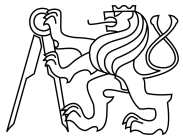
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

| | |
|---|---|
| **Název:** | Systém pro sběr a prezentaci geografických dat |
| **Student:** | Tadeáš Musil |
| **Vedoucí:** | Ing. Jiří Daněček |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2017/18 |

## Pokyny pro vypracování

Seznamte se s problematikou a nástroji multiplatformního vývoje pro mobilní zařízení.
Navrhněte, implementujte a otestujte aplikaci v architektuře klient-server pro sběr a prezentaci geografických údajů se zaměřením na sport.
Jako minimální funkcionalitu umožněte zadávat trasy a hodnotit je podle různých kritérií (obtížnost, frekventovanost atp.).
Pro klientskou část použijte zvolenou multiplatformní technologii a pro serverovou část technologii Java EE.
Serverová část aplikace bude implementovat veškerou business logiku, kterou bude poskytovat klientům prostřednictvím REST API.
Součástí serverové strany aplikace bude i webový klient určený pro administraci.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 17. ledna 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

# Systém pro sběr a prezentaci geografických dat

***Tadeáš Musil***

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Daněček

15. května 2018

# Poděkování

Úvodem bych chtěl především poděkovat všem, kteří mě na vysoké škole kdy podpořili. V první řadě bych rád poděkoval mému vedoucímu práce Ing. Jiřímu Daněčkovi za skvělé vedení práce, emailovou komunikaci a čas strávený na konzultacích. V druhé řadě chci poděkovat oponentovi doc. Ing. Ivanovi Šimečkovi, Ph.D. Dále děkuji rodině, blízkým a přátelům za podporu a rozmluvy nad teoretickými, ale i praktickými problémy.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií


### Odkaz na tuto práci

Musil, Tadeáš. *Systém pro sběr a prezentaci geografických dat.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`https://bakalarska-prace.gitlab.io/client/`⟩.

# Abstrakt

Cílem bakalářské práce je analyzovat problematiku vývoje multiplatformních aplikací. Z této analýzy vybrat konkrétní technologie a ty použít pro vytvoření vzorové aplikace. Aplikace je zaměřena na sportovní trasy. Uživatel v nich může vyhledávat i hodnotit podle různých kritérií a přidávat nové.

**Klíčová slova** multiplatformní aplikace, mobilní zařízení, progresivní webové aplikace, webové komponenty, sport

# Abstract

The goal of this thesis is an analysis of cross-platform application development. We are going to choose specific technologies and apply those in a sample application build. The sample application focuses on sports routes: allowing the user to search (according to various criteria), rate and add new routes.

**Keywords** multiplatform aplications, mobile device, progressive web apps, webcomponents, sport

# Obsah

# Seznam obrázků

# Úvod

V dnešní době moderních technologií si lze jen stěží představit vyhledávání veřejně známých informací bez internetu. Takovýchto dat je opravdu nepřeberné množství. Pokud vyhledáváme konkrétní termín (např. místo, událost, osobnost, ad.), většinou nalezneme co potřebujeme v přijatelném čase. Do nepříjemné situace se dostáváme v okamžiku, kdy potřebujeme nalézt soubor informací podle specifických kritérií. Pro lepší nastínění situace uvedu modelový případ. Rádi bychom vyrazili s rodinou na výlet do lesa, do terénu vhodného pro jízdu s kočárkem. Vzhledem k tomu, že jsme na dovolené v turisticky rušné oblasti, rádi bychom některou méně frekventovanou trasu. Nalezení výletu splňujícího všechna kritéria bude za běžných okolností vyhledávání na internetu opravdu náročné a zabere nám mnoho času.

Jak jsem již zmínil, vše jsme zvyklí hledat na internetu. To však lze v několika formách a reprezentacích. Například v internetovém vyhledávači v prohlížeči, v desktopové aplikaci a v neposlední řadě v aktuálně populárních aplikacích chytrých zařízení, jako jsou telefony, tablety nebo televize. Rád bych nabídl podobnou možnost a v ideálním případě navrhl aplikaci dostupnou ve všech zmíněných zařízeních.

K vývoji takovéto aplikace se dá přistoupit v zásadě dvěma odlišnými metodami. V prvním případě jde o oddělený vývoj aplikace pro každou platformu zvlášť. V druhém případě jde o multiplatformní vývoj, kdy na bázi jednoho zdrojového kódu vytváříme verzi aplikace pro každou platformu. Obě metody s sebou přináší svá specifika, výhody i nevýhody. V analytické části popisuji obě tyto varianty, včetně jejich podvariant, avšak při implementaci vzorové aplikace jsem se vydal cestou druhé metody.

Výstupem bakalářské práce je informační systém v podobě aplikace dostupné na všech platformách, především mobilních, ale i dalších již zmíněných. Spolu s ním detailně popisuji i varianty multiplatformního vývoje.

Kapitola 1

# Cíl práce

Cílem práce je analyzovat a popsat možnosti multiplatformního vývoje mobilních aplikací. Z této analýzy vybrat konkrétní řešení a použít ho pro vytvoření vzorové aplikace. Vzorová aplikace je na téma sport, kde uživatel může vyhledávat trasy dle různých kritérií a přidávat nové. Po technické stránce jde o architekturu klient-server, kde je klientská strana řešena vybranou multiplatformní technologií a serverová strana je postavená na Java EE technologii.

Kapitola 2

# Analýza

Analytická část má za úkol popsat stávající přístupy k programování aplikací pro více platforem. Obecně existuje několik přístupů, avšak zde uvedu pouze ty nejznámější. Na základě této analýzy vyberu jeden konkrétní a podrobněji ho rozvedu v kapitole návrhu.

V návrhu popíšu konkrétní vybrané řešení. Detailně nastíním princip fungování jednotlivých technologií a vzorů z technologického balíku. Konkrétní použití bude však uvedeno až v sekci s realizací.

Když mluvím o platformách, myslím tím různé typy elektronických zařízení, ale i operační systémy (dále jen OS) na nich nainstalované. Platformami mohou být stolní počítače, telefony nebo tablety. Smysl multiplatformního vývoje je tedy v tvorbě aplikací pro více než jednu platformu. Často je tímto termínem označována pouze tvorba pro jeden typ zařízení o více OS, ale toto označení již podle mého názoru ztrácí na významu.

V zásadě zde tedy máme dvě metody. První metoda popisuje oddělený vývoj, kdy pro každou platformu vytváříme novou aplikaci. Druhá metoda je založená na sdílení kódu mezi platformami, toho lze však dosáhnout dvěma principiálně odlišnými způsoby.

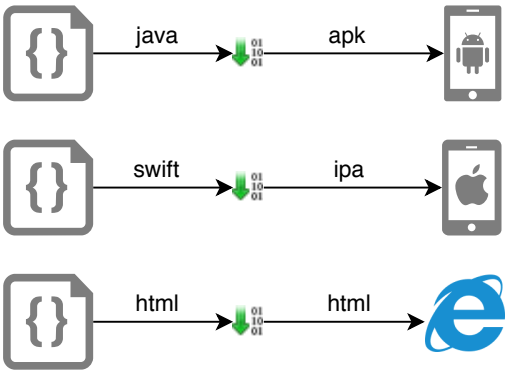
Troufám si tvrdit, že snem každého programátora je napsat kód jednou tak, aby byl funkční na všech platformách. Proti tomu jdou však samotní tvůrci zařízení a OS, kteří z byznysových a dalších důvodů poskytují vlastní proprietární API. Z toho důvodu je zapotřebí části aplikace tvořit pro konkrétní platformu. S tímto faktem se následující metody vypořádávají rozličným způsobem.

Na konci popisu každé metody vypíšu její výhody a nevýhody. Také zde najdete informaci, které všechny platformy lze daným přístupem nebo frameworkem pokrýt.

## 2.1 Oddělený vývoj

Přese všechny teoretické ale i praktické problémy, které tato metoda s sebou přináší, je nejrozšířenější. To je zapříčiněno několika faktory. Vývojáře zaměřené na jednu konkrétní platformu není složité najít, každý programátor preferuje určitý programovací jazyk. Není zapotřebí se učit novým technologiím. V případě mobilního vývoje jsou často firmy zaměřeny pouze na Android a iOS, tím pádem se jim zdá, že udržitelnost vývoje je přijatelná.

Od aplikace na různých platformách jednoho typu zařízení očekáváme, že zjednodušeně řečeno, bude vypadat stejně. Dále by také mohla nabízet stejné funkcionality a mít podobné vlastnosti. Už jen z principu oddělených verzí, udržet toto v synchronicitě, musí být velice náročné. Mluvím především o udržitelnosti vývoje, který musí být ekosystémem podporován. Na rozdíl od následujících dvou metod, je zde podpora ekosystému více než nedostačující a záleží pouze na dané společnosti, jak se k vývoji postaví a kterou metodiku použije.




Obrázek 2.1: Oddělený vývoj

Výhody:

- Jednoduché nalézt programátora zaměřeného na specifickou platformu.
- Maximální kontrola vzhledu a funkcionalit aplikace na konkrétní platformě.
- Aplikace můžou využívat specifiké prvky platformy, jako např. přední kamera u mobilních zařízení, vysoké rozlišení televizních obrazovek.
- Není zapotřebí se učit novým technologiím.

Nevýhody:

- Není možné sdílet žádné vrstvy klientské části aplikace.
- Vícenásobné psaní kódu pro každou platformu extra.

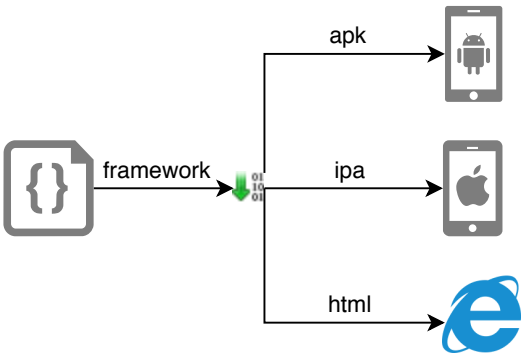
- Složitá udržitelnost funkcionalit bez implicitní podpory nástrojů.
- Zdání, že takovýto přístup není problémový.

