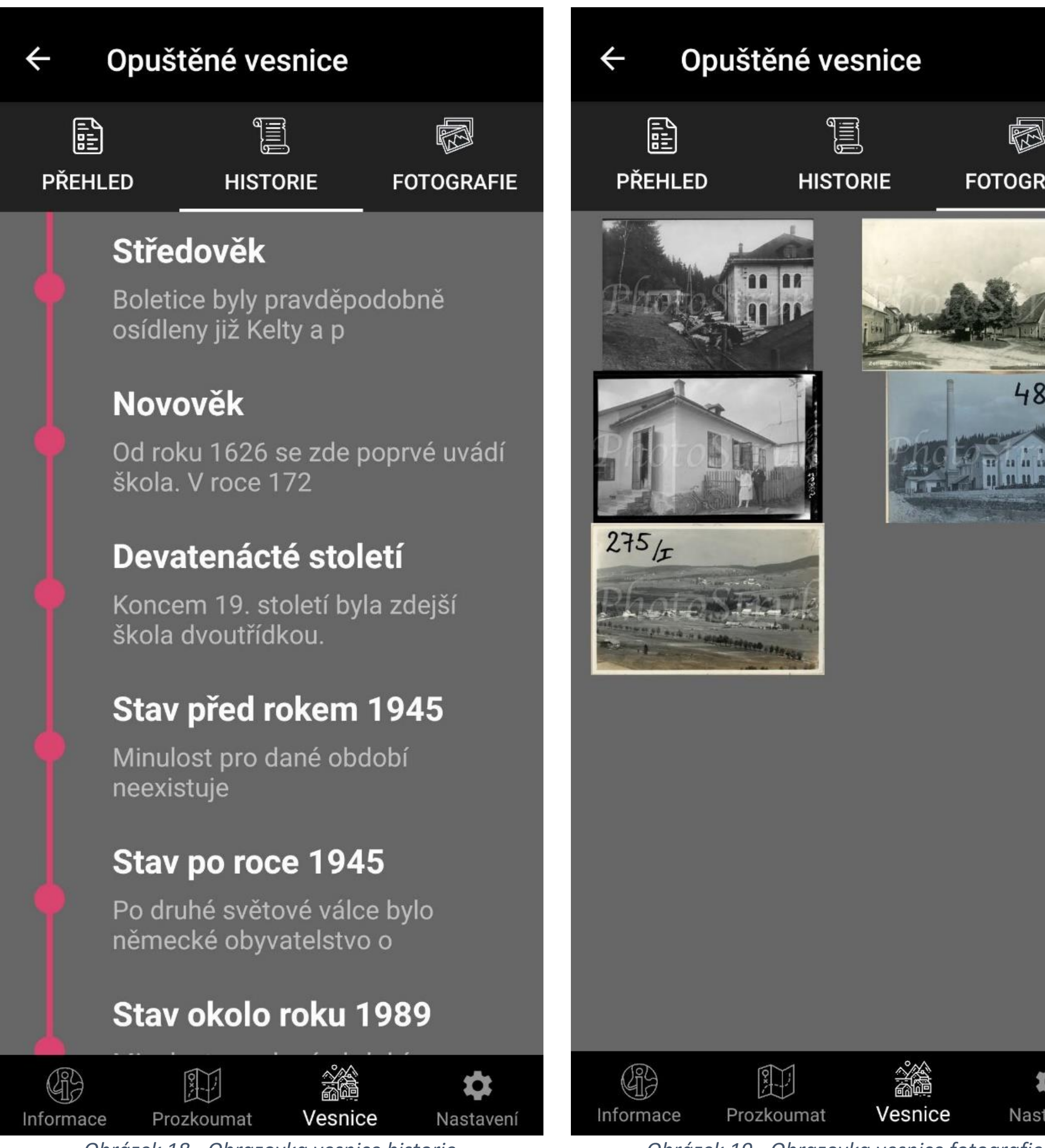


Obrázek 15 - Obrazovka seznam vesnic



Obrázek 16 - Obrazovka filtr

Obrázek 17 - Obrazovka vesnice přehled



Obrázek 18 - Obrazovka vesnice historie

Obrázek 19 - Obrazovka vesnice fotografie

Příloha B – Použitý software

Microsoft Word - https://www.microsoft.com/en/microsoft-365/word

Visual Studio - https://visualstudio.microsoft.com/cs/

ASP.NET Core - https://docs.microsoft.com/en-us/aspnet/core/

Xamarin Forms - https://docs.microsoft.com/en-us/xamarin/xamarin-forms/

Entity Framework - https://docs.microsoft.com/en-us/ef/

Realm - https://realm.io/

GitLab - https://about.gitlab.com/

Ubuntu - https://ubuntu.com/

Azure - https://azure.microsoft.com/en-us/

PostgreSQL - https://www.postgresql.org/

Nginx - https://www.nginx.com/

QGIS - https://qgis.org/

MapTiler - https://www.maptiler.com/

Postman - https://www.postman.com/

pgAdmin - https://www.pgadmin.org/

Visio - https://www.microsoft.com/en/microsoft-365/visio/flowchart-software