Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Virtuální arboretum

Bakalářská práce

Tomáš Svatek

Vedoucí práce: PhDr. Miloš Prokýšek, Ph.D.

České Budějovice 2018

Bibliografické údaje

Svatek Tomáš, 2018: Virtuální arboretum. [Virtual arboretum. Bc. Thesis, in Czech], Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Bakalářská práce se zabývá vytvořením a vývojem mobilní aplikace, která uživateli poskytne informace o všech dostupných dřevinách v jeho blízkosti. V rámci této práce je aplikace vyvíjena pro potřeby operačního systému Android, nicméně je přihlíženo k možnosti budoucí portace i pro systém iOS. Aplikace implementuje několik uživatelských scénářů, například navigaci a kvíz. Součástí práce je analýza, návrh, implementace a testování této aplikace.

Annotation

This bachelor's thesis is dealing with creation and development of a mobile application which provides information about all kinds of trees near the user's location. Under the terms of this thesis, the application is developed for use of Android, nevertheless the future porting to iOS is taken into consideration. Application implements several user scenarios, for instance a navigation and a quiz. The thesis also documents the process of analysing, designing, implementing and testing of this application.

Prohlášení

Prohlašuji, že svoji bakalářskou 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 bakalářské práce, a to – v nezkrácené podobě – elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejich 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 12.12. 2018 .....

Tomáš Svatek

Poděkování

Rád bych toto poděkování věnoval panu PhDr. Miloši Prokýškovi, Ph.D., za jeho cenné rady a čas strávený při konzultacích. Dále mé poděkování patří společnosti SAFE TREES a městu Písek za poskytnutí dat. V neposlední řadě bych rád poděkoval své rodině a přítelkyni za jejich podporu během studia.

Obsah

1 Úvod

Tématem bakalářské práce je tvorba mobilní aplikace, která prostřednictvím geolokace poskytuje uživateli informace o dřevinách v jeho blízkosti. Aplikace využívá několik datových zdrojů, které jí poskytují informace o dřevinách z celé České republiky. Díky tomu má aplikace potenciál pokrýt území celého státu.

Na trhu neexistuje podobná mobilní aplikace, která by poskytovala uživateli informace o všech dřevinách. Naskýtá se tedy příležitost nabídnout uživatelům aplikaci, která na trhu zatím chybí.

V České republice existuje společnost SAFE TREES. Společnost se specializuje na projekční a inženýrskou činnost v oblasti arboristiky, zahradní architektury a technických inventarizací. Jedním z projektů společnosti je portál Stromy pod kontrolou[1]. V rámci tohoto projektu vznikla široká databáze informací o stromech na veřejně přístupných plochách. Společnost na svých webových stránkách uvádí, že eviduje celkem 1 milion stromů. Kromě toho má tato organizace také vlastní klientské API, které není veřejné. Klientské API poskytuje vybraným externím subjektům (aplikacím, službám, ...) přístup k základním údajům o stromech, které jsou zaneseny v databázi portálu Stromy pod kontrolou. Pro aplikaci se klientské API stalo elementárním zdrojem dat. Sekundárním zdrojem jsou pak data z města Písek.

Společnost SAFE TREES má několik aplikací, které jsou určeny primárně pro odborníky. Pro běžné uživatele jsou tyto aplikace příliš složité. Slovním spojením „běžní uživatelé“ je myšlena široká veřejnost, lidé, kteří nemají žádné odborné znalosti v oblasti arboristiky. Běžného uživatele zajímají zejména základní informace o dřevinách jako jsou zeměpisná poloha, taxon, fotografie stromu a jiné další zajímavé vlastnosti dřevin. Odborné informace je vhodné pro jednoduchost používání vynechat a přizpůsobit celou aplikaci široké veřejnosti tak, aby byla snadno pochopitelná.

První část práce bude sloužit jako zadávací dokumentace. Zadávací dokumentace musí nezbytně obsahovat formulaci logického rámce, specifikaci požadavků a návrh architektury aplikace. Ve druhé části se práce musí zabývat popisem procesu vývoje aplikace od kódování, testování až po nasazení do

[1] Portál *Stromy pod kontrolou* je dostupný na: https://www.stromypodkontrolou.cz/

produkčního provozu. Poslední část práce představuje evaluaci, ve které je nutné shrnout, vyhodnotit celý vývoj a následně nastínit budoucí vylepšení a rozšíření aplikace.

1.1 Cíle práce

Hlavním cílem této bakalářské práce je vytvořit mobilní aplikaci, která uživateli poskytne informace o dřevinách v jeho blízkosti. Tyto informace budou poskytovány prostřednictvím několika uživatelských scénářů, které budou navrženy tak, aby samotné použití aplikace bylo pro uživatele maximálně atraktivní a zábavné. V rámci této práce je aplikace vyvíjena pro potřeby operačního systému Android, nicméně je přihlíženo k možnosti budoucí portace i pro systém iOS. Pro naplnění hlavního cíle práce je nezbytné splnit několik dílčích úkolů:

- Analyzovat podobné typy aplikací
- Vytvořit uživatelské scénáře
- Specifikovat funkční a nefunkční požadavky
- Navrhnout architekturu aplikace
- Implementovat aplikace
- Testování aplikace
- Nasadit aplikaci do produkčního provozu

1.2 Abstraktní popis řešení

Softwarové řešení „Virtuální arboretum“ (dále Systém) je technicko-funkčním celkem složeným především z mobilní aplikace (dále Aplikace) a serverové části (dále Back End).

Většina mobilních aplikací a her potřebuje back end službu pro služby a funkce, které nelze provést pouze na zařízení, například sdílení, zpracování dat více uživateli nebo ukládání velkých souborů [1]. Tato Aplikace vyžaduje pro své fungování Back End, který agreguje několik datových zdrojů a udržuje data aktuální. Samotná Aplikace je v podstatě jen tenký klient s GUI. Pokud Aplikace potřebuje načíst či uložit data, požádá o tuto službu Back End. Aby Aplikace mohla komunikovat s Back Endem potřebuje mít stále internetové připojení.

V rámci této práce vznikne Aplikace, která bude dostupná pouze pro uživatele operačního systému Android od verze 6.0 Marshmallow. Nicméně, již při vzniku této práce je přihlíženo k možnosti budoucí portace i pro systém iOS. Hned od začátku je tedy potřeba brát v úvahu fakt, že v budoucnu bude potřeba, mít stejnou aplikaci dostupnou i pro uživatele operačního systému iOS. Kvalitním návrhem architektury aplikace a správným výběrem technologií lze výrazně snížit náklady a čas dalšího vývoje.

2 Analýza

Tato kapitola se zabývá analýzou funkčních a nefunkčních požadavků Systému a jeho specifikací. Pro stanovení základních parametrů projektu byl použit logický rámec, který poskytne základní přehled o realizovaném projektu. Kapitola dále obsahuje popis uživatelských scénářů, ze kterých byly následně odvozeny funkční požadavky na Systém. Druhá část kapitoly se zabývá porovnáním přístupů k vývoji mobilních aplikací.

2.1 Podobné typy aplikací

V roce 2018 vznikla aplikace Na Ovoce[2], která se stala mezi uživateli velmi populární. Podle statistik[3] Google Play ze dne 5.12. 2018 si aplikaci stáhlo již více než 10 000 uživatelů. Aplikace zobrazuje na mapě ovocné stromy. Uživatelé mohou sami přidávat nové stromy do mapy a psát k nim poznámky. Cíl je zřejmý, zjistit, kde se jaké ovoce nachází a následně si ho natrhat. Uživatelé tak mají snadný přístup k čerstvému ovoci a navíc zdarma. Tato aplikace tedy obsahuje obdobnou funkcionalitu jako Systém ve smyslu sdílení informací o poloze dřevin v krajině. Data jsou získávána uživateli (crowdsourcing data collection).

Další řešenou službou je produkt společnosti SAFE TREES, který prezentuje data stromů na webu svého projektu Stromy pod kontrolou[4]. Stromy jsou zobrazeny v mapě pomocí markerů a clusterů. Společnost také disponuje vlastní mobilní aplikací. Nicméně, tato aplikace je určena pro sběr a management dat stromů.

[2] Webová prezentace aplikace *Na Ovoce* je dostupná na: https://na-ovoce.cz/web/

[3] https://play.google.com/store/apps/details?id=com.naovoce.naovoce

[4] *Stromy pod kontrolou* mapa stromů: https://www.stromypodkontrolou.cz/map

2.2 Metodika vývoje

Vývoj mobilní aplikace má oproti desktopovým aplikacím mnohem kratší životní cyklus a vývojový životní cyklus. Jsou na ni kladeny časté požadavky na změny od uživatelů a potřebuje častější aktualizace [2]. Vývoj aplikace se tedy téměř nikdy nezastaví a je důležité umět pružně reagovat na požadované změny.

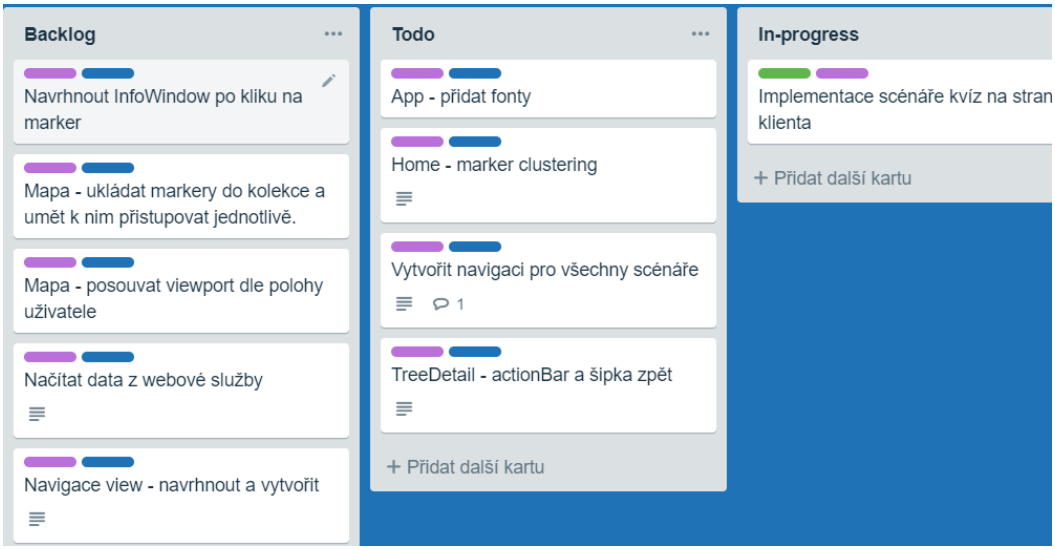
Tradiční metodika vodopád, která je založena na sekvenčním přístupu – jednotlivé fáze vývoje se neopakují, není tedy příliš vhodná pro mobilní aplikace. Mnohem vhodnější jsou agilní metodiky, které jsou založeny na iterativním a inkrementálním přístupu. Snaží se co nejlépe reflektovat zákazníkovy požadavky pravidelnou dodávkou software.