## 2.2 Sdílený kód kompilovaný pro každou platformu

Až na dvě výjimky, ve svém vlastním základu následující varianty pokrývají pouze OS Windows Phone, iOS, OSX a Android.

Tato metoda je mnohem přívětivější než předchozí. Umožňuje na základě jednoho kódu aplikace, vytvářet deriváty pro každou platformu zvlášť, a to ještě předtím, než je aplikace do daného zařízení nainstalována. Konkrétněji si představme, že máte k dispozici nějaký proprietární framework pro C/C++, pomocí které dokážete vytvořit aplikaci pro mobilní zařízení nezávisle na OS. Dalším krokem před vypuštěním na trh je samozřejmě vytvoření instalovatelného balíčku (Android - APK, iOS - IPA, exe, . . . ). V tomto kroku se tato první podvarianta druhé metody zásadně liší o druhé podvarianty druhé metody. Například onen zmíněný C/C++ kód je přetransformován do jazyka pro konkrétní platformu a tím pádem ve výsledku bude na každém zařízení spuštěn jiný kód.

V následujících podsekcích jsou ve zkratce popsány některé nejznámější frameworky. První dva jsou typickými zástupci. Třetí se pohybuje na pomezí sekcí Sdílený kód kompilovaný pro každou platformu a Sdílený kód funkční na všech platformách.




Obrázek 2.2: Sdílený kód kompilovaný pro každou platformu

Výhody:

- Sdílené části (často převážná většina) kódu, je výhodou číslo 1.
- Některé frameworky poskytují emulaci iOS zařízení.
- Absence nutnosti vlastnit Mac.

Nevýhody:

- Málo rozšířené mezi programátory.
- Je zapotřebí učit se novým frameworkům.
- Je nutné rozumět, že kód může obsahovat části pro různé platformy.
- Často se jedná o placené frameworky.

### 2.2.1 Xamarin

Xamarin je framework od společnosti Microsoft. Nejedná se pouze o jeden produkt, ale tento projekt zahrnuje i další produkty. Jmenovitě jde o produkty Xamarin Platform, Visual studio app center, Xamarin university, zákaznická podpora a další.

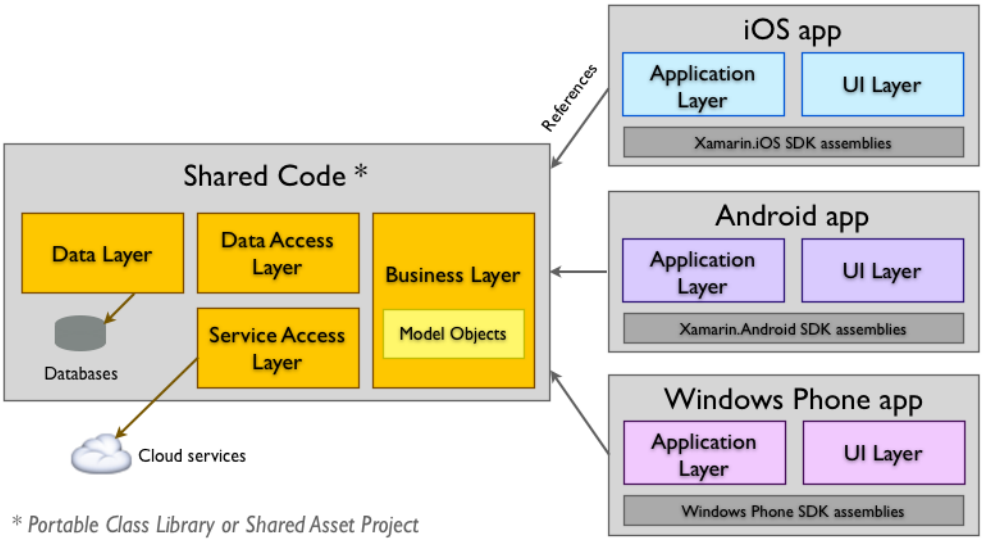
Xamarin platform je prioritní produkt díky kterému je umožněn samotný vývoj pro přední OS mobilních zařízení jako Android, iOS a Windows Phone. Na rozdíl od podobných frameworků, Xamarine má přímou podporu nejen pro Windows Phone, ale obecně pro Windows apps.

Přestože věřím, že je možné vyvíjet v libovolném editoru, všechny Xamarin tutoriály odkazují na Visual studia komunitní edici. Mimo to portál appcenter.ms takzvané Centrum aplikací zaštiťuje cloudové výpočetní centrum pro část životního cyklu aplikace od sestavení, testování až po distribuci a monitoring.

Co mě bohužel zaskočilo, základní dokumentace nepojednává o samotném frameworku, ale spíš o Xamarin.Forms, což je pouze rozšiřující knihovna. Věřím že ona chybějící dokumentace je dostupná v Xamarin university, kam je ovšem zapotřebí registrace.

Po technické stránce Xamarin je framework pro jazyk C# s nativní integrací ve Visual studiu. Sdílet kód definuje byznys vrstvu, datový model, napojení na externí síťové služby a API pro komunikaci s následující částí aplikace. Oproti Sdílenému kódu tedy stojí část aplikace definující UI, která komunikuje se sdíleným kódem pomocí vystaveného API (viz obr. 2.3). Tato část je vytvořena pro každou platformu zvlášť pomocí specifického Xamarin SDK. Pokud aplikace podporuje iOS, Windows Phone a Android, pak bude obsahovat jednu část se Sdíleným kódem a 3 části UI využívající Xamarin.iOS SDK, Windows Phone SDK a Xamarin.Android SDK. Takto v základu funguje Xamarin, avšak hranice sdíleného kódu lze posunout ještě dál, a to až na úroveň některých UI prvků. Xamarin.Forms toto umí zprostředkovat. Jde o sadu UI prvků, které jsou za běhu překládány na nativní prvky platformy.

Xamarin platforma je zdarma dostupná ve Visual studiu, avšak Centrum aplikací je již placené. Samotnou částku jsem bohužel nezjistil, jelikož není uvedena na oficiálních stránkách.




Obrázek 2.3: Sdílený kód vs kód s UI prvky

### 2.2.2 Codename one

Codename one je open source framework od stejnojmené společnosti. Opět se nejedná o jeden produkt, ale poskytuje samotný framework, sestavovací servery jako službu, integraci do editorů, UI builder a portál pro výuku.

Codename one má za sebou robustní ekosystém, podporující překlad do opět všech předních platforem. Pro překlad do iOS není zapotřebí vlastnit Mac a podobně pro Windows a to díky sestavovacím serverům v cloudu jako služba. Bohužel bezplatná varianta produktu neumožňuje vytvářet aplikace větší než 1MB.

Otevřená dokumentace je značně rozsáhlá, ale je zde i k dispozici portál pro výuku s placenými, ale i zdarma video tutoriály.

Po technické stránce Codename one je framework pro jazyk Java nezávislý na vývojářském studiu. Pro akademické účely se hodí populární Netbeans, pro komerční IntelliJ a pro fajnšmekry Eclipse. Codename one umožňuje psát aplikaci také v Kotlinu, což je určitě vítanou vlastností pro programátory píšící nativní aplikace pro Android. Přestože sami autoři doporučují psát všechen kód pouze v Jave, nabízejí také možnost spuštění nativního kódu. Jak jsem již řekl, překlad probíhá na sestavovacích serverech. Pro iOS je aplikace zkompilována do bytecodu, který se následně staticky přeloží do jazyka C, z kterého již Xcode umí vytvořit IPA soubor. Pro Windows je situace podobná, bytecode je pomocí nástroje iKVM přeložen do .netu. Překvapivě nabízí také překlad do Javascriptu pomocí TeaVM.

Codename one má několik placených úrovní a základní neplacenou.

### 2.2.3 Flutter

Flutte je open source framework od společnosti Google. Jak později uvidíte, není to jediný projekt od společnosti Google, který se angažuje v odvětví multiplatformního vývoje. Dlouho jsem přemýšlel, jestli ho do této sekce zařadit, jelikož v zásadě půlka aplikace je kompilována pro každou platformu zvlášť, ale druhá je ponechána. Na rozdíl od předchozích dvou frameworků nemá vlastní sestavovací servery, z čehož vyplývá, že je zapotřebí řešit Apple vývojářský program.

Opět na rozdíl od předchozích dvou frameworků tento neobsahuje placenou dokumentaci. To ovšem není možné brát jako zápor, jelikož otevřená dokumentace je více než dostačující. Obsahuje samotný základní popis frameworku, ukázkové aplikace, ale také dokumentaci k volitelným balíčkům.

Framework bohužel podporuje pouze iOS a Android, je ale otázkou, zda je podpora pro Windows nutná. Velkým bonusem frameworku je jeho jednoduchá integrace s webovým vývojem a také přímá podpora frameworku React Native, s kterým se dá kombinovat.

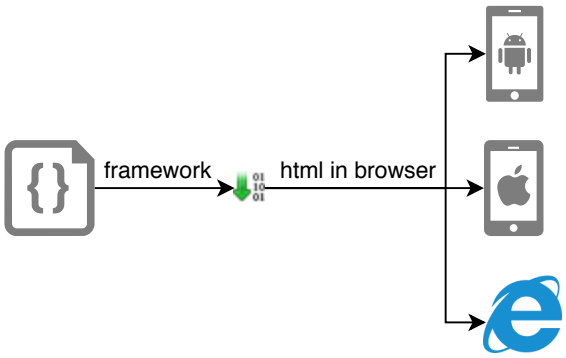
Framework je zaměřen na tvorbu přívětivého UI, které vypadá na každém zařízení stejně. Jak jsem již uvedl, toho lze dosáhnout tím, že podle platformy přepínáme na správnou komponentu. Zde však byl použit jiný přístup. Vývojáři do svého frameworku integrovali open source grafickou knihovnu Skia, napsanou v jazyce C++. Díky této knihovně je možné na každém zařízení vykreslit komponentu úplně stejně. Poté už jen stačilo pomocí této knihovny vytvořit nepřeberné množství UI prvků použitelných ve Flutteru.

Po technické stránce se programátor nemusí bát, že by psal aplikaci v jazyce C/C++. Flutter je navržen pro jazyk Dart, využívajícího jednoduchého API pro použití vytvořených komponent. Proč jsem tedy přemýšlel, jestli Flutter zařadit právě do této kategorie? Skia je zapotřebí zkompilovat pro každou platformu, což nahrává do karet této kategorii. Na druhou stranu jazyk Dart není při sestavování nijak kompilován, ale je ve stejné formě dodán do každého zařízení. Zde přichází na řadu nástroj AOT kompilátor, který je také přibalen spolu s aplikací. Právě kvůli tomuto kompilátoru je možné Flutter zařadit i do následující kategorie.

## 2.3 Sdílený kód funkční na všech platformách

Abychom mohli spouštět stejný kód na různých platformách, musíme něco obětovat. Každý z následujících frameworků se s tím vyrovnává jiným způsobem, ale všechny jsou založené na webových technologiích, jako HTML, CSS a hlavně javascript. Myšlenka, na které jsou postaveny následující frameworky není vůbec nová a její nejznámějším představitelem je programovací jazyk java s jeho JRE/JDK. Jelikož cílové zařízení implicitně nerozumí jazykům, v kterých jsou frameworky napsány, potřebuje nějakou knihovnu, která ano. Jak takovou knihovnu do zařízení dostat? Java tuto zodpovědnost přenáší

na uživatele, ale následující frameworky nikoliv. Aplikace napsané v Ionicu, PhoneGapu a React Nativu mají takovou knihovnu přibalenou přímo v balíčku aplikace. Aplikace postavené na myšlence PWA požadují po cílové platformě, aby měla nainstalovaný internetový prohlížeč.




Obrázek 2.4: Sdílený kód funkční na všech platformách

Rád bych řekl, že všechny následující frameworky jsou absolutně multiplatformní, to ale můžu bohužel říct pouze o posledním. První dva (Ionic a PhoneGap) jsou postavené na projektu Cordova. React Native přibaluje k aplikaci vlastní řešení. V případě PWA nemůžeme zcela mluvit o frameworku, ale spíše o sadě webových standardů, kterým rozumí internetové prohlížeče v nichž jsou ve výsledku aplikace spouštěny.

Cordova je open source framework od společnosti Apache a jde o nejznámější multiplatformní framework. Cordova podporuje širokou škálu platforem: Android, Windows 8.1, Windows Phone 8.1 a 10, iOS, OS X, Fire OS, LG OS a málo známý Unix-like Sprite. Rozhodl jsem se ho nezařadit do následujícího seznamu frameworků, přestože by si to zasloužil a to z těchto důvodu:

- Není určen pro samostatné použití.
- Neposkytuje žádné UI prvky.
- Nespecifikuje žádný MV* návrhový vzor.
- Poskytuje pouze běhové prostředí pro aplikaci
- Zmíněné nevýhody vyvažují frameworky jako Ionic, PhoneGap a celá řada dalších.

Každá platforma dnes ve svém balíčku UI prvků obsahuje takzvané WebView (Android - WebView, iOS - UIWebView, ...) (viz. obr. 2.5). WebView prvek umožňuje přímo v aplikaci zobrazovat obsah internetových stránek, podobně jako to dělají internetové prohlížeče. Oproti internetovým prohlížečům nemají tolik funkcionalit, jako hlavní nevýhodou je absence UI prvků (zpět a dopředu

v historii, apod.). Cordova umožňuje programátorovi psát webovou aplikaci, kterou zabalí a zobrazí právě ve zmíněném WebView.

Výhody:

- Všechen kód je ve všech zařízení stejný.
- Absence nutnosti vlastnit Mac.
- Frameworky stavějí na standardizovaném HTML.
- Webové technologie jako HTML, CSS a Javascript zná každý.
- Současná rozšířenost webových technologií předpovídá dlouhodobou životnost takovýchto aplikací.

Nevýhody:

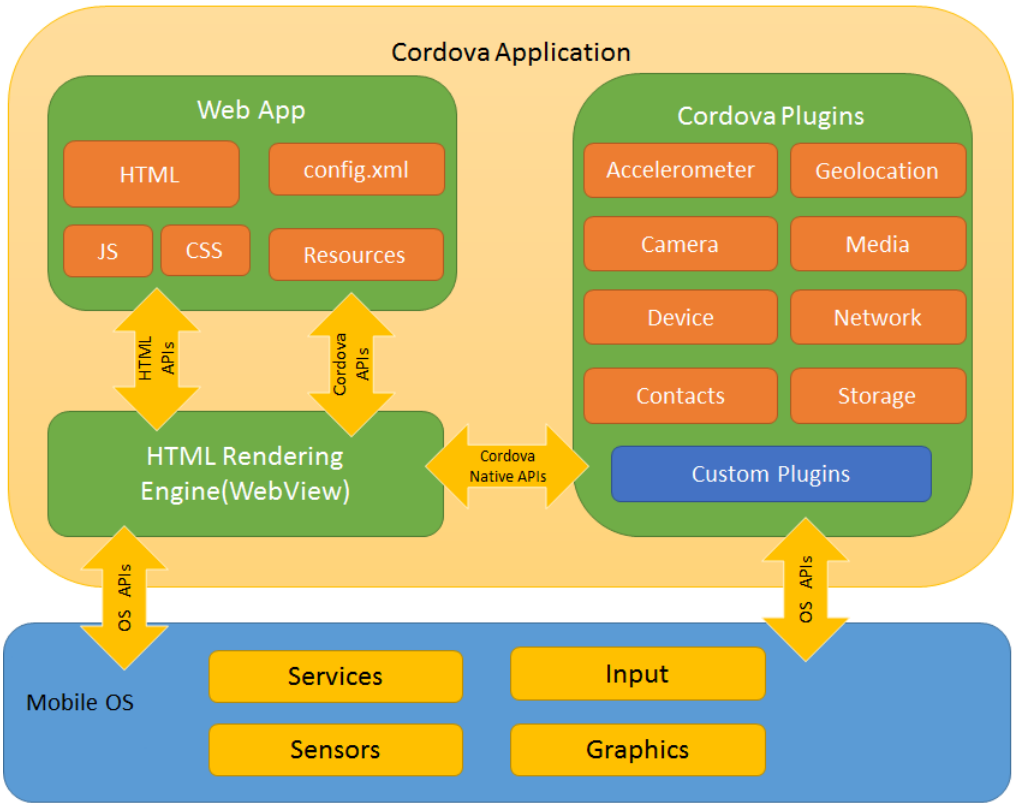
- Málo rozšířené mezi programátory.
- Je zapotřebí učit se novým frameworkům.
- Pouze technologie HTML, CSS a Javascript.
- Spolu s aplikací jsou přinášeny knihovny navíc nebo v případě PWA je zapotřebí mít nainstalovaný prohlížeč.

### 2.3.1 Cordova based - Ionic

Zřídka kdy se setkáváme s úspěšnými projekty od neznámých společností. Výjimkou potvrzující pravidlo je Ionic. Ionic je open source framework od společnosti s původním jménem Drifty, nyní spíše známy jako Ionic. Jde o velice mladý a přesto úspěšný projekt, který podle oznámení společnosti k roku 2015 využilo již 1.3 milionu aplikací. Ionic nabízí celkem 5 produktů: Creator, View App, Deploy, Package, Monitoring a rozsáhlou sadu tutoriálů, mimo jiné také o architektuře a tvorbě PWA, bohužel pro ten je potřeba registrace.

Creator je nástroj pro tvorbu uživatelského rozhraní v uživatelském rozhraní dostupný ve webovém prohlížeči. Uživatelské rozhraní se vytváří přetahováním UI prvků použitím myši. Intuitivním způsobem lze vytvářet animace, měnit motiv aplikace, ale také doprogramovávat rozšířenou funkcionalitu komponentám.

View App je nástroj pro sdílení, testování a získávání zpětné vazby. Tímto nástrojem je možné postihnout všechny členy zainteresované do projektu tvorby aplikace, jako jsou vývojáři, členové kontroly kvality, testeři ale i zákazník. Všechny tyto členy je možné rozřadit do skupin a skupinám přiřazovat práva, například jakou verzi aplikace mohou vidět nebo které její komponenty jsou jim dostupné.




Obrázek 2.5: Cordova architektura

Deploy je nástroj určený pro testování, ale především pro aktualizování již nainstalované aplikace bez nutnosti asistence obchodů s aplikacemi. Nejedná se o obcházení obchodů s aplikacemi, ale jde spíš o rychlejší dodání aktualizací typu security update nebo bug fixing, které jsou přesto následně dostupné i v obchodech s aplikacemi. Package je sestavovací cloud jako služba, která se stará i o distribuci a tak doplňuje produkt Deploy.

Po technické stránce je Ionic založen na Cordově, který rozšiřuje o nevyčerpatelné množství voleb pro tvorbu UI. Většina programování se odehrává v jazyku TypeScript, který je transpilován do Javascriptu ES5. Ionic je postaven na jazyku Angular, který doporučuje používat i pro samotnou tvorbu aplikace. Produkt Package je sestavovací cloudový nástroj jako služba určená pro distribuci do aplikačních obchodů předních platforem. Bohužel pro testování na iOS je stále potřeba Xcode. Přestože Ionic je sám o sobě framework a PWA se dá také označit jako vlastnost aplikace nežli framework, je možné integrovat PWA do Ionicu. Další zajímavou možností je integrace Ionicu a Elektronu. Elektron je framework pro multiplatformní vývoj aplikací pro Windows, Linux a OSX, což znamená, že teoreticky stejná aplikace fungující v mobilním zařízení

by měla mít schopnost být nainstalována i do PC. Ionic lze tedy z jistého pohledu označit jako druhý nejvíce multiplatformní framework, hned po PWA.

Ionic nabízí několik placených plánu, ale také plán zdarma, který bohužel nezahrnuje většinu předních funkcionalit, které jsou až v placených verzích. iOS od iOS9 nevyžaduje Apple developer account, ale pořád je potřeba Xcode.