Vývoj Systému probíhal dle Kanbanu. Kanban těžko nazvat metodikou, je to spíš sada principů regulující workflow tak, aby každý věděl, v jakém stavu jsou rozdělané úkoly, minimalizovalo se množství nedodělané práce a posílila se samostatnost týmu. Vizualizace workflow je pro *Kanban* zcela klíčová [3]. Kanban využívá tabuli rozdělenou na sloupce a karty pro vizualizaci workflow. Každý sloupec reprezentuje stav, ve kterém se daná karta nachází. Karta může reprezentovat úkol, uživatelský příběh, chybu nebo cokoli jiného. Jaké má mít tabule, sloupce a druhy karet, není striktně definováno. Každý jednotlivec či tým si může přizpůsobit vzhled a strukturu tak, aby co nejlépe odpovídala jeho požadavkům.

Vývoj Systému byl řízen pomocí nástroje *Trello*[5], který používá pro řízení projektů právě *Kanban* a je k dispozici zcela zdarma. Tabule je rozdělena do pěti sloupců – *backlog, todo, in-progress, test* a *done.* Sloupec *in-progress* může obsahovat maximálně dvě karty. Toto omezení má za cíl omezit množství rozdělané práce. Vede vývojáře k tomu, aby nejprve dokončil rozdělaný úkol, než začne pracovat na novém.

[5] https://trello.com/

Na začátku se *backlog* naplní kartami – úkoly. Postupně se vybírají karty do sloupce *todo,* které budou zpracovány v dalším sloupci *in-progress*, ale až poté, co se v něm uvolní místo. Potom, co jsou karty zpracovány, přesouvají se do sloupce *test*, kde čekají na testování. Konec životního cyklu karty je ve sloupci *done*, kde se nachází zpracované a úspěšně testované karty. Zelená barva karty reprezentuje uživatelské příběhy, červená barva chyby a modrá barva úkoly.



Obrázek 1 - Trello nástěnka

2.3 Logický rámec

Cíl projektu	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	
Vytvořit mobilní aplikaci pro platformu Android.	*Funkční mobilní aplikace.*	*Mobilní aplikace je dostupná v Google Play Store.*	
Účel projektu	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	**Předpoklady**
Poskytnout uživatelům přehled a informace o dřevinách v jejich blízkosti prostřednictvím mobilní aplikace pracující s polohovacími službami.	*Zájem o aplikaci mezi uživateli. Počet stažení přesáhne 25 stažení během prvních pěti týdnů po nasazení aplikace do Play Store.*	*Statistiky stažení z Google Play Store.*	*O aplikaci bude mezi uživateli zájem.*

Výstupy projektu	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	**Předpoklady**
1. zadávací dokumentace *2. mobilní aplikace* *3. REST API* *4. uživatelská dokumentace*	*zadávací dokumentace* *Funkční mobilní aplikace umístěna v Google Play Store* *uživatelské dokumentace*	*Do konce listopadu 2017 budou implementované první tři scénáře.* *Do konce března bude aplikace připravena k uživatelskému testování.*	*Implementace systému bude probíhat bez vážnějších problémů.* *Podaří se získat dostatek kvalitních dat.*

Aktivy projektu	**Prostředky / Vstupy**	**Harmonogram**	**Předpoklady**
1.1 analýza požadavků *1.2 návrh řešení* *2.1 implementace jádra aplikace* *2.2 implementace rozšiřujících částí* *2.3 testování aplikace* *3.1 návrh REST API* *3.2 implementace API* *4.1 vytvoření uživatelské dokumentace*	*Dostatek kvalifikovaných lidí* *Webový server* *Vývojové a testovací prostředí* *Mobilní zařízení pro testování aplikace*	*1. 10/2017* *2. 10/2017* *3. 11/2017 - 12/2017* *4. 03/2018*	*Dostatek časových a finančních zdrojů.*

2.4 Specifikace softwarových požadavků

Začátkem vývoje každého software je specifikace softwarových požadavků. Tyto požadavky lze zachytit hned několika způsoby. Nejčastějším způsobem jsou případy užití, které ale nejsou příliš efektivní, pokud systém nebo aplikace nemá alespoň jednoho aktéra. Z toho důvodu nejsou případy užití v analýze Systému obsaženy.

2.4.1 Funkční požadavky

Na počátku analýzy požadavků bylo vytvořeno několik uživatelských scénářů a každému z nich byla přiřazena priorita. Priorita určuje, v jakém pořadí budou jednotlivé scénáře implementovány.

První tři scénáře byly označeny za jádro Aplikace. Detailní popis uživatelských scénářů se nachází dále v textu. Funkční požadavky se nevztahují přímo na Aplikaci, ale na celý Systém, protože vyžadují implementaci jak na straně Aplikace, tak na straně Back Endu. Z těchto scénářů vychází tyto funkční požadavky:

1. Systém bude sledovat aktuální GPS polohu uživatele.
2. Systém bude zobrazovat všechny stromy v uživatelově oblasti.
3. Systém bude vyhledávat strom, který je nejblíže k uživateli.
4. Systém bude vykreslovat stromy v mapě.
5. Systém bude zobrazovat detail stromu.
6. Systém bude zobrazovat dendrologické informace.
7. Systém bude zjišťovat GPS pozici nejbližšího stromu vyhovujícího zadaným parametrům.
8. Systém bude schopen navigovat uživatele k vybranému stromu.
9. Systém bude schopen nalézt zadaný počet stromů a zjistit jejich GPS souřadnice.
10. Systém bude zaznamenávat uživatelovy odpovědi.
11. Systém bude vyhodnocovat uživatelovy odpovědi.
12. Uživatel bude moci přidávat nové stromy.
13. Uživatel bude moci editovat stromy.

2.4.2 Nefunkční požadavky

1. Aplikace bude dostupná pro uživatele operačního systému Android.
2. Aplikace bude pracovat v online režimu.
3. Aplikace bude mít jednoduché a intuitivní uživatelské rozhraní.
4. Aplikace bude načítat data z REST rozhraní.
5. REST rozhraní bude agregovat data z více datových zdrojů.
6. Systém bude možno jednoduše rozšířit o novou funkcionalitu.
7. Zdrojový kód bude čitelný, udržovatelný a přehledně zdokumentovaný.

2.5 Uživatelské scénáře

Jádro Aplikace tvoří první tři uživatelské scénáře, které byly navrženy na začátku projektu. V každé tabulce se nachází analýza scénáře a jeho popis.

Název: Nejbližší strom	**Priorita:** Nezbytná	**Pořadí implementace:** 1
Popis: Uživatel chce najít nejbližší strom ve svém okolí. Zmáčkne tlačítko s titulkem „nejbližší strom“. Aplikace vyhledá nejbližší strom a označí ho v mapě. Uživatel si může zobrazit detail tohoto stromu, který obsahuje všechny dostupné informace.		
Funkční požadavky: 1. Systém bude sledovat aktuální GPS polohu uživatele. 3. Systém bude vyhledávat strom, který je nejblíže k uživateli. 5. Systém bude zobrazovat detail stromu.		
Předpoklady: o Uživatel udělil Aplikaci přistupovat k aktuální poloze. o Uživatel má přístup na internet.		

Název: Navigace	**Priorita:** Nezbytná	**Pořadí implementace:** 2
Popis: Uživatel si přeje vyhledat nejbližší konkrétní strom. Např. si přeje najít dub. Zadá tedy do aplikace, že chce najít nejbližší dub. Aplikace tento dub vyhledá a vykreslí do mapy nejkratší cestu k cíli.		
Funkční požadavky: 1. Systém bude sledovat aktuální GPS polohu uživatele. 3. Systém bude vyhledávat strom, který je nejblíže k uživateli. 4. Systém bude vykreslovat stromy v mapě. 7. Systém bude zjišťovat GPS pozici nejbližšího stromu vyhovujícího zadaným parametrům. 8. Systém bude umět navigovat uživatele k vybranému stromu.		
Předpoklady: o Uživatel udělil Aplikaci přístup k aktuální poloze. o Uživatel má přístup na internet.		

Název: Knihovna	**Priorita:** Nezbytná	**Pořadí implementace:** 3
Popis: Uživatel si může postupně přečíst informace o jednotlivých druzích stromů. Graficky to bude působit tak, jako kdyby listoval v knize.		
Funkční požadavky: 6. Systém bude zobrazovat dendrologické informace.		
Předpoklady: o Uživatel má přístup na internet.		

Název: Kvíz	**Priorita:** Nezbytná	**Pořadí implementace:** 4

Popis:

Před spuštěním kvízu je uživateli vysvětlen průběh kvízu. Následně uživatel vybere počet stromů v kvízu. Aplikace poté vybere zadaný počet stromů v uživatelově okolí. Uživatel je poté postupně navigován k vybraným stromům. Jakmile se nachází v blízkosti stromu (cca 1-10 metrů), aplikace ho vyzve k rozpoznání stromu. Poté aplikace vyhodnotí uživatelovu odpověď a aktualizuje skóre kvízu. V případě, že uživatel neprošel všechny stromy, celá posloupnost se opakuje. Počet stromů v kvízu je limitován množstvím stromů v oblasti, ve které se uživatel nachází.

Funkční požadavky:

1. Systém bude sledovat aktuální GPS polohu uživatele.
3. Systém bude vyhledávat strom, který je nejblíže k uživateli.
4. Systém bude vykreslovat stromy v mapě.
8. Systém bude umět navigovat uživatele k vybranému stromu.
9. Systém bude umět nalézt zadaný počet dřevin a zjistit jejich GPS souřadnice.
10. Systém bude zaznamenávat uživatelovy odpovědi.
11. Systém bude vyhodnocovat uživatelovy odpovědi.

Předpoklady:

- Uživatel udělil Aplikaci přístup k aktuální poloze.
- Uživatel má přístup na internet.

2.6 Mobilní technologie

Výběr správných technologií při vývoji mobilní aplikace je velmi důležitým rozhodnutím, které výrazně ovlivňuje rychlost a cenu vývoje.

V současnosti dominují na trhu chytrých telefonů dva operační systémy. Operační systém Android, využívaný téměř všemi výrobci chytrých telefonů, který má, dle podílu na trhu[6] v roce 2018, na svém chytrém telefonu 84.8 % uživatelů. Druhý dominantní systém je iOS, kterým vybavuje své chytré telefony výhradně společnost Apple, využívá 15.1 % uživatelů. Zbylou desetinu procenta tvoří ostatní operační systémy.