### 2.3.2 Cordova based - PhoneGap

PhoneGap je framework od společnosti Adobe. V roce 2011 došlo v tomto projektu k zásadnímu zvratu, kdy se vedení rozhodlo jádro projektu oddělit a svěřit do vývoje společnosti Apache. Z tohoto jádra vznikl právě zmiňovaný framework Cordova. Přesto však nebo právě proto je Cordova velice rozvinutým jádrem, na kterém je PhoneGap postaven. Paralelu integrace Cordovy a PhoneGapu lze označit jádro WebKit pro moderní webové prohlížeče. Toto roztrhnutí a vytvořená abstrakce dala vzniku konkurenčních nebo doplňujících projektů jako Ionic, Monaca, Framework7 a spoustě dalším [1].

PhoneGap opět není jen framework, ale nabízí celou řadu dalších produktů. Mezi hlavní takové produkty se řadí PhoneGap Desktop App, PhoneGap Developer App, PhoneGap CLI, PhoneGap Build a nepřímým produktem je podpora rozsáhlé komunity.

PhoneGap Build je sestavovací server poskytující stejné služby jako u ostatních frameworků. Nástroj PhoneGap Desktop App nabízí integraci se PhoneGap Build tak, aby programátor nemusel skrze webové rozhraní nahrávat zdrojové kódy k sestavení balíčků, což navyšuje komfort oproti jiným frameworkům. Desktop App umožňuje spoustu dalších věcí a je doporučován začátečníkům tohoto frameworku. Desktop App je ve své podstatě jen UI pro ovládání PhoneGap CLI, která nabízí další množství příkazů navíc a je doporučená pro pokročilejší použití.

PhoneGap CLI je nadstavba nad Cordova CLI. Cordova CLI lze stále využívat explicitně, nebo PhoneGap CLI nabízí podpříkaz pro použití Cordova CLI.

Po technické stránce framework je založen na Cordově, takže teoreticky lze vytvářet aplikace pro všechny zmíněné platformy, avšak kolekce UI prvků jsou navrženy explicitně pro Android, iOS a Windows Phone. Pro urychlení vývojářského cyklu je zde explicitní podpora ve formě hot reload funkcionality. Tu nabízí produkt Developer App, který je zapotřebí nainstalovat do telefonu prostřednictvím obchodu s aplikacemi dané platformy. Bohužel od určité doby Apple přestal tuto aplikaci podporovat a rozhodl se jí odstranit z iTunes obchodu.

[1] `https://cordova.apache.org`

### 2.3.3 Custom solution - React native

React Native je framework od společnosti Facebook. Přestože framework stále nebyl vydán v plné verzi[2], je hojně rozšířen v tomto odvětví. Na rozdíl od předchozích produktů, React Native nezahrnuje žádné další produkty (krom rozsáhlé otevřené dokumentace), jako je sestavovací server nebo UI builder.

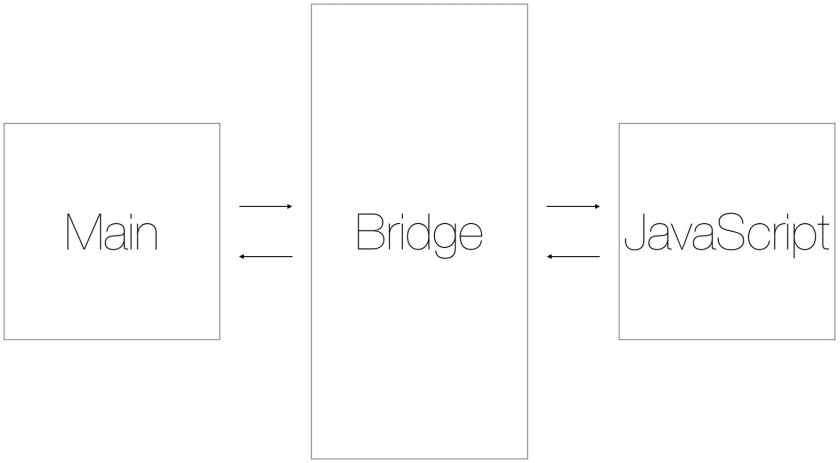
React Native plně využívá frameworku React, který je již několik let jedním z vlajkových frameworků na poli vývoje webových aplikací. Tato možnost využití již rozšířeného webového frameworku v mobilních zařízeních je důvodem obrovské popularity mezi vývojáři. React jako první představil myšlenku virtuálního DOMu, který způsobil značný posun vpřed ve webovém vývoji. Uvedení myšlenky integrace virtuálního DOMu do mobilních zařízení byl onen moment, který dal vzniku frameworku React Native. Přesto React je pouze javascriptová knihovna bez problému použitelná ve frameworku Cordova. Co tedy bylo opravdovým důvodem vzniku Reactu Native? React Native přišel s vlastním řešením, které je podstatně odlišné od Cordovy.

Po technické stránce React Native není založen na myšlence WebView. Zároveň nepoužívá HTML ani CSS jako takové, ale pouze Javascript ve standardu JSX, který slouží jako náhrada za HTML a CSS. Jediné co nám v této skládačce chybí, je zodpovězení otázky, jak je tedy javascript na cílové platformě spuštěn. Android a iOS ve své distribuci obsahují JavaScriptCore engine a Windows Phone obsahuje engine ChakraCore. Toto zjištění pro mě bylo opravdu překvapivé. Javascriptové enginy jsou oproti komplexnějším WebView výkonnější a zatím přinášejí nepřekonatelný uživatelský prožitek z plynulosti aplikace.

Architektura Reactu Native je založená na dvou hlavních vláknech a mostu mezi nimi. První vlákno je standardní vlákno používané nativními aplikacemi. Javascriptový engine běží ve druhém vlákně. Základním myšlenkou je, aby tyto vlákna mezi sebou nikdy přímo nekomunikovala nebo nepoužívala stejná data, což by mohlo zapříčinit jejich vzájemné blokování nebo záseky v aplikaci. Proto byl mezi nimi vystaven most (viz. obr. 2.6) s třemi základními vlastnostmi:

- Asynchronicita - všechna komunikace mezi vlákny je asynchronní, díky čemuž se nikdy nezablokují.
- Dávkování - pokud hned po sobě z jednoho vlákna přijde více zpráv, druhému vláknu jsou přeposlány jako jedna, což optimalizuje výkon.
- Serializovatelnost - jako zprávu lze poslat třeba objekt, který je však na přemostění zkopírován, což zajistí, že obě vlákna nikdy nepracují se stejnou pamětí.

[2]aktuálně se nachází ve verzi 0.55




Obrázek 2.6: React Native architektura

Jak jsem již řekl, projekt neposkytuje další produkty a proto je úplně bezplatný.

### 2.3.4 PWA

Na úvod bych chtěl říct, že právě tuto variantu, jsem si vybral pro vytvoření vzorové aplikace. Myslím že její potenciál předčí všechny předchozí a v budoucnu se stane[3] novým standardem pro vývoj aplikací obecně.

Hned z počátku je nutno říct, že PWA není framework jako všechny předchozích případy. PWA je zkratkou pro progresivní webové aplikace a nejedná se ani o konkrétní technologii nebo API. PWA je přístup k vývoji webových aplikací, který využívá stávajících technologií a nástrojů pro nabídnutí ideální uživatelské zkušenosti. Přestože se jedná o webovou aplikaci, jejím hlavním cílem je nabídnutí uživatelům stejný zážitek, který se jim dostává při používání nativních aplikací.

Od této chvíle budu o PWA mluvit jako o architektuře. Proto abychom tuto architekturu mohli akceptovat jako variantu pro vývoj multiplatformních aplikací, je zapotřebí si připustit fakt, že už nejenom OS nebo samotná zařízení lze chápat jako platformu, ale zároveň i samotné gigantické internetové prohlížeče, můžeme označit platformou. Stále více aplikací se přesouvá do tohoto prostoru, a proto myslím, že tato akceptace je nutná.

Proč tato architektura vznikla? Důvodů je hned několik:

[3]Prognózy předpovídají do roku 2020 [1]

- Přijatelná distribuce aplikací do chytrých zařízení byla pouze v jedné formě a to proprietárních obchodů jako Google Play, iTunes a Microsoft store.
- Nejednotný přístup k distribuci aplikací. Každá platforma má vlastní formát (apk, ipa, ...).
- Nativní aplikace mohou získat přístup k citlivým datům.
- Webové technologie jsou bezkonkurenčně nejrozšířenější.
- Plno dalších ...

Jak jsem již poznamenal, tato architektura je sadou již existujících technologií, které jsou samy na sobě nezávislé, ale jejich společné použití dokáže přinést unikátní výsledek. Všechny tyto webové technologie o kterých budu psát, byly vyvinuty nezávisle na sobě a není problém použít pouze libovolnou podmnožinu z nich. Technologie vypíši v pořadí dle mého vyznačující jejich důležitost a všem se dá přivlastnit postfix API: Service Worker [2], Web app manifest [3], A2HS [4], Cache, Fetch, Promises, Push, Notifications. Myslím že bez těchto 8 API bychom nemohli předpokládat úspěch PWA architektury, ale následný seznam vypsaný se stejnou myšlenkou vyjmenovává čistě volitelné technologie, který více či méně přináší komfort uživatelům a programátorům. Jde o: HTTP2, Background sync, Payment Request, Custom Elements [5], Shadow DOM [6], HTML Imports [7], HTML Template [8]. Kdekoho by mohl zděsit a odradit jejich počet, ale nutno poznamenat, že v drtivé většině případů, se jedná o API o pramálo funkcích. V této kapitole se již nebudu zabývat jejich popisem a použití, to provedu podrobněji v kapitolách Návrh 3 a Realizace 4.

Všechny zmíněné technologie se používají v internetových prohlížečích a je tedy na nich jak je podporují. S myšlenkou této architektury jako první přišla společnost Google a velice aktivně jí podporuje ve svých prohlížečích Chrome a Chrome pro Android. Vzhledem k tomu, jak se tato architektura stává čím dál více oblíbenou, vzrůstá podpora i dalších prohlížečů jako Safari, Opera, Edge a Samsung. Prohlížeč IE od společnosti Microsoft je však ze všech nejkonzervativnější a podpora z jeho strany se moc neočekává. Ze samotného filozofie se také neočekává podpora od prohlížeče Opera mini. Nutno poznamenat, že se stále bavíme o sadě technologií, takže podpora nepřihází naráz, ale pomalou implementací jednotlivých API. Technologie jako HTML Imports, Shadow DOM a Background sync jsou pro řadu prohlížečů kontroverzním spíše než implementačním tématem.

Právě protože jsem přijal internetový prohlížeč jako platformu, z důvodů vzniku této architektury, mé dosavadní znalosti a zkušenosti s některými zmíněnými technologiemi a přes teprve narůstající podporu u prohlížečů, jsem se rozhodl PWA použít pro realizaci vzorové aplikace.

Kapitola 3

# Návrh

V předchozí kapitole jsem rozebral dostupná řešení pro vývoj multiplatformní aplikace, z kterých jsem vybral architekturu PWA jako ideální řešení. Tato kapitola je zaměřená na návrh aplikace, takže zde uvidíte doménový 3.1 model, případy použití 3.2, návrh infrastruktury 3.3, strukturu serverové části 3.4, návrh uživatelského rozhraní 3.5 a relační model 3.6. Na závěr navrhuji a popisuji vhodné vlastnosti aplikace, které by zvolená architektura PWA měla mít.3.7

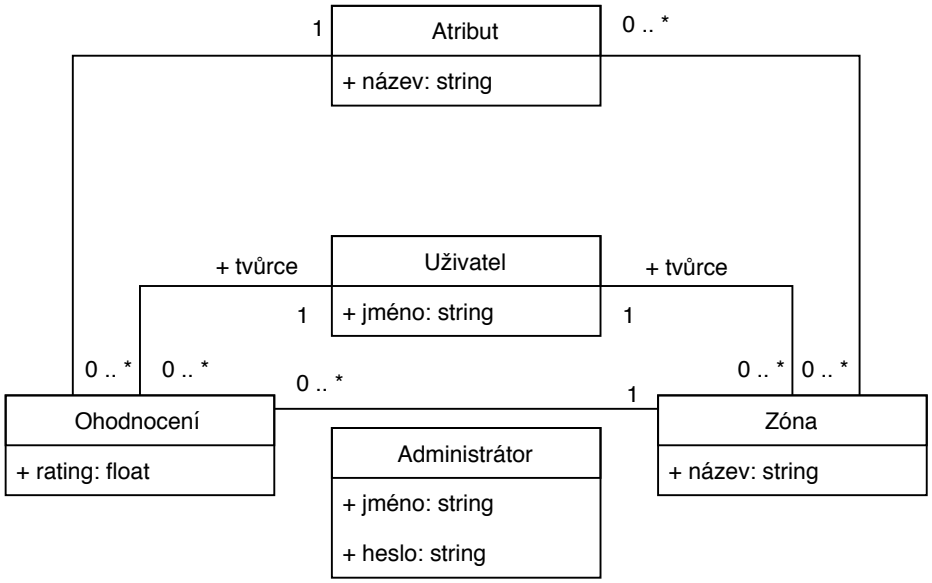
Samotné ukázky použití jednotlivých technologií ukáži až v kapitole o realizaci 4. Obrázky v této kapitole byly vytvořeny v online nástroji draw.io od společnosti JGraph Ltd. Draw obsahuje rozsáhlou kolekci tvarů, díky které lze vytvářet libovolné diagramy. Tato kolekce je rozdělena do kategorií podle využití tvaru.

Tvorba bakalářské práce je především zaměřena na analýzu a zvolené řešení, jeho principů a jejich implementace. Z toho důvodu se návrh aplikace snaží být hodně střídmý a nikterak komplexní, abych měl možnost zaměřit se především na korektní implementaci architektury.

Návrh slouží především jako doporučení pro implementaci, tudíž výsledný produkt se může lišit. Obecně změny při vytváření produktu jsou časté a je zapotřebí je respektovat, zejména pokud jsou z iniciativy zákazníka.

## 3.1 Doménový model

Sekce se zabývá obecným popisem entit, vyskytujících se v jakékoliv části aplikace, a vztahy mezi nimi. Každý box na obrázku vyjadřuje jednu entitu a položky v boxu představují její atributy. Vztahy mezi entitami vyjadřují čáry a čísla vyjadřují kolikrát se mohou v oné relaci vyskytnout.3.1




Obrázek 3.1: Doménový model

### 3.1.1 Uživatel

Aby aplikace byla schopná evidovat autora ostatních entit, bude zapotřebí vytvořit entitu, která tohoto autora bude reprezentovat. Reprezentaci autora tvoří entita Uživatel. Evidování autora je dobrým základem pro úspěšnou správu obsahu aplikace. Uživatel může být autorem všech entit, kromě sebe sama a Administrátora. Mezi atributy je pouze jméno Uživatele.

### 3.1.2 Zóna

Zóna je nejzákladnější entitou aplikace. Zóna reprezentuje oblast na mapě, které jsou přiřazeny žádný až více Atributů, podle kterých může Uživatel ohodnocovat. Zóně jsou také přiřazena již samotná Ohodnocení. Forma oblasti na mapě je záměrně blíže nespecifikovaná, jelikož se může jednat například o trasu, bod na mapě, a tak podobně. V návrhu je Zóna takto abstraktně pojata, aby nebyl problém v budoucnu přidat další konkrétní formy. Mezi atributy je pouze název Zóny.

### 3.1.3 Atribut

Atribut náleží žádné až více Zónám. Pod touto entitou je možné si představit třeba Převýšení, Frekventovanost, Vhodnost pro malé děti a tak dále. Entita má dvě využití, jednak jeho relace se Zónou vyjadřuje, že konkrétní Zóna může být tímto Atributem ohodnocena. Druhé použití je ve vztahu k Ohodnocení. Přiřazení Atributu k Zónám má na starosti právě Administrátor. Atribut

je opět pojat abstraktně, aby při běhu aplikace nebyl problém přidávat nové druhy Atributů. Mezi atributy patří pouze název Atributu.

### 3.1.4 Administrátor

Administrátor má za úkol spravovat obsah aplikace. V tomto případě bohatě postačí, když bude moci přiřazovat Atributy Zónám tak, aby podle nich mohl Uživatel ohodnocovat. Mezi atributy patří přihlašovací jméno a heslo Administrátora.

### 3.1.5 Ohodnocení

Ohodnocení představuje třetí, nejdůležitější entitu aplikace. Umožňuje Uživateli vyjádřit míru spokojenosti s jedním konkrétním Atributem na jedné konkrétní Zóně. Míru spokojenosti vyjádří ve formě číselné hodnoty v intervalu 0 až 10. Z čehož 0 vyjadřuje absolutní nespokojenost a 10 naopak absolutní spokojenost. Průměry těchto Ohodnocení budou zajisté hodnotnými údaji, reprezentujícími spokojenost Uživatele se Zónou. Relace jsou tedy evidentní, Ohodnocení náleží právě jednomu uživateli, jednomu Atributu a jedné Zóně. Mezi atributy patří pouze celočíselná hodnota rating.

## 3.2 Případy použití

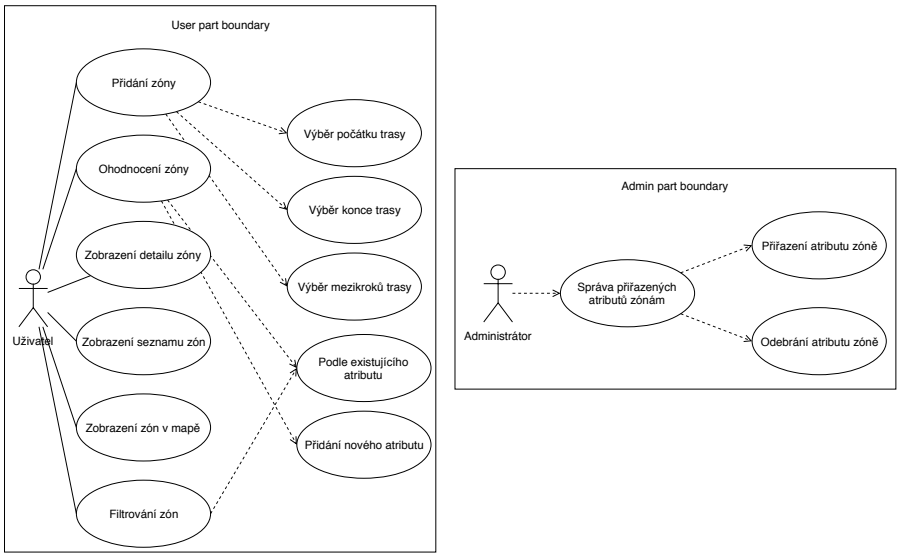
Sekce se zabývá obecným popisem chování aplikace, který se v poměru 1:1 promítá uživateli do nabízených funkcí. Popis vyplývá z cílů práce. Kapitolu lze pojmout také jako stručný popis funkčních požadavků.

Aplikace bude mít ve své podstatě dva typy uživatelů a to administrátory a klasické uživatele. Administrátoři budou mít na starosti přiřazování Atributů k Zónám a Uživatelé vytváření Ohodnocení podle přiřazených Atributů. Uživatelé také budou vytvářet nové Zóny a Atributy.

Úvodní stránka bude zobrazovat základní přehled evidovaných Zón ve dvou formách. První forma bude jednoduchý seznam se základním popisem Zóny. Pro lepší představu si uživatel bude moci přepnout pohled na mapu, kde uvidí vyznačené Zóny. Úvodní stránka také nabídne odkazy do ostatních podstránek. Ostatní podstránky budou: filtr zobrazovaných Zón, detail Zóny a přidání nové Zóny.3.2

- Úvodní stránka musí Uživateli poskytnout základní přehled evidovaných Zón.
- Podstránka filtru musí umožnit Uživateli zvolit konkrétní kritéria, které chce aby zobrazované Zóny na Úvodní stránce splňovaly.
- Podstránka detailu Zóny poskytne uživateli detailnější popis Zóny, ale také hodnotit dle Atributů a přidávat nové Atributy.