Před vznikem Aplikace je nutné rozhodnout, na jakých platformách bude aplikace dostupná. Na základě toho poté zvolit technologie a způsob vývoje. Existují tři základní postoje, jak k vývoji Aplikace přistoupit:

1. Nativní mobilní aplikace (Java, Kotlin, Swift nebo Objective-C)
2. Multiplatformní aplikace
    - Hybridní HTML5 aplikace (Ionic, Cordova, PhoneGap)
    - Hybridní nativní aplikace (Xamarin, React native)
3. Webová stránka optimalizovaná pro chytré telefony

2.6.1 Nativní mobilní aplikace

Nativní mobilní aplikace je vytvořena pro konkrétní platformu a je vyvíjena v programovacím jazyce, který platforma definuje [4]. Nativní iOS aplikace lze vyvíjet pomocí programovacích jazyků Objective-C nebo Swift. Nativní Android aplikace jsou vyvíjeny pomocí programovacího jazyka Java.

Nativní aplikace jsou vyvinuty a kompilovány pomocí stejného programovacího jazyka a využívají API jako platforma, na které běží [5]. Díky tomu běží mnohem rychleji než multiplatformní mobilní aplikace. Další výhody jsou vynikající uživatelský prožitek, podpora nejnovější API[7] a jednoduchá rozšiřitelnost.

[6] https://www.idc.com/promo/smartphone-market-share/os

[7] Nativní aplikace má přístup k interním API zařízení. Může využívat například lokální uložiště, fotoaparát a vibrace.

Hlavní nevýhoda plyne ze samotné definice nativní aplikace. Nelze ji spustit na jiných platformách.

Pokud tedy vznikne požadavek, aby aplikace byla nativní a běžela na více platformách, bude nutné vytvořit aplikaci pro každou platformu pomocí nástrojů a technologií, které platforma definuje ve svém SDK[8]. Vývoj takové aplikace bude velmi dlouhý a drahý.

2.6.2 Multiplatformní aplikace

Multiplatformní aplikace je mobilní aplikace, která je kompatibilní s více platformami a může být spuštěna na smartphonu či tabletu [6]. Filozofii multiplatformních aplikací vystihuje oblíbené heslo většiny multiplatformních frameworků „napiš jednou, spusť kdekoliv“. Zachycuje největší výhodu multiplatformních aplikací, kterou je sdílení určitého množství kódu napříč různými platformami. Množství sdíleného kódu závisí na konkrétním frameworku a typu aplikace. Vývoj je mnohem rychlejší než u nativních aplikací. U jednodušších aplikací v kombinaci se správným frameworkem lze sdílet až 99 % kódu. Frameworků pro vývoj multiplatformních aplikací existuje celá řada a je velmi těžké vybrat ten správný. Multiplatformní frameworky lze rozdělit na hybridní nativní aplikace a hybridní webové aplikace, někdy také nazývané jako HTML5 aplikace.

Hlavní myšlenkou hybridních webových aplikací je vytvořit web, který bude vypadat stejně jako mobilní aplikace. Hybridní webové aplikace jsou stejné jako jakékoliv jiné aplikace nainstalované v mobilním zařízení. Musejí se instalovat a lze je najít v App Store. Aplikace jsou vytvořeny pomocí webových technologií HTML5, CSS a JavaScriptu. Hlavní rozdíl je v tom, že hybridní webové aplikace jsou hostované uvnitř nativní aplikace, která využívá mobilní platformu WebView. WebView si lze představit jako okno prohlížeče, které je nastavené tak, aby běželo v režimu celé obrazovky. WebView umožnuje přistupovat k API zařízení jako je akcelometr, fotoaparát, kontakty a další. K těmto API nemá obvykle mobilní webový prohlížeč přístup [7]. JavaScript je podporován všemi prohlížeči, hybridní webové aplikace tedy mohou běžet na jakémkoliv mobilním zařízení, které má nainstalovaný WebView.

[8] *SDK* je sada vývojových nástrojů umožňující vytváření aplikací pro jisté platformy.

Tyto technologie jsou velmi dobře známé webovým vývojářům. To je velká výhoda, protože najmout webové vývojáře je snadnější, mnohdy i levnější než vývojáře Java, Switft, Objective-C nebo C# [8]. Nevýhodou těchto aplikací je jejich výkon a přístup ke specifickým nativním API zařízení. Nejznámějšími frameworky pro budování hybridních webových aplikací jsou Cordova, Ionic a PhoneGap.

Hybridní nativní aplikace jsou téměř nerozeznatelné od nativních aplikací. Používají nativní UI kontrolky, nativní API zařízení. Na rozdíl od hybridních webových aplikací neběží ve WebView. Nejznámějšími zástupci jsou Xamarin a React Native.

3 Design

Tato kapitola se zabývá návrhem konceptu Systému. Jsou zde wireframy, které popisují funkčnost a obsah jednotlivých obrazovek Aplikace. Dále je představena architektura celého Systému. Na konci kapitoly jsou popsány technologie, které byly použity při vývoji.

3.1 Wireframes

Důležitým nástrojem ve fázi analýzy jsou drátěné modely neboli wireframes. Jednoduše lze pomocí nich definovat funkci a obsah jednotlivých obrazovek Aplikace. Tato Aplikace se skládá ze čtyř obrazovek. Wireframy hlavních obrazovek Aplikace jsou na obrázku 2.

MainScreen

Jedná se o hlavní obrazovku celé Aplikace. Primární funkcí této obrazovky je zobrazit stromy v okolí uživatele. Stromy jsou reprezentovány v mapě pomocí markerů. Každý druh stromu má svoji barvu. Po stisknutí markeru se zobrazí informační okno, ve kterém je uvedeno, o jaký druh stromu se jedná. Pokud uživatel stiskne informační okno, Aplikace ho přesměruje na DetailScreen.

DetailScreen

V této obrazovce se nachází veškeré dostupné informace o stromu, který uživatel vybral v mapě. Informace o stromech mohou být odlišné, protože jsou závislé na zdroji dat, ze kterého byly získány. Obrazovka tedy může vypadat jinak, ve smyslu množství zobrazených informací, zvláště pro stromy z různých datových zdrojů. Nicméně, každý strom by měl obsahovat svoje druhové jméno v českém i latinském jazyce. Všechny informace o stromu jsou umístěny v komponentě karty.

AddScreen

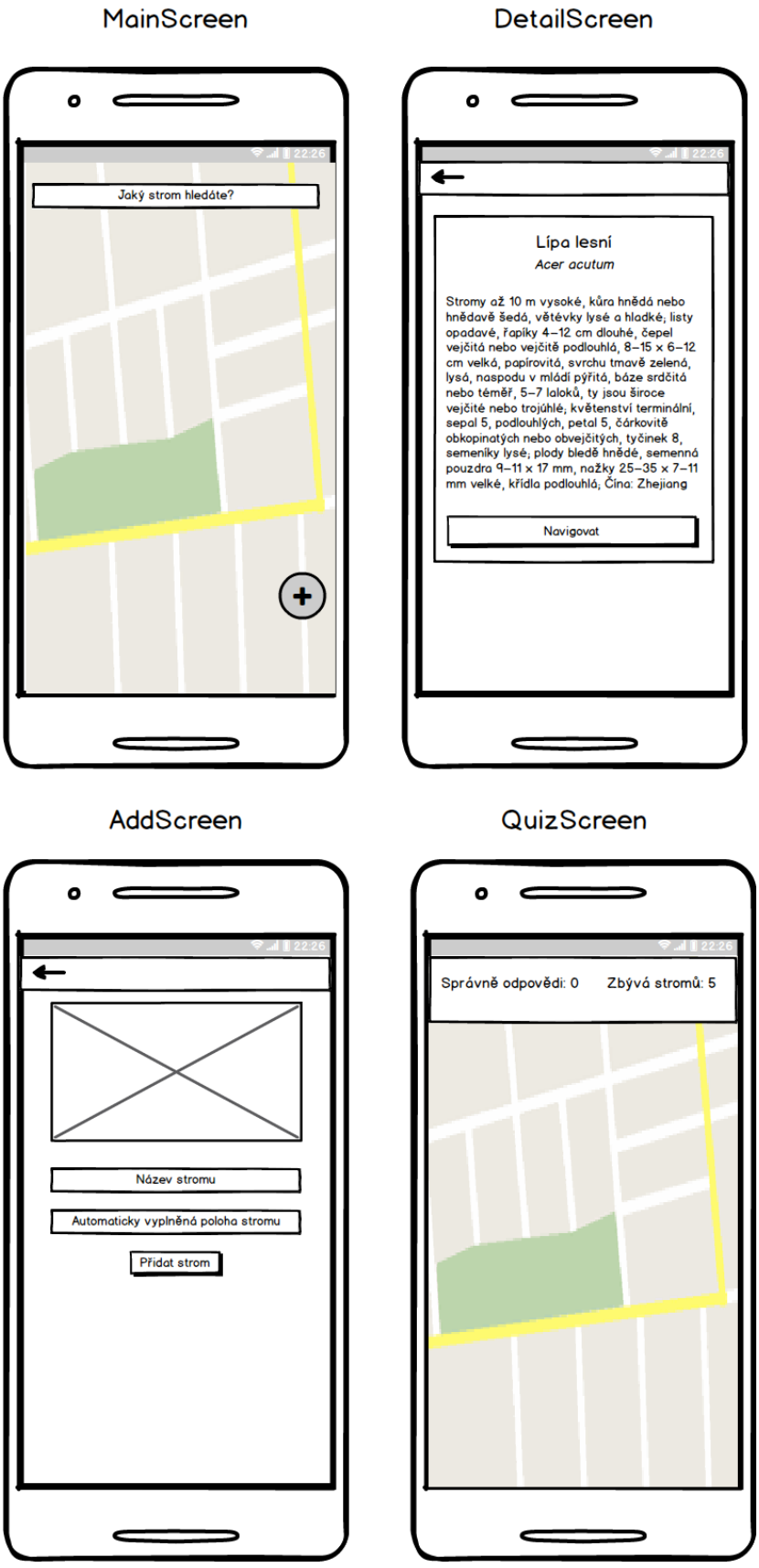
Obrazovka slouží ke sběru nových stromů. Uživatel musí vyplnit pouze český název stromu. Atribut poloha stromu je automaticky předvyplněn podle aktuální polohy uživatele a zobrazen v čitelné podobě.

QuizScreen

Tato obrazovka je implementací uživatelského scénáře Kvíz. V horní části obrazovky se nachází informace o průběhu spuštěného kvízu. Ve zbytku obrazovky je mapa, kde jsou vykresleny stromy, které uživatel bude v kvízu rozeznávat. Během kvízu je nutné vyzvat uživatele k rozpoznání stromu. Uživatel je vyzván k rozpoznání ve chvíli, kdy se nachází u stromu. V tu chvíli se zobrazí modální okno, do kterého uživatel zadá jméno stromu, který má před sebou.

LearningScreen

Obrazovka je implementací uživatelského scénáře Knihovna. Obrazovka slouží zejména pro edukaci uživatele v oblasti dendrologie. Wireframe této obrazovky vychází z obrazovky DetailScreen. Nicméně s tím rozdílem, že v této obrazovce je potřeba plynule přecházet mezi jednotlivými kartami stromů. Plynulého přechodu lze dostáhnout pomocí animace, která připomíná otočení listu v knize.



Obrázek 2 - wireframy jednotlivých obrazovek Aplikace

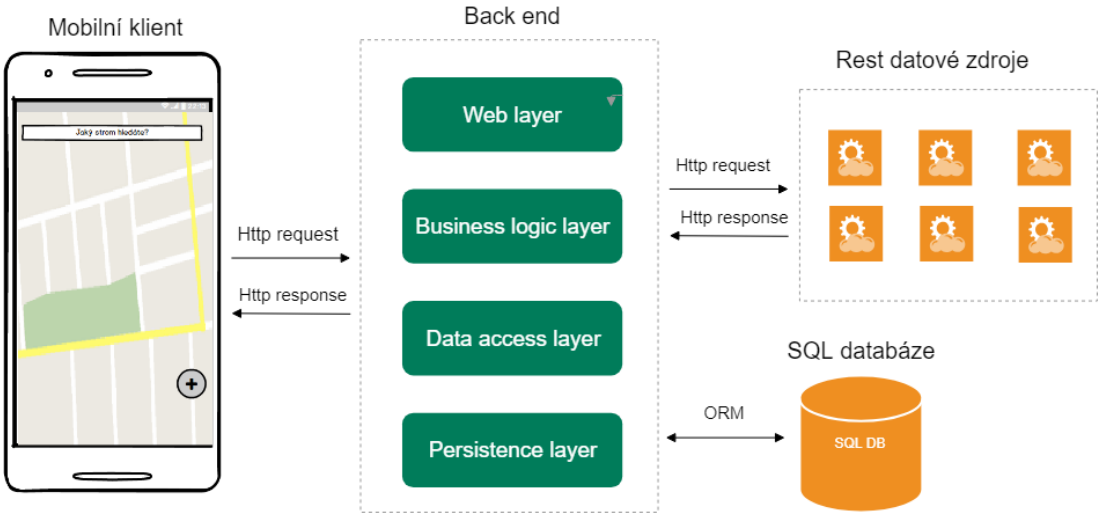
3.2 Architektura systému

Návrh architektury je klíčová činnost, které se vyplatí věnovat větší úsilí a čas. Správně navržená architektura ušetří v budoucnu velké množství času při rozšiřování aplikace [9].

Architektura systému byla navržena tak, aby splňovala následující požadavky:

- Škálovatelnost
- Nezávislost na frameworku
- Nezávislost na databázi
- Nezávislost na UI

Škálovatelnosti neboli rozšiřitelnosti Systému lze dosáhnout pomocí uplatnění několika principů. Jedním z principů je udržet volnou vazbu mezi jednotlivými jednotkami Systému. Jednotkou může být třída, modul nebo vrstva Systému. Dalším principem je vysoká koheze. Každá jednotka Systému má být zodpovědná právě za jednu činnost. Tyto principy jsou zahrnuty nebo se prolínají s jednou z nejznámějších sadou návrhových principů uváděnou pod akronymem SOLID. Všechny tyto principy byly aplikovány při návrhu Systému.



Obrázek 3 - vícevrstvá architektura systému

Na obrázku číslo 3 je vyobrazena architektura vyvíjeného Systému. Klíčovou částí je Back End, který je navržen dle typické vícevrstvé architektury. Back End je rozdělen do čtyř vrstev. Každá vrstva má svoji zodpovědnost. Nejvyšší vrstvou je webové rozhraní, které je implementováno dle architektury REST.

Back End využívá několik datových zdrojů, mezi které patří data z města Písek a klientské API společnosti SAFE TREES. Data access layer je zodpovědná za komunikaci s těmito zdroji. Data se mohou nacházet v SQL databázi nebo jsou přístupná přes webové rozhraní. Nové datové zdroje mohou v budoucnu přibývat.

3.3 Datový model

Ačkoli Back End pracuje s velkým množstvím dat (až 1 milion záznamů), datový model je velmi jednoduchý. Obsahuje pouze dvě entity tree a dendrology. Entita tree reprezentuje jeden konkrétní strom. Entita dendrology reprezentuje rodové jméno a doplňující informace o druhu stromu. Vztahy mezi entitami jsou následující:

- Strom musí mít právě jedno rodové jméno.
- Rodové jméno může mít nula nebo více stromů.

V budoucnu může být datový model rozšířen o entitu user, která bude reprezentovat uživatele aplikace.

3.4 Použité technologie

V této kapitole jsou stručně popsány technologie, které byly použity při vývoji Systému. Kapitola je rozdělena na dvě části. V první části jsou popsány technologie, které byly použity pro vytvoření Back Endu. V druhé části jsou popsány technologie použité při vývoji Aplikace.

3.4.1 Back end

Back End je postaven na vývojářské platformě .NET vyvinuté společností Microsoft. Pomocí .NET platformy lze vytvořit jakoukoliv aplikaci pro jakoukoliv platformu. Velkou výhodou je unifikované prostředí poskytnuté Microsoftem. Microsoft nastavil určité standarty v podobě frameworků a nástrojů. Vývojář tedy

nemusí vybírat mezi nepřeberným množstvím frameworků, nástrojů a vývojových prostředí.

Platforma .NET byla vybrána, zejména kvůli tomu, že autor této práce, má s platformou kladné zkušenosti z předchozích projektů.

ASP.NET CORE

ASP .NET Core je platformě univerzální, vysoce výkonný framework typu open-source, který slouží k vytváření moderních cloudových aplikací připojených k internetu [10]. Framework poskytuje funkce pro budování webových aplikací a REST API. Celý framework je založen na architektonickém vzoru MVC[9], který se stal jakýmsi standardem webových aplikací napříč všemi webovými frameworky. Na rozdíl od svého předchůdce ASP.NET MVC, který je postaven na .NET Frameworku, je ASP.NET Core multiplatformní. Může běžet na platformách Windows, Linux a macOS.

Entity Framework Core

Entity Framework Core (EF Core) je nejnovější verze Microsoftem doporučené ORM technologie postavené nad platformou .NET Core. Framework byl navržen tak, aby byl odlehčený, rozšiřitelný a multiplatformní. Jedná se v podstatě o objektově relační mapovač. Objektově relační mapování je technika, která umožňuje vývojářům pracovat s daty uloženými v relační databázi jako s objekty [11]. EF Core si lze představit jako rozhraní mezi světem databázových tabulek a objektů. Dotazování dat se provádí pomocí dotazovacího jazyku LINQ[10]. Jedná se o integrovaný dotazovací jazyk, který byl poprvé přidán do .NET Frameworku 3.5.

3.4.2 Klientská část

Tato Aplikace je postavena na frameworku React Native. Tato technologie byla vybrána zejména kvůli možnostem multiplatformního vývoje. Pomocí React Native je možné napsat nativní mobilní aplikaci, jen za použití programovacího jazyku JavaScript.

[9] MVC – Model-View-Controller (model - pohled - kontrolér)
[10] LINQ – Language Integrated Query.

React Native

React Native je JavaScriptový framework pro vytváření nativních mobilních aplikací platformy iOS a Android [12]. Je postaven na webovém frameworku React[11] od společnosti Facebook. Stejně jako jeho webový kolega React Native je založen na komponentovém přístupu. Hlavní myšlenka Reactu je vytvářet nezávislé, znovupoužitelné celky (komponenty), které budou mezi sebou komunikovat.

React Native se velmi liší od mobilní webové aplikace. Na rozdíl od mobilních webových aplikací dokáže vykreslovat nativní UI kontrolky a má přístup k nativním API konkrétní platformy.

Redux

Redux je deterministický stavový kontejner pro JavaScriptové aplikace. Pomáhá vytvářet aplikace, které se chovají konzistentně, běží v různých prostředí a jsou snadno testovatelné [13].

Zavedením Reduxu se velmi zvýší komplexnost celé aplikace, proto samotní tvůrci knihovny doporučují Redux zavést až ve chvíli, kdy je to skutečně nutné. React Native má vlastní vnitřní mechanismus pro řízení stavů aplikace. Nicméně, tento mechanismus je nedostačující pro aplikace, které potřebují sdílet data mezi více komponenty. V těchto případech přichází na řadu Redux.

Redux popisuje stav aplikace jako čistý JavaScriptový objekt. Změna jakéhokoliv stavu probíhá pomocí vyslání akce. Akce je čistý JavaScriptový objekt, který popisuje, co se právě v aplikaci stalo. Pro navázání stavu na akci slouží funkce zvané Reducery. Jsou to funkce, které přijímají stav a akci ve svých parametrech a vrací nový stav aplikace. Stav aplikace je držen v objektu zvaném Store. Na Store jsou navázány jednotlivé komponenty. Komponenty mají přístup ke stavu aplikace a při každé změně stavu je jim zaslán stav nový [13].

[11] https://reactjs.org/

Expo

Expo je nejjednodušší způsob, jak začít vyvíjet React Native aplikace. Umožňuje začít vyvíjet nový projekt bez instalace nebo konfigurace Android Studia a Xcode. Tyto nástroje se používají pro nativní vývoj [14]. Expo je open source sada nástrojů postavena okolo React Native. Expo rozšiřuje React Native o UI komponenty, které využívá většina mobilních aplikací [15].

Expo aplikace může obsahovat pouze JavaScript kód. Pokud vývojář potřebuje nativní modul nebo API, které Expo nepodporuje, nebo chce jen svoji aplikaci rozšířit o vlastní nativní modul, pak musí provést proces zvaný ejecting. Ejecting je zjednodušeně řečeno převedení Expo projektu na standartní nativní projekt. Tento projekt je poté vyvíjen pomocí Android Studia nebo Xcode. Projekt využívá knihovnu ExpoKit, která umožňuje nadále používat Expo i ve standartních nativních projektech.

4 Implementace

V této kapitole je podrobně popsán průběh implementace celého Systému. Na začátku kapitoly je uvedeno několik zásad, které byly striktně dodržovány při kódování. Poté autor podrobně popisuje implementaci Back Endu. Následuje popis Aplikace. Popis jednotlivých celků Systému je doplněn o několik screenshotů zdrojového kódu.

4.1 Zásady vývoje

- Pojmenování proměnných podle konvencí daného programovacího jazyka
- Pojmenování proměnných podle jejich významu
- Každá třída je umístěna ve vlastním souboru se stejným jménem
- Dokumentovat složité metody a fragmenty kódu
- Dodržování principů SOLID[12], KISS[13], DRY [14] a YAGNI[15]
- Po dokončení nové funkčnosti práci zahrnout do verzovacího systému Git[16]

4.2 Back end

Data, se kterými Back End pracuje, byla získána z několika zdrojů. Ve fázi vývoje Systému jsou tyto zdroje dva. Datový set z města Písek, který byl získán ve formátu .xls a importován do SQL databáze. Druhý datový zdroj je klientské webové API společnosti SAFE TREES, které komunikuje se svými klienty pomocí HTTP protokolu. Další datové zdroje mohou postupně přibývat. Tuto skutečnost je nutné brát v úvahu a navrhnout Back End tak, aby přidání nového datového zdroje bylo jednoduché a mělo co nejmenší dopad na ostatní celky.

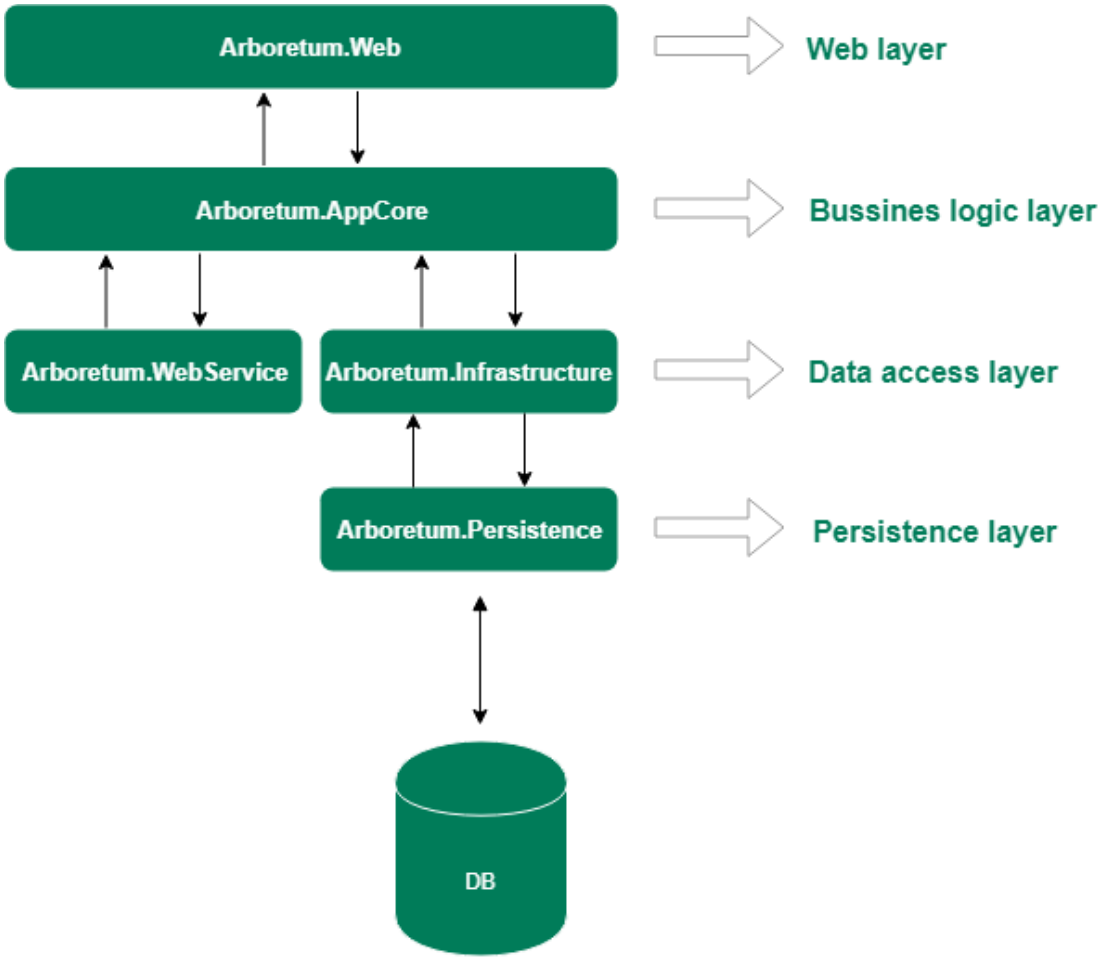
Hlavním úkolem Back Endu je všechny dostupné datové zdroje agregovat a poskytnout svým klientům jednotné rozhraní pro přístup k datům. Komunikace mezi Aplikací a Back Endem je velmi jednoduchá a přímočará. Aplikace pošle požadavek na Back End, ten požadavek zpracuje a zašle Aplikaci odpověď. Odpovědí můžou být data ve formátu JSON, nebo pouze stavový kód HTTP. Služby Back Endu může

[12] SOLID je akronym pro sadu pěti návrhových principů, které vytvořil Rober C. Martin.
[13] KISS – Keep It Simple Stupid (Udělej to jednoduše, hlupáku!)
[14] DRY – Don't repeat yourself (Neopakuj se)
[15] YAGNI – You aren't gonna need it (Nebudeš to potřebovat)
[16] Git je distribuovaný systém správy verzí vytvořený Linusem Torvaldsem.

využívat jakákoli klientská aplikace, která je schopna komunikovat pomocí HTTP protokolu.

4.2.1 Popis architektury

Back End je rozdělen do čtyř vrstev. Každá vrstva má přesně definovanou odpovědnost a rozhraní. Na obrázku číslo 4 je diagram, který ukazuje rozložení jednotlivých vrstev. Data access layer je rozdělena do dvou samostatných projektů (assembly).



Obrázek 4 - Back End

Arboretum.Web

Arboretum.Web je webové rozhraní, které je implementováno dle architektury REST[17]. REST API vystavuje sadu endpointů[18]. Každý endpoint jednoznačně identifikuje resource[19] a operace, které lze nad ním provádět. Příklad takového endpointu je api/trees, který je vidět na obrázku číslo 5. Tento endpoint načítá všechny stromy v zadané oblasti. Oblast je specifikována v query stringu[20].

V této vrstvě se nenachází žádná business logika. Jsou zde pouze API kontroléry a viewmodely. Úkolem této vrstvy je přijmout HTTP request, zavolat příslušnou službu a poté vrátit HTTP response. Služba vykoná veškerou práci a poté vrátí výsledek své práce *ServiceResult*. Kontrolér na základě tohoto výsledku pošle příslušnou HTTP response.

```
[HttpGet] // api/trees
0 references | Tomáš Svatek, 5 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<IActionResult> GetTrees([FromQuery] VisibleRegionViewModel visibleRegion)
{
    var result = await _treeService.GetTreesAsync(visibleRegion);
    if (result.HasViolations)
    {
        return BadRequest(result.ToString());
    }

    var trees = result.Data;
    var vm = ViewModelMapper.MapTreeToViewModel(trees);

    return Ok(vm);
}
```

Obrázek 5 - GetTrees endpoint

Arboretum.AppCore

Jedná se o hlavní vrstvu Back Endu. Obsahuje modely, business logiku a agreguje data získaná z datových zdrojů. Vystavuje vyšším vrstvám rozhraní, přes které má tato vrstva přístup k veškeré funkčnosti. V předchozí větě záměrně není

[17] REST je cesta, jak jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání.
[18] API endpoint je unikátní URL adresa, která jednoznačně identifikuje resource.
[19] Klíčovou abstrakcí informace v RESTU je resource. Každá informace, která má jméno může být resource [19].
[20] Query string je částí URL. Používá se pro přiřazení hodnoty specifickému parametru.

specifikována vyšší vrstva, ačkoliv je vyšší vrstvou REST API. Nicméně, toto rozhraní může využívat jakákoli jiná forma prezentační vrstvy. Není tedy problém vzít tuto vrstvu a postavit nad ni například MVC nebo WPF aplikaci.

```
4 references | Tomáš Svatek, 4 days ago | 1 author, 5 changes
public interface ITreeService
{
    3 references | Tomáš Svatek, 20 days ago | 1 author, 3 changes | 0 exceptions
    Task<ServiceResult<IList<ITree>>> GetTreesAsync(IRegion region);

    2 references | Tomáš Svatek, 20 days ago | 1 author, 2 changes | 0 exceptions
    Task<ServiceResult<ITree>> GetTreeById(int id, ProviderName providerName);

    3 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    Task<ServiceResult<IList<ITree>>> GetClosestTreesAsync(IRegion region, double latitude,
                                                           double longitude, int? count);
    2 references | Tomáš Svatek, 4 days ago | 1 author, 1 change | 0 exceptions
    Task<ServiceResult<ITree>> GetClosestTree(IRegion region, double latitude,
                                              double longitude, string commonName);

    2 references | Tomáš Svatek, 4 days ago | 1 author, 4 changes | 0 exceptions
    ServiceResult<ITree> CreateTree(Tree tree);

    2 references | Tomáš Svatek, 21 days ago | 1 author, 2 changes | 0 exceptions
    ServiceResult UpdateTree(int id, Tree tree);
}
```

Obrázek 6- rozhraní ITreeService

Každá metoda rozhraní má návratový typ *ServiceResult*. *ServiceResult* je třída, jejíž instance přepravují do vyšší vrstvy výsledek volání každé metody definované v rozhraní. Může vrátit data nebo popis chyby, která nastala v nižších vrstvách.

AppCore vrstva nijak nezodpovídá za ukládání a načítání dat. Nezávisí na databázi. Komunikace s databází a s dalšími datovými zdroji probíhá pomocí *repository*[21] rozhraní. Implementace těchto rozhraní se nachází na nižší vrstvě. AppCore obsahuje dva typy *repository* rozhraní.

První typ rozhraní slouží pro komunikaci s databází. Rozhraní může být implementováno libovolně. Jeho implementace může pracovat s relační databází, NoSQL databází nebo in-memory databází. Důležité je, že implementace plní kontrakt, který mu rozhraní předepisuje.

[21] Repository je design pattern, který zapouzdřuje logiku pro přístup k datům.

Druhý typ rozhraní slouží pro komunikaci s datovými zdroji, které poskytují data pomocí REST rozhraní.

Arboretum.Infrastructure

V Infrastructure se nachází implementace databázových repositářů. Jsou zde dva repositáře *TreeRepository* a *DendologyRepository*. Repositář se dotazuje na data pomocí *DbContextu*. *DbContext* je třída, která reprezentuje spojení s databází a poskytuje API komunikaci s databází [16].

Na obrázku číslo 7 je metoda *GetDendrologies,* která vrací kolekci objektů *Dendrology*. Parametr *IReduction* obsahuje informace potřebné pro stránkování. Pokud některá property *IReduction* má výchozí hodnotu *null*, tak metoda vrátí všechny instance entity *Dendrology,* které jsou uloženy v databázi.

```
2 references | Tomáš Svatek, 4 days ago | 1 author, 1 change | 0 exceptions
public IList<IDendrology> GetDendrologies(IReduction reduction)
{
    var query = DbContext.Dendrologies;
    if (reduction.PageNumber != null && reduction.PageSize != null)
    {
        var skipCount = (reduction.PageNumber.Value - 1) * reduction.PageSize.Value;
        var reductionDendrologies = query.Skip(skipCount)
            .Take(reduction.PageSize.Value);

        var reductionDomainDendrologies = MapDbDendrologiesToDomain(reductionDendrologies)
            .ToList();

        return reductionDomainDendrologies;
    }

    return GetDendrologies();
}
```

Obrázek 7 - implementace metody GetDendrologies

Arboretum.Webservice

Hlavní zodpovědností Webservice je interakce s datovými zdroji, které poskytují data pomocí REST rozhraní. Tato vrstva implementuje rozhraní *IRestRepository*, který se nachází na vrstvě AppCore.

Datový zdroj je reprezentovaný rozhraním *ITreeDataProvider*. Rozhraní obsahuje čtyři *properties*. Název datového zdroje, adresu serveru, kolekci HTTP

hlaviček a stav, který udává, zda je možné jeho data editovat. Některé tyto *properties* slouží jako konfigurace HTTP klienta.

```
4 references | Tomáš Svatek, 20 days ago | 1 author, 3 changes
public interface ITreeDataProvider
{
    4 references | Tomáš Svatek, 41 days ago | 1 author, 1 change | 0 exceptions
    ProviderName Name { get; }
    4 references | Tomáš Svatek, 41 days ago | 1 author, 1 change | 0 exceptions
    string BaseAddress { get; }
    2 references | Tomáš Svatek, 41 days ago | 1 author, 1 change | 0 exceptions
    bool IsEditable { get; }
    4 references | Tomáš Svatek, 41 days ago | 1 author, 1 change | 0 exceptions
    IList<RequestHeaders> RequestHeaders { get; set; }
    2 references | Tomáš Svatek, 20 days ago | 1 author, 3 changes | 0 exceptions
    Task<IList<ITree>> GetTreesAsync(IRegion region);
    2 references | Tomáš Svatek, 20 days ago | 1 author, 2 changes | 0 exceptions
    Task<ITree> GetTreeByIdAsync(int id);
}
```

Obrázek 8 - rozhraní ITreeDataProvider

Rozhraní dále specifikuje dvě metody *GetTreesAsync* a *GetTreeByIdAsync*. Každý datový zdroj je tedy zodpovědný za načtení svých dat. Toto rozhodnutí vychází z toho, že každý datový zdroj vrací jiné objekty. Konkrétním příkladem může být zdroj, který vrací objekt stromu obsahující tyto properties*: commonName, scietificName* a na druhé straně zdroj, který vrací také objekt stromu s těmito *properties*: *czechName, latinName.* Deserializace takových JSON objektů by musela být implementována pomocí příkazu *switch* v jedné metodě. Tomuto se chtěl autor vyhnout.

```
2 references | Tomáš Svatek, 20 days ago | 1 author, 2 changes | 0 exceptions
public async Task<ITree> GetTreeByIdAsync(int id)
{
    try
    {
        var json = await _httpClient.FetchDataAsync(BaseAddress, $"tree/{id}", RequestHeaders);
        var deserializeTree = JsonHelper<TreeDto>.Deserialize(json);
        var tree = MapToDomain(new List<TreeDto> {deserializeTree}).First();

        return tree;
    }
    catch (Exception)
    {
        return null;
    }
}
```

Obrázek 9 - implementace GetTreeByIdAsync

Všechny datové zdroje se musí zaregistrovat v *RestRepository*, který je implementací *IRestRepository*. Na obrázku číslo 9 je implementace metody *GetTreeByIdAsync*. Metoda má dva parametry: identifikátor a identifikátor datového zdroje. Pomocí těchto dvou parametrů dokáže vybrat správný datový zdroj a zavolat správnou metodu.

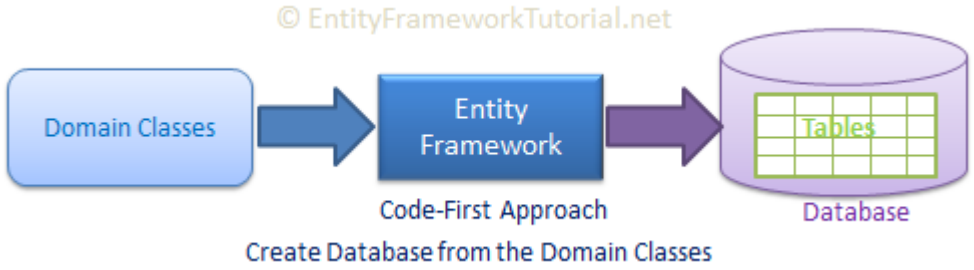
```
2 references | Tomáš Svatek, 20 days ago | 1 author, 2 changes | 0 exceptions
public async Task<ITree> GetTreeByIdAsync(int id, ProviderName providerName)
{
    var provider = GetProviderByName(providerName);
    if (provider == null)
    {
        throw new ArgumentException($"Provider with id={id} does not exist.");
    }

    var tree = await provider.GetTreeByIdAsync(id);
    return tree;
}
```

Obrázek 10 – implementace metody z rozhraní IResRepository

Přidání nového datového zdroje je jednoduché a nijak nenarušuje ostatní celky Back Endu. Vytvoří se třída reprezentující nový datový zdroj, která implementuje rozhraní *ITreeDataProvider*. Nakonec se třída zaregistruje v metodě *RegisterProviders*, která se nachází v *RestRepository*.

Arboretum.Persistence

Persistence vrstva definuje datový model. Je zde použit Entity Framework Core (EF) s přístupem Code-First. V Code-First přístupu EF API vytvoří databázi a tabulky pomocí migrací založených na konvencích a konfiguraci poskytnuté v doménových třídách [16].



Obrázek 11 - EF code-first přístup [16]

Konfigurace je umístěna v samostatných třídách, které implementují rozhraní *IEntityTypeConfiguration*. Konfigurace jednotlivých entit je poté zaregistrována ve třídě odvozené z *DbContextu* v metodě *OnModelCreating*.

```
0 references | Tomáš Svatek, 20 days ago | 1 author, 2 changes | 0 exceptions
protected override void OnModelCreating( ModelBuilder modelBuilder )
{
    modelBuilder.ApplyConfiguration( new TreeConfiguration( ) );
    modelBuilder.ApplyConfiguration( new DendrologyConfiguration( ) );

    base.OnModelCreating( modelBuilder );

    modelBuilder.Seed();
}
```

Obrázek 12 - přetížení metody OnModelCreating

4.2.2 Geolokace

Hlavní obrazovka Aplikace obsahuje mapu, která vykresluje stromy v oblasti uživatele. Oblast uživatele je ta část mapy (výřez mapy), kterou uživatel aktuálně vidí na svém smartphonu. REST API vystavuje endpoint pro tuto obrazovku, který má parametr *VisibleRegion. VisibleRegion* je objekt, který obsahuje informace o oblasti, ve které se uživatel právě nachází. Ukázkové volání endpointu lze vidět níže.

```
api/trees?latitudeMin=49.52385023088912&latitudeMax=49.54205023088911&lon
gitudeMin=14.79986833093368&longitudeMax=14.81806833093368
```

REST API by mohlo vystavovat endpoint, který by vracel všechny stromy bez ohledu na uživatelovu polohu. Aplikace by obdržela všechny stromy a vykreslila je do mapy. Nicméně, toto řešení by značně degradovalo výkon Aplikace. Předpokládejme, že v databázi je uloženo sto tisíc stromů a uživatel se nachází v oblasti, kde je stromů pouze pět. Výsledkem volání tohoto endpointu by byla HTTP response, která by obsahovala všech sto tisíc stromů, ačkoliv uživateli se jich v mapě zobrazí pouze pět.

Vrstva DAL, konkrétně repositář *TreeRepository*, implementuje načítání pouze těch stromů, které se nachází v oblasti uživatele. Na obrázku číslo 13 je LINQ dotaz, který vrací pouze stromy v oblasti uživatele.

```
2 references | Tomáš Svatek, 20 days ago | 1 author, 3 changes | 0 exceptions
public IList<ITree> GetTrees(IRegion region)
{
    var query = DbContext.Trees
        .Where(t =>
            (t.Latitude > region.LatitudeMin && region.LatitudeMax > t.Latitude) &&
            (region.LongitudeMin < t.Longitude && region.LongitudeMax > t.Longitude));

    var domainTrees = MapDbTreeToDomain(query).ToList();

    return domainTrees;
}
```

Obrázek 13 - LINQ dotaz

Jeden z hlavních funkčních požadavků je: „Systém bude vyhledávat strom, který je nejblíže k uživateli“. Tento funkční požadavek definují první dva uživatelské scénáře.

Vzdálenost mezi dvěma GPS body je spočítána pomocí Haversine formule (vzorec). Vzorec je velmi populární a často používaný při vývoji aplikací, které pracují s GPS. Vzorec se používá pro výpočet vzdálenosti mezi dvěma GPS body na zeměkouli [17].

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi 1 \cdot \cos\varphi 2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{(1-a)}\right)$$

$$d = R \cdot c$$

Φ – latitude, λ – longitude, R – radius of the earth (R ≈ 6.371 km)

Obrázek 14 - Výpočet Haversine formule [17]

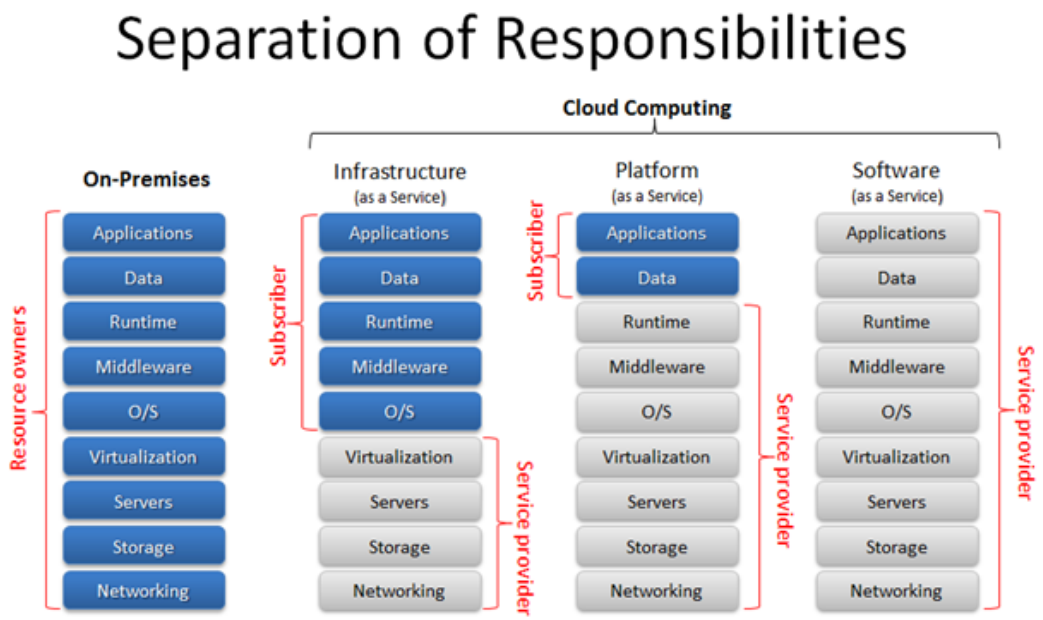
Vzorec je možné implementovat na úrovni SQL dotazu. Nicméně, Back End načítá data nejen z SQL databáze, ale i přes webové rozhraní. Z tohoto důvodu je vzorec implementován na úrovni programu.

```
2 references | Tomáš Svatek, 21 days ago | 1 author, 1 change | 0 exceptions
public static double Calculate(double originLat, double originLon,
                               double targetLat, double targetLon)
{
    var r = 6372.8;

    var latOrigin = originLat.ToRadians();
    var latDestination = targetLat.ToRadians();

    var dLat = (targetLat - originLat).ToRadians();
    var dLon = (targetLon - originLon).ToRadians();

    var a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
            Math.Sin(dLon / 2) * Math.Sin(dLon / 2) *
            Math.Cos(latOrigin) * Math.Cos(latDestination);

    return r * 2 * Math.Asin(Math.Sqrt(a)) / 1000;
}
```

Obrázek 15 - implementace Haversine formule v C#

4.2.3 Nasazení do produkčního provozu

Microsoft nabízí různé distribuční modely (IaaS, PaaS, SaaS), které umožňují hostování různých druhů aplikací v cloudu.



Obrázek 16 - distribuční modely [18]

Back End je provozován v cloudové platformě Microsoft Azure. V rámci Back Endu spadají veškeré použité služby do kategorie služeb Platfrom as a Service (PaaS). Tento distribuční model umožňuje efektivní vytváření aplikací, bez starosti o použitou infrastrukturu a konfiguraci operačního systému. Za konfiguraci infrastruktury a konfiguraci operačního systému zodpovídá poskytovatel cloudových služeb (Microsoft). To umožňuje vývojářům soustředit se pouze na samotný vývoj.

Pro práci s cloudovou platformou Microsoft Azure je nejprve nutné vlastnit uživatelský účet Microsoft. Pro využívání služeb této platformy je dále nezbytné disponovat předplatným. Při tvorbě této práce bylo využito předplatné Azure for Students, které umožňuje využívat vybrané služby s měsíčním kreditem 100 dolarů.

Služby platformy Microsoft Azure se sdružují do logického kontejneru, takzvaného Resource Group. Tato aplikace je sdružena do Resource Group s názvem Arboretum. Arboretum obsahuje App Service plán, na kterém je hostován Back End.

Zvolená úroveň App Service plánu je B1 Basic. Tato úroveň je naprosto dostačující pro potřeby Back Endu. App Service plán využívá operační systém Linux.

Součástí App Service plánu je jedna App Service. App Services nabízí několik typů aplikací. Jedním z těchto typů je Web App, která je použita v rámci této práce jako prostor pro běh Back Endu. Pomocí Web App je možné Back End spravovat a monitorovat.

Další části v Resource Group arboretum je SQL server a SQL databáze. Jedná se o služby, které souvisejí s persistentní vrstvou Back Endu.

Back End je dostupný na adrese: https://arboretum.azurewebsites.net/api. Na této adrese se nachází nejvyšší vrstva Back Endu REST API. Dokumentace REST API je dostupná na adrese: https://app.swaggerhub.com/apis/arboretum/arboretum-client_api/v1.

4.3 Klientská část

Tato kapitola popisuje implementaci samotné Aplikace. Popisuje strukturu projektu a jeho celky. Popis GUI a ovládání Aplikace lze nalézt v Uživatelské dokumentaci, která se nachází v příloze.

4.3.1 Struktura projektu

React Native je velmi flexibilní framework, který nijak nespecifikuje, jak organizovat strukturu projektu. Defaultně obsahuje adresář src, který je určen pro kód vyvíjené aplikace. Nicméně, pokud by se tvůrce Aplikace nijak nezabýval strukturou své Aplikace a všechny soubory/kód by umístil do adresáře src, velmi snadno by se začal ve svém kódu ztrácet. Aplikace byla strukturována následovně.

Actions

V tomto adresáři jsou umístěny tzv. Action Creators. Action Creator je jednoduchá funkce, která vysílá akce a informuje Redux Store, že v aplikaci nastala událost, která mění stav aplikace. Akce pouze popisují, co se v aplikaci stalo, ale nepopisují, jak změnit stav aplikace [13]. Action Creator může obsahovat volání API, zjišťování polohy a jiné operace, které získávají data. Tyto data jsou spolu s informací, o jakou akci se jedná, odeslána do Redux Storu.

```
export const fetchDendrologies = () => async dispatch => {
    try {
        const url = GET_DENDROLOGIES_ENDPOINT;
        const { data } = await axios.get(url);
        dispatch({type: FETCH_DENDROLOGIES, payload: data});
    }
    catch(error) {
        console.error(error);
    }
}
```

Obrázek 17 – načtení druhů stromů pomocí Action Creator

Reducers

Reducer je funkce, která určuje, jak se mění stav aplikace v závislosti na akcích poslaných do Redux Storu. Na obrázku číslo 18 je Reducer, který mění stav aplikace na základě akce, kterou vytváří Action Creator z obrázku číslo 17.

```
export default function(state = [], action) {
    switch(action.type) {
        case FETCH_DENDROLOGIES:
            return action.payload;
        default:
            return state;
    }
}
```

Obrázek 18 - dendrology Reducer

Components

Znovupoužitelné komponenty, které lze použít na více místech Aplikace. Nemají žádný stav. Často se nazývají stateless[22]. Data jsou do nich vložena z rodičovských komponentů pomocí props. Tyto komponenty plní pouze prezentační funkci a nemají žádnou znalost o logice Aplikace ani o jejích datech.

[22] Stateless komponenta – komponenta, který nemá žádný stav.

Popisovaná Aplikace využívá sadu UI komponent React Native Elements[23]. Tyto komponenty velmi urychlují vývoj UI Aplikace. Nicméně, mají velmi základní vzhled, je tedy nutné použít vlastní styly.

Screens

Ve Screens se nacházejí jednotlivé obrazovky Aplikace, které lze vidět na obrázku číslo 2. Skládají se ze stateless komponent. Jsou to komponenty, které mají stav a jsou propojeny s Redux Storem. Stavy, které jsou specifické pro konkrétní obrazovku, si obrazovka drží a řídí sama.

```
return (
  <View style={{flex:1}}>
    <NavigationEvents
      onWillFocus={payload => this.componentWillFocus(payload)}
    />
    <View style={styles.containerStyle}>
      <View style={statusBar.style.statusBar}></View>
      <Notification
        isVisible={this.props.notification.message}
        notificationStyle={{backgroundColor: this.props.notification.color}}
        notificationMessage={this.props.notification.message}
        textColor="#fff"
        onNotificationPress={this._handleNotificationPress}
        />
      <Map
        onRef={(ref) => this.mapRef = ref}
        mapStyle={styles.mapStyle}
        overlayMapStyle={styles.overlayMapStyle}
        region={this.props.region}
        showsUserLocation
        followUserLocation
        onCalloutPress={(event) => this._handleCalloutPress(event)}
        renderMarkers={ this._renderMarkers(this.props.trees)}
        renderPolyline={this._renderPolyline()}
      >
        {this._renderOverlay}
      </Map>
```

Obrázek 19 - hlavní obrazovka Aplikace

Utils

Utils obsahuje sadu užitečných funkcí, které se používají v celé Aplikaci. Každá funkce je ve vlastním souboru a je napsána v čistém JavaScriptu. Funkce lze

[23] https://react-native-training.github.io/react-native-elements/

znovupoužít na jiném projektu, který je postaven na JavaScriptu. Příkladem užitečné funkce je *replaceStringPlaceholders,* kterou lze vidět na obrázku číslo 20.

```
/**
 * Replace placeholders in the string with given values. If a regex is not provided
 * it will use a default regex: /{[0-9]}|{[a-zA-Z]*}/. The default regex match placeholders
 * like this: api/{0}/{1} or api/{ownerId}/car/{id}.
 *
 * @param string String with placeholders.
 * @param regex Optional regular expression for replacing placeholders.
 * @param values Values to replace placeholders.
 */
export const replaceStringPlaceholders = (string, regex, ...values) => {
    let reg = !regex ? /{[0-9]}|{[a-zA-Z]*}/ : regex;

    values.forEach((item, index) => {
      string = string.replace(reg, item);
    });

    return string;
  }
```

Obrázek 20 - funkce pro konstrukci API endpointu

4.4 Testování

Testování Systému probíhalo po celou dobu vývoje. Každé assembly bylo testováno zvlášť a manuálně vývojářem. Vývojář si vytvořil konzolovou aplikaci a referencoval jednotlivé assemblies. Testovány byly zejména metody, které pracují s polohou uživatele. V druhé fázi byly testovány REST API endpointy. Jednotlivé endpointy byly provolávány pomocí nástroje Swagger[24].

Aplikace byla testována na fyzickém zařízení Huawei P8 Lite s operačním systémem Android 6.0 Marshmallow. Aplikace byla dále testována na virtuálních zařízeních s operačním systémem Android ve verzích 7.0 až 9.0.

Na závěr bylo provedeno uživatelské testování. Uživatel testoval zejména uživatelský scénář Kvíz. Test probíhal způsobem, že byly nejprve připraveny testovací data, které obsahovaly pět stromů v oblasti jednoho kilometru. Jednotlivé stromy byly od sebe umístěny zhruba 300 metrů. Testovala se především reakce aplikace na změnu polohy uživatele.

[24] Swagger je open source nástroj, který pomáhá vývojářům při budování API.

5 Plánovaný rozvoj

Rozvoj Systému bude velmi záviset na uživatelské základně Aplikace. Již v době vzniku této práce, je jasné, že Aplikaci si budou stahovat zejména lidé zajímající se o dřeviny. Lze tedy očekávat několik desítek uživatelů. Pokud by uživatelů bylo méně, budoucí rozvoj nedává smysl.

Aplikace byla vyvíjená tak, aby její budoucí portace na platformu iOS byla co nejjednodušší. Nabízí se tedy, co nejdříve po nasazení na platformu Android, se pokusit vyvinout a nasadit Aplikaci i pro platformu iOS. Tento vývoj by měl být velmi rychlý.

Hlavní vylepšení aplikace, které nastane ještě před nasazením Aplikace na platformu Android, bude zavedení uživatelských účtů a funkčnosti s nimi spojenou. Uživatel si bude moci vytvořit uživatelský účet, který mu dovolí přidávat a editovat stromy. Zmiňovaná funkčnost přidávání a editace stromů je již na Back Endu implementována. Nicméně, dokud nebudou existovat uživatelské účty nelze tuto funkcionalitu v Aplikaci využívat. Nebylo by možné zjistit, kdo, jaký strom přidal či editoval.

Další zajímavou funkcionalitou by mohlo být ohlášení nebezpečného stromu. Nebezpečným stromem, může například být strom, který má nalomenou větev. Uživatel by mohl takový strom označit za nebezpečný a tato zpráva by byla automatický poslána příslušné organizaci, která je za tyto věci zodpovědná.

6 Závěr

Cílem práce bylo vytvořit mobilní aplikaci, která poskytne uživateli informace o dřevinách v jeho blízkosti. Pro stanovení základních parametrů projektu byl použit logický rámec, který poskytl základní přehled o realizovaném projektu. V dalším kroku byly vytvořeny uživatelské scénáře, ze kterých byly následně odvozeny funkční požadavky na aplikaci. Po stanovení funkčních požadavků byly vytvořeny wireframy obrazovek, které upřesnily funkci jednotlivých obrazovek aplikace. Následně byla navržena architektura aplikace, na jejímž základě byly vybrány použité technologie. Mobilní aplikace byla vyvinuta za pomoci multiplatformního frameworku React Native, díky kterému je možné snadno vytvořit nativní aplikaci pro dvě hlavní mobilní platformy, Android a iOS. Při vývoji back endu byl použit multiplatformní framework ASP.NET Core. Back End je implementován dle architektury REST. Právě díky této architektuře může služby Back Endu využívat jakákoli aplikace, která je schopna komunikovat pomocí HTTP protokolu. Následně byla aplikace otestována a back end část aplikace nasazena do produkčního prostředí. Nasazení mobilní aplikace pro platformu Android do produkčního prostředí je plánováno na začátek roku 2019.

V průběhu projektu se vyskytl zásadní problém, kterým byla špatně zvolená technologie pro vývoj mobilní aplikace. Autor původně zamýšlel aplikaci vyvíjet pomocí frameworku Xamarin Forms, který se později ukázal jako nevhodný pro tento typ aplikace. Celý vývoj mobilní aplikace tak musel začít nanovo. Tento problém se zásadně odrazil na nedodržení časového harmonogramu projektu.

Aplikace je vhodná pro běžné uživatele, kteří mají zájem poznat stromy ve svém okolí. Může sloužit jako sběrač nových stromů a informací o již existujících stromech.

Autor této práce dává celý Systém k dispozici jako open source. Zdrojové kódy Aplikace a Back Endu jsou umístěny v oddělených repositářích na GitHubu.

7 Seznam literatury

[1] *Mobile App Backend Services* [online]. 2018 [cit. 2018-03-19]. Dostupné z: https://cloud.google.com/solutions/mobile/mobile-app-backend-services

[2] GAJOS, Piotr. *What Is Agile Development for Mobile Apps?* [online]. 2015 [cit. 2018-02-27]. Dostupné z: http://sourcebits.com/app-development-design-blog/what-is-agile-development-for-mobile-apps/

[3] JANIŠ, Petr. Jak využít Kanban při vývoji software [online]. 2013 [cit. 2018-02-27]. Dostupné z: http://www.projectman.cz/clanky/posts/35-jak-vyuzit-kanban-pri-vyvoji-software

[4] DUA, Kinjal. *A Guide to Mobile App Development: Web vs. Native vs. Hybrid* [online]. 2018 [cit. 2018-11-06]. Dostupné z: https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid

[5] TODARO, Dave. *What Are the Key Benefits of Native Mobile Apps?* [online]. [cit. 2018-11-06]. Dostupné z: http://www.ascendle.com/blog/what-are-the-key-benefits-of-native-mobile-apps

[6] KLUBNIKIN, Andrew. *Cross-platform vs. Native Mobile App Development: Choosing the Right Dev Tools for Your App Project* [online]. 2017 [cit. 2018-11-06]. Dostupné z: https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81

[7] BRISTOWE, John. *What is a Hybrid Mobile App?* [online]. 2015 [cit. 2018-11-29]. Dostupné z: https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/

[8] WOUTS, François. *Choosing the right technology for your mobile app* [online]. [cit. 2018-11-29]. Dostupné z: https://medium.com/@fwouts/demystifying-the-different-types-of-mobile-apps-818c91bc5e47

[9] CAMPIÃO, Miguel. *WHY SOFTWARE ARCHITECTURE MATTERS* [online]. [cit. 2018-11-21]. Dostupné z: https://www.imaginarycloud.com/blog/why-software-architecture-matters/

[10] *Introduction to ASP.NET Core*. Oficiální dokumentace ASP.NET Core [online]. 2018 [cit. 2018-12-10]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-2.2

[11] *An Introduction To Entity Framework Core*. Learnentityframeworkcore [online]. 2018 [cit. 2018-12-01]. Dostupné z: https://www.learnentityframeworkcore.com/

[12] EISENMAN, Bonnie. *Learning React Native* [online]. [cit. 2018-11-07]. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[13] Oficiální dokumentace *Redux* [online]. [cit. 2018-12-05]. Dostupné z: https://redux.js.org

[14] Oficiální dokumentace *React Native* [online]. [cit. 2018-12-06]. Dostupné z: https://facebook.github.io/react-native/docs/

[15] Oficiální dokumentace *Expo* [online]. [cit. 2018-12-06]. Dostupné z: https://expo.io/

[16] *Entity Framework Core* [online]. [cit. 2018-11-21]. Dostupné z: http://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[17] TARNAWSKI, Tomasz. *Haversine formula – Calculate distance between two Latitude/Longitude* [online]. 2017 [cit. 2018-12-09]. Dostupné z: https://ttarnawski.usermd.net/2017/08/17/haversine-formula/

[18] *Service Models* [online]. [cit. 2018-12-11]. Dostupné z: http://cloudlighthouse.be/cloud/service-models/

[19] THOMAS FIELDING, Roy. Representational State Transfer (REST): Deriving REST [online]. 1 [cit. 2017-11-22]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

8 Seznam obrázků

9 Slovník pojmů

Backlog – seřazený seznam všech funkcí, které musí produkt obsahovat

GUI - grafické uživatelské rozhraní

Datový zdroj – subjekt, od kterého jsou získávány data

Framework – sada knihoven, které řeší typické problémy při vývoji software

Klientská část – prezentační vrstva aplikace

Marker – grafická reprezentace stromu v mapě

Multiplatformní aplikace – software, který je možné spustit alespoň na dvou platformách

UI – uživatelské rozhraní

Platforma – hardware nebo software používaný pro hostování aplikací či služeb

Portace – úpravy software za účelem jeho fungování na jiné platformě

Wireframe – nástroj pomocí, kterého lze vytvořit prvotní náhled nového řešení

Workflow – pracovní, technologický postup

10 Přílohy

Přílohy jsou rozděleny do dvou částí – Testovací dokumentace a obsah CD. CD obsahuje zdrojové kódy Systému a Uživatelskou dokumentaci Aplikace.

10.1 Testovací dokumentace

Při těchto testech se pracuje s aktuální polohou uživatele. Aktuální poloha uživatele je simulována pomocí aplikace Fake GPS, která dokáže nastavit aktuální polohu na jakékoliv souřadnice na zeměkouli. Testování probíhalo s výchozí polohou [49.533143, 14.809086].

Název	Prvotní načtení stromů podle polohy uživatele
Účel	Testuje, zda se načítají stromy podle polohy uživatele.
Podmínky	Uživatel má přístup na internet Uživatel má povolené polohovací služby
Kroky	1. Uživatel zapne aplikaci
Očekávaný výsledek	1. Uživatel se nachází na hlavní obrazovce 2. Do mapy se vykreslí markery

Název	Scénář Navigace
Účel	Otestování funkčnosti uživatelské scénáře Navigace
Podmínky	Uživatel má přístup na internet Uživatel má povolené polohovací služby
Kroky	1. Uživatel vyhledá strom pomocí ve vyhledávacím poli 2. Uživatel přejde na Detail vyhledaného stromu 3. Uživatel stiskne tlačítko „Navigovat“ nacházející se v obrazovce Detail
Očekávaný výsledek	1. Aplikace přiblíží mapu, na strom, který uživatel hledá a otevře informační okno 2. Mapa se přiblíží na polohu uživatele a vykreslí nejkratší cestu k hledanému stromu

Název	Scénář Kvíz
Účel	Otestování funkčnosti uživatelského scénáře Kvíz
Podmínky	Uživatel má přístup na internet Uživatel má povolené polohovací služby V okolí uživatele se nachází dostatek stromů
Kroky	1. Uživatel se přepne do obrazovky Kviz 2. Uživatel se nachází u cílového stromu ve vzdálenosti alespoň 10 metrů 3. Uživatel zadá odpověď 4. Dokud uživatel neprošel všechny stromy, Kvíz pokračuje bodem 2
Očekávaný výsledek	1. Do mapy se vykreslí první strom v Kvízu a nejkratší cesta k němu 2. Aplikace vyzve uživatel k rozpoznání stromu 3. Aplikace vyhodnotí odpověď a aktualizuje skoré. 4. Pokud uživatel neprošel všechny stromy, tak se vykreslí další strom a cesta k němu. Pokud již prošel všechny stromy, vykreslí se modální okno, které obsahuje konečné skoré kvízu.

10.2 Obsah CD

Přiložené CD obsahuje uživatelskou dokumentaci a zdrojový kód celého řešení.

V kořenovém adresáři se nachází .pdf soubor obsahující uživatelskou dokumentaci. V adresáři App se nachází dva adresáře backend a mobile_app. Adresář backend obsahuje zdrojové kódy serverové části aplikace (Back End). Adresář mobile_app obsahuje zdrojové kódy mobilní aplikace (Aplikace).