- Poslední podstránka umožní uživateli zaevidovávat nové Zóny a to intuitivně.

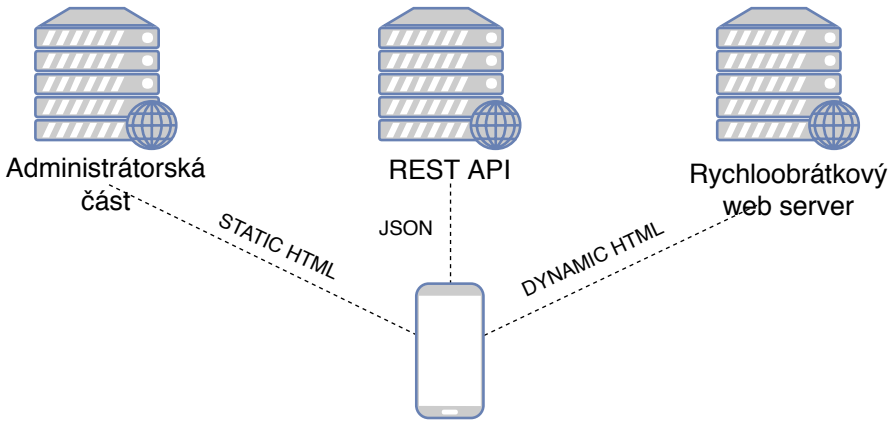



Obrázek 3.2: Model případů použití

## 3.3 Infrastruktura

Považuji za důležité uvést přehled zařízení a jejich roli v aplikaci. Tato sekce se tímto přehledem zaobírá a rovnou bere v potaz vybrané řešení z analýzy. Architektura PWA doporučuje, nikoliv přikazuje, aby všechen kód renderovaný na uživatelských zařízeních byl statický, nikoli dynamický [4]. Klientská část aplikace bude poskytována rychloobrátkovým HTTP serverem. Obsah aplikace, který představují data o Zónách, Uživatelích apod., budou poskytována skrze REST API na dalším serveru HTTP serveru. Tento server bude vystavěn na technologii Java EE, jak požadují cíle. Další část aplikace, určená pro administrátory, bude poskytována opět na dalším serveru, zde se však neočekává velká vytíženost, tudíž tato část může být napsána dynamickým kódem. Poslední server, který je potřeba poznamenat, je server databázový.

[4] V tomto pojetí je rozdíl mezi statickým a dynamickým kódem v tom, že dynamický kód vytváří HTTP odpověď teprve při zpracovávání požadavku, což se mimo jiné promítne na době potřebné pro vytvoření odpovědi. Na druhou stranu statický kód pouze vezme již předem vygenerovanou odpověď a tu vrátí. Tento rozdíl je znám především díky evoluci webových technologií, kdy před příchodem cgi scriptů nebo jazyka php, byl používán právě statický kód. Po příchodu těchto scriptovacích jazyků nadešla éra dynamicky generovaných odpovědí, která trvá přinejmenším do dneška avšak z výkonnostních důvodů webových stránek se přechází zpět.

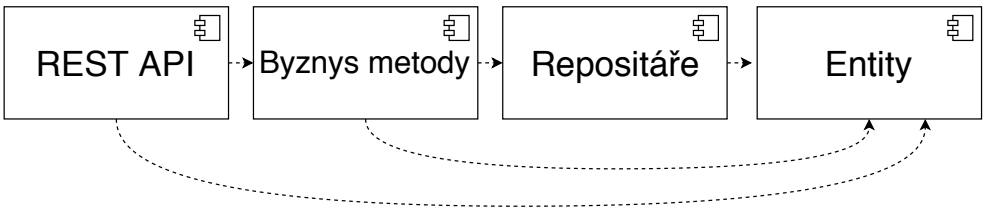



Obrázek 3.3: Model infrastruktury

## 3.4 Struktura serverové části

Rád bych také ve stručnosti popsal návrh struktury serverové části, jejíž výstupem bude REST API. Strukturu serverové části se bude skládat ze čtyř vrstev, které budou hierarchicky uspořádány 3.4. Tento seznam popisuje jejich závislost, kdy každá předcházející je závislá na té následující, ale nikoli opačně. Seznam: REST API, byznys metody, repositáře a entity. Tato hierarchie zajišťuje modulárnost aplikace a jednoduchou rozšiřitelnost, případně vyměnitelnost modulů.

Představa o fungování takového návrhu je následující. REST API přijme HTTP požadavek, který zvaliduje a pokud je vše v pořádku, zavolá příslušnou byznys metodu, jejíž výsledek převede do formátu požadovaného v HTTP požadavku. Byznys metoda získá resp. vloží data z resp. do repositářů a nad nimi provede byznys transformaci. Repositář již rozumí, kde a jakým způsobem jsou data uložena, takže již ví, jakým způsobem se na ně dotázat nebo je upravit. Poslední vrstva s entitami bude jako jediná sdílena napříč všemi vyššími vrstvami tak, aby byla zajištěna jednoduchá práce na objektové úrovni.




Obrázek 3.4: Popis struktury serverové části

## 3.5 Návrh uživatelského rozhraní

Tato sekce je zaměřena na návrh uživatelského rozhraní. Jak jsem v analýze uvedl 2.3.4, PWA se zobrazuje v internetovém prohlížeči, takže je dostupné na jakékoliv platformě s nainstalovaným prohlížečem. Je důležité si uvědomit, že se aplikace může zobrazit na malých, ale zároveň i velkých displejích, od mobilních telefonů přes stolní počítače až po projektory. Zde mám dvě možnosti, jak navrhnout UI a to responzivní nebo pro každé zařízení odlišný. Již ve fázi návrhu se doporučuji rozhodnout pro první možnost, jelikož je méně komplexní. Cílem práce je navrhnout především mobilní aplikaci, tudíž následující návrhy jsou zobrazovány jako mobilní.

Sekce je rozdělená do dalších podsekcí pro každou podstránku jedna podsekce. V každé podsekci jsou objasněny funkce jednotlivých prvků.

Aplikaci bude tvořit základní rozložení dostupné na všech podstránkách, které se skládá z hlavičky a postranního menu. V hlavičce bude dostupné kontextové menu a také tlačítko na otevření druhého postranního menu.

### 3.5.1 Úvod

Úvodní stránka 3.5 nabídne základní přehled o evidovaných zónách a zároveň jejich zobrazení v mapě. Základní výpis bude ve formě seznamu tvořeného z karet [9], kde každá karta bude obsahovat stručné informace o zóně. Druhá forma výpisu bude v podobě vkreslených zón do mapy. Pokud mapa bude vykreslovat například trasu, je zapotřebí, aby byly jasně označené mezikroky. Mezi seznamem a mapou bude možné přepínat toggle tlačítkem, které svítí šedivě pokud zobrazuje seznam s kartami a modře pokud mapu. Tlačítko vedle tlačítka na přepínaní pohledu je tlačítko sloužící pro přechod na podstránku Filtru. Tlačítko zakroucené šipky slouží pro aktualizaci stránky. Tlačítko plus v pravém dolním rohu slouží pro přechod na podstránku Přidání.

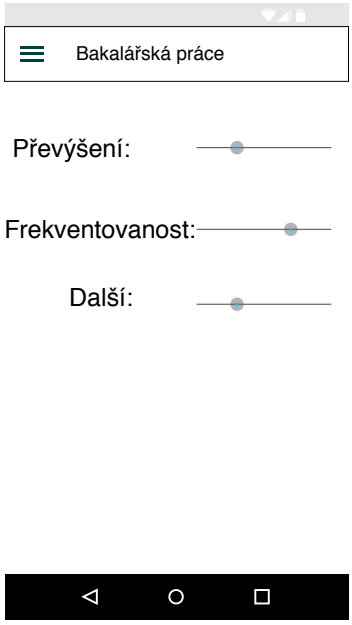
### 3.5.2 Filtr

Z doménového modelu vyplývá, že uživatel může ohodnocovat číselnou hodnotou a jde o zásadní fakt i pro tuto podstránku 3.6. Číselná hodnota lze reprezentovat opět několika způsoby a mezi nejpříhodnější formy se řadí textový vstup omezený na číselné hodnoty v určitém intervalu [10], hvězdičkovací metoda [11] nebo vstup s posuvníkem [12]. Navrhuji použít posuvník ke kterému vykreslit stupnici pro přehlednost.

### 3.5.3 Detail

Podstránka Detail 3.7 bude zobrazovat bližší informace o samotné zóně, mimo to také umožní uživateli hodnotit zónu nebo přidávat nové atributy.


Obrázek 3.5: Wireframe úvodí stránky




Obrázek 3.6: Wireframe podstránky filtru


Obrázek 3.7: Wireframe podstránky detailu zóny

### 3.5.4 Přidání

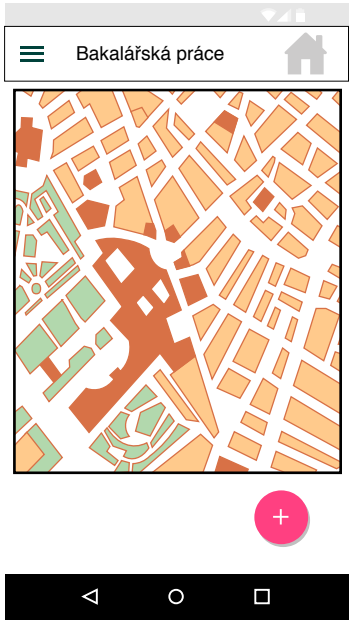
Podstránka Přidání 3.8 slouží právě a jen pro přidání nové zóny. Přidání bude probíhat výběrem zóny v mapě, například pokud bude přidávat trasu, prvním kliknutím vybere start a druhým cíl. V pravém dolním rohu se nachází tlačítko fajfky, které slouží pro potvrzení výběru zóny a její zaevidování. Uživatel po přidání bude přesměrován na Úvod.

## 3.6 Relační model

Na rozdíl od doménového modelu nebo konceptuálního modelu, Relační model je již implementačně nezávislý. Z určitého úhlu pohledu jde o jejich nadmnožinu, jelikož obsahuje všechny předchozí informace, ale i další upřesňující. Navíc zde přibudou datové typy položkám, identifikátory, cizí klíče a další omezení.

V tuto chvíli je potřeba navrhnout konkrétní databázový server, abych byl schopen převést entity na tabulky [5], nebo byl schopen určit datové typy sloupečkům v tabulkách. Databáze H2 se zdá jako ideální volba. Přestože její velikost je necelé 2MB a mohlo by se zdát, že toho moc nenabídne, opak je pravdou [13]. Jedná se o relační databázi podporující SQL standard, které lze přepínat kompatibilitu mezi databázemi MySQL, Derby, PostgreSQL, HSQLDB a dalšími, tudíž podporuje všechny známe datové typy.

[5] Když bych si vybral třeba grafovou databázi, nejspíš bych entity nepřeváděl na tabulky




Obrázek 3.8: Wireframe podstránky přidání nové zóny

V popisu doménového modelu jsem o zóně pojednával 3.1.2 jako o abstraktní entitě, ale zde tuto abstraktnost musím již konkretizovat. Navrhuji aby aplikace podporovala zónu typu trasa a zónu určenou jedním bodem na mapě. Trasa se bude skládat z více bodů na mapě a jejich pořadí bude určovat sloupeček *ORDER_ID*. Ostatní entity z doménového modelu lze převést jedna ku jedné, ovšem z praktických důvodů dojde k přejmenování některých atributů a všech názvů tabulek.

## 3.7 Vlastnosti aplikace

Stále se nacházím v kapitole s návrhem aplikace, proto tato sekce neukazuje konkrétní použití následujících technologií v bakalářské práci, ale navrhuje vlastnosti, jaké by aplikace mohla mít. Kapitolu lze také pojmout jako seznam nefunkčních požadavků. Návrh je založen především na vlastních znalostech a zkušenostech z používání aplikací, ale také na vlastnostech architektury PWA [14] [15].

Na rozdíl od webových stránek očekáváme od nainstalovaných aplikací více vlastností. Mít možnost aplikaci jednoduše nainstalovat je příklad jedné takové základní vlastnosti. Dále bych uvedl bezpečnost aplikace, nezávislost na internetovém připojení, responzivita a rychlá odezva. V poslední době je velice populární sdílení obsahu a již jsme si zvykli, že toto aplikace také umožňují. Možnost samotnou aplikaci nebo její obsah vyhledat pomocí internetového vyhledávače je bez diskusí další důležitou vlastností. Předešlý výčet defin-

uje vlastnosti, které bych rád, aby výsledná aplikace měla a v následujícím seznamu tuto představu upřesním.

[14, Přechodem na progresivní webovou aplikaci byl tým společnosti Lyft schopen snížit technickou a operační cenu spojenou s podporou více platforem. Důležitější však bylo, že narostl počet nových uživatelů na zařízeních iOS a Android, ale také z dalších platforem, které byly pro Lyft do té doby nedosažitelné: Windows Phone a Amazone Fires.]

### 3.7.1 Instalovatelnost

Instalovatelnost je fundamentální vlastnost aplikace. Instalovatelnost označuje především proces instalace aplikace do OS, jehož výsledkem je ikona na ploše. Linux a Mac mají své správce balíčků, Windows explicitní instalační programy (Wizard) a chytré telefony své obchody s aplikacemi. V tomto případě však požaduji instalovatelnost internetové stránky skrze prohlížeč. Této vlastnosti by mělo být možno dosáhnout kombinací technologií Web app manifest [3] a A2HS [4].

### 3.7.2 Bezpečnost

Jedna ze základních podmínek instalovatelnosti aplikace je přenos přes zabezpečený kanál a právě proto bude tato vlastnost nedílnou součástí aplikace. Aby prohlížeč nabídl uživateli nainstalovat aplikaci, požaduje po stránce přenos přes HTTPS. Důvodů je několik, ale hlavní důvod je, aby nikdo nemohl podvrhnout aplikaci, kterou se uživatel může chystat nainstalovat. Důsledek je evidentní, webový server musí poskytovat klientskou část aplikace přes HTTPS.

### 3.7.3 Nezávislost na internetovém připojení

Mobilní zařízení se mohou nacházet ve třech stavech vzhledem k internetovému připojení, online, offline nebo pomalé a nespolehlivé připojení. Nezávislost na internetovém připojení je vlastnost, která se snaží postihnout rozdíly mezi těmito třemi stavy. Aplikace by se každopádně ve všech těchto třech situacích měla alespoň spustit a v případě, že není schopná dále pracovat, upozornit uživatele. Nabízejí se také další vlastnosti jako funkce práce offline, nabízení již dříve načteného obsahu a opožděné odeslání dotazů. Této vlastnosti by mělo jít dosáhnout kombinací technologií Service Worker, Fetch, Cache, Background sync a Promises.

### 3.7.4 Responzivita

Responzivita je vlastnost aplikace postihující její vzhled a označuje takové navržení designu díky kterému je aplikace použitelná na jakémkoliv zařízení. Této vlastnosti by mělo být možno dosáhnout kombinací technologií Custom

elements, Shadow DOM, HTML Imports, HTML Template a použitím již existujících elementů třetích stran [6].

### 3.7.5 Rychlá odezva

Více než polovina uživatelů opustí stránku po delším čekání než 3 sekundy. Jakmile je stránka jednou nahrána, očekáváme že bude rychlá, bez zasekávání, stejně jako její další načtení. [16] Rychlého přenačtení stránky lze dosáhnout znovupoužitím již dříve načtených částí webu, jako například základní rozložení. Pokud chci načíst stránku, která již byla dříve načtená, nemusím na odpověď čekat z internetu, ale mohu použít dříve načtenou verzi. Této vlastnosti by mělo jít dosáhnout kombinací technologií Service Worker, Fetch, Cache a Promises.

### 3.7.6 Vyhledatelnost

Aplikace a její obsah by měl být zaindexovatelný tak, aby ho mohli nabízet internetové vyhledávače, jako to dělají s klasickými stránkami. Zároveň s tím by mělo být jednoduché aplikaci sdílet. Aplikace je ve své podstatě internetová stránka obohacená o spoustu vlastností, tudíž její zaindexování je bezproblémové a pro sdílení se dají použít url odkazy. Někdo by mohl namítnout, že stránka je plná javascriptu, a vyhledávače tyto scripty neumí spouštět. Opak je pravdou, vyhledávače dnes již spouští javascript a pro ty kteří ne, jsou zde metodiky předrendrování nebo rendrování na serveru. Této vlastnosti by mělo jít dosáhnou technologií Web app manifest, SSR, Prerender.

[6] `https://www.webcomponents.org/elements`

Kapitola 4

# Realizace

Kapitola má za úkol popsat použití vybraných technologií při vývoji aplikace.

Vývoj obecně nikdy nebyl, není a s největší pravděpodobností nikdy nebude o samotných technologiích, které aplikace používá pro svou funkčnost, ale jde především o podpůrné technologie. Podpůrné technologie mohou být vývojová prostředí, dokumentace, tiketové nástroje, kontinuální integrace a nasazení a dodání, testování, běhové prostředí aplikace a mnoho dalších. Kladl jsem velký důraz na integraci a používání výše zmíněných technologií a také představím realizaci této integrace.

Nejdříve popíši integraci technologií z následujícího seznamu. Podpůrné: Gitlab 4.1.1, Docker 4.1.2, Heroku 4.1.3, Intellij IDEA 4.1.4, WebStorm 4.1.5. Serverové: Wildfly 4.2.1, Apache HTTP Server 4.2.2, H2 database 4.2.3. Věřím, že tímto předvedu bonusy které tento ekosystém přináší do vývoje celé aplikace.

Dále rozebírám implementaci serverové části 4.3, uživatelské části 4.4 a administrátorské části 4.5.

## 4.1 Podpůrné technologie

Podpůrné technologie mají zpravidla za úkol ulehčit programátorům, ale i dalším práci na projektu. Nástroje pro správu verzí se staly neodlučitelnou součástí vývoje stejně jako multifunkční editory. Velký boom v komerčním prostředí již několik let zažívá kontejnerizace. Společnou integrací těchto technologií lze dosáhnout velice efektivní práce.

Představme si situaci z pohledu firmy spravující IS sídlící v EU. EU vydá směrnici GDPR, která pro IS znamená relativně drobné změny na vzhledu, ale značně rozsáhlé změny v databázi a byznys logice. Nejdříve se vytvoří úkol v tiketovém systému, ten je rozdělen na tři podúkoly, jelikož úprava zasáhne všechny 3 servery. Plnění úkolů probíhá asynchronně, jelikož servery mají na starosti různí programátoři. Zákazník by rád kontroloval stav pouze tohoto úkolu, proto je potřeba testovací prostředí. Pro každý úkol je ve výsledku

vytvořen balíček aplikace obsahující všechny 3 servery. Tento balíček je potřeba nějakým způsobem spustit, aniž by byl ovlivněn ostatními, již spuštěnými balíčky. Jako tiketový systém jsem si vybral Gitlab ale také pro kontrolování verzí serverů zabalený do balíčků. Navíc Gitlab automaticky spouští vytvořené balíčky v běhovém prostředí. Pro sestavování balíčků jsem vybral technologii Docker a běhové prostředí pro balíčky Docker swarm. Samotné programování probíhalo v editorech IntelliJ IDEA a WebStorm od společnosti JetBrains.

### 4.1.1 Gitlab

Gitlab je open source git repositář od stejnojmenné společnosti a nabídkou funkcionalit zdaleka předčí alternativy jako Bitbucket a Github. Díky možnosti rozřazovat repositáře do skupin jsem vytvořil skupinu, která obsahuje repositáře client, server, database, dokumentace a infrastruktura. První 4 zmíněné mají jasný úkol a pátý slouží pro sestavování balíčků. Balíčky lze stáhnout právě z tohoto repositáře, bohužel jsem nenašel službu která by zdarma poskytovala běhové prostředí pro tyto balíčky.

Gitlab také nabízí vlastní tiketový systém integrovaný s git větvemi, kdy pro každý úkol jsem vytvořil větev. Větve jsem zahrnoval do hlavní větve master pomocí Merge Requestů, v kterých lze mimo jiné nalézt odkaz na Review. Tento způsob práce má pozitivní následky v dalších částech.

Review je funkcionalita umožňující vkládat do Merge Requestů URL odkazy, a jejich nejčastější cíl je verze webové stránky v onom Merge Requstu. Bohužel jsem nenašel poskytovatele umožňujícího hostovat takové množství internetových stránek zdarma, takže odkazy v mém případě vedou vždy na stejný cíl Pages.

Pages je funkcionalita umožňující hostovat pouze statický kód. Jelikož klientská část aplikace se skládá pouze ze statického kódu, neváhal jsem využít této nabídky.

Registry je registr Docker Obrazů, který poskytuje až 15GB úložného prostoru. Obrazy sem nahrávám při každém aktualizaci repositáře, která je detekována kontinuální integrací popsanou v souboru `.gitlab-ci`. Kontinuální integrace v repositářích client nebo server spustí kontinuální integraci v repositáři infrastruktura. Kontinuální integrace v repositáři infrastruktura vytváří balíček v podobě DAB souboru.

Sada těchto funkcionalit mi umožnila vytvořit robustní systém pro testování.

### 4.1.2 Docker

Docker je open source projekt pro práci s kontejnery. Kontejner je odlehčený, soběstačný, spustitelný balíček kusu softwaru, který obsahuje všechno ke spuštění jako: samotnou aplikaci, proměnné prostředí, systémové nástroje, knihovny, nastavení a podobně. [17] Každý kontejner by měl mít pouze jeden účel,

například spustit webový server. Docker je sám o sobě skládačka s následující hierarchií. Základním stavebním kamenem je Dockerfile obsahující textový popis vytvoření Obrazu. Obraz má vlastní jméno a je s nadsázkou řečeno pozastavený a zhmotněný OS sloužící ke spuštění kontejneru. Kontejner je instance Obrazu a může být přiřazen službě. Služba je sada identických kontejnerů přiřazená do Stacku. Stack je sada služeb, které jsou mezi sebou navzájem dostupné, ale odříznuté od zbytku a pro popis Stacku slouží Composefile. Composefile je textový popis spuštění jednoho Stacku v Docker swarmu. Soubor DAB má stejný účel jako Composefile, na rozdíl od něho pro popis, jaké obrazy se mají použít používá hash kódy obrazů, nikoli jména obrazů.

Repositáře client a server obsahují vlastí Dockerfile díky kterému může kontinuální integrace vytvořit obraz a nahrát ho do registru pro pozdější použití.

Soubor DAB jsem využil jako onen balíček, který obsahuje celou aplikaci, toto tvrzení může ovšem klamat. Samotnou aplikaci neobsahuje, pouze její popis ve formě hash kódů. Teprve při spouštění Stacku aplikace dochází ke stahování obrazů z registrů. Spuštění Stacku pomocí souboru Composefile resp. DAB:

```
$ docker stack deploy NAZEV
```

resp.

```
$ docker stack deploy --bundle-file docker-bundle.dab NAZEV
```

### 4.1.3 Heroku

Heroku je platforma jako služba od společnosti Salesforce a nabízí celou řadu produktů. Heroku jsem využil pro hostování serverové části. Heroku není klasický hosting, ale umožňuje spouštět Docker kontejnery, kterým lze přiřazovat takzvané dyno jednotky reprezentující dostupný výkon. Jednoduše řečeno, pokud nemá kontejner přiřazen dyno jednotku, ani se nespustí. Počet dostupných dyno jednotek je určen vybraným plánem a bezplatný plán nabízí pouze jednu dyno jednotku. Pokud je kontejner delší dobu neaktivní a běží v bezplatném plánu, je automaticky vypnut a spuštěn pokud je po něm požadována aktivita. Spuštění kontejneru trvá několik minut, což zapříčiňuje pomalé první načtení klientské části.

### 4.1.4 IntelliJ IDEA

IntelliJ je multifunkční editor od společnosti JetBrains zaměřený na vývoj aplikací v Jave. IntelliJ byl můj primární nástroj na vývoj serverové části aplikace a využíval jsem hlavně dvou jeho skvělých funkcionalit. První funkcionalita se jmenuje Test RESTfull Web Service a umožňuje posílat HTTP požadavky z pohodlí editoru. Druhá funkcionalita se jmenuje Configurations a umožňuje nakonfigurovat různé druhy příkazů a spouštět je opět z pohodlí editoru. Využil jsem hlavně možnosti spustit Docker kontejner a navíc je tato konfi-

gurace nahraná v repositáři, takže jí mohou využívat i další vývojáři. Wildfly server se spustí na adrese `127.0.0.1:8889`.

### 4.1.5 WebStorm

WebStorm je multifunkční editor od společnosti JetBrains zaměřený na vývoj webových aplikací s podporou nejrůznějších populárních frameworků. Všechny editory od společnosti JetBrains obsahují podobné funkcionality takže jsem podobným způsobem využil stejné dvě funkce i zde. Spuštěním Docker kontejneru se spustí server Apache na adrese `127.0.0.1:8888`.

## 4.2 Serverové technologie

Aplikace se zpravidla skládají z několika komponent. Každá komponenta má jinou zodpovědnost a to přináší do struktury aplikace přehlednost, což navyšuje udržitelnost a rozšířitelnost. Komponentizovat se dá na několika úrovních třeba jako rozdělování kódu do funkcí, funkce třídit do objektů, objekty přiřazovat do balíčků, balíčky do logických celků a logické celky do serverů. Sekce se zabývá konkrétními použitými servery a popisem jejich zodpovědnosti. Toto rozdělený vyplývá z cílů a architektury PWA.

### 4.2.1 Wildfly

Wildfly je webový server od společnosti RedHat implementující framework Java EE a je open source odnoží komerční varianty JBoss EAP. Wildfly jsem si vybral především díky jeho podpoře Javy EE8 a nové dokumentaci. Vybral jsem si Wildfly verze 12 varianty full profile, abych nemusel řešit dostupné knihovny. Wildfly jsem si vybral pro poskytování serverové části aplikace respektive REST API.

### 4.2.2 Apache HTTP Server

Apache HTTP Server je webový server od společnosti Apache a jde o nejznámější server na světě. Jeho použití je hodně spjato s jazykem php, přestože jde o multifunkčně založený server. Výkonnostně exceluje především v poskytování statického kódu a tím splňuje mé požadavky na poskytování klientské části aplikace.

### 4.2.3 H2 database

H2 database je open source databázový server od zakladatel Thomase Muellera a vyznačuje se svými módy, v kterých může být spuštěn. Jde o tyto tři módy: Zarytý, Serverový a Kombinovaný. Každý z těchto módů v režimu In memory nebo Persistent. Mimo tyto tři módy, lze databázi spouštět v různých dialektech IBM DB2, Apache Derby, HSQLDB, MS SQL Server, MySQL, Oracle

a PostgreSQL [13]. H2 jsem si vybral kvůli Zarytému módu v režimu In memory a implicitní podporou administrátorské konzole ve webovém prohlížeči. In memory režim uchovává data pouze v paměti RAM a volba tohoto režimu je pouze otázkou konfigurace jednoho řádku kódu. H2 server neběží na odděleném zařízení, ale je zaryt do serveru Wildfly a to znamená, že žije a umírá společně s ním.

## 4.3 Serverová část

Jedna instance serverové části je provozována u poskytovatele Heroku na adrese `https://bakalarska-prace.herokuapp.com/rest/rest/`. Počáteční odpověď může trvat několik minut, jelikož aplikace běží v bezplatném režimu, který je velice pomalý. Zpracování požadavku je možné sledovat po přihlášení do služby Heroku v konzoly, na kterou lze přejít z menu Více. V případě zájmu je možné mě kontaktovat a projekt zpřístupním.

Primárním úkolem serverové části je poskytnout REST API klientské části aplikace, které bude schopno zpracovávat požadavky a korektně odpovídat.

### 4.3.1 Lokální spuštění

Aplikaci je možné spustit třemi způsoby:

1. Vlastní instalace do Wildfly serveru.
2. Ruční spuštění v Docker kontejneru.

   ```
   $ mvn package
   $ docker build -t musiltad/rest .
   $ docker run -it --rm -p 8889:8080 musiltad/rest
   ```

3. Automatické spuštění v Docker kontejneru použitím konfigurace *Wildfly in docker* v editoru IntelliJ IDEA.

### 4.3.2 Struktura

Projekt je rozdělen do tří logických částí: REST API, byznys metody a datová část. Jako sestavovací nástroj jsem si vybral Maven. Projekt je rozdělen do pěti podmodulů 4.3.2 každý s vlastní zodpovědností. Byznys metody a datová část jsou společně v podmodulu ejb a REST API je v podmodulu rest. Podmodul h2admin obsahuje pouze registraci servletu *org.h2.server.web.WebServlet* a jeho konfiguraci a slouží pro vystavení administrátorské části aplikace 4.5. Podmodulem web nemá žádný význam se momentálně zaobírat, jelikož je zde jen pro budoucí rozšíření aplikace a kód psaný v tomto podmodulu by měl být ve frameworku JSF. Hlavní účel podmodulu ear je v konfiguraci a sestavený výsledného archivu ve formátu EAR, který je možné nahrát do Java EE

- `ear` ..... Výsledkem kompilace je ear.ear
- `ejb` .. Obsahuje služby, repositáře a entity. Výsledkem kompilace je ejb.jar
- `h2admin` Vystavuje administrátorskou část aplikace. Výsledkem kompilace je h2admin.war
- `rest` ..... Vystavuje REST API. Výsledkem kompilace je rest.war
- `web` Připraveno do budoucna pro dynamický kód. Výsledkem kompilace je web.war
- `.idea` ..... Konfigurační soubory editoru IntelliJ IDEA

serveru. Navíc v mém případě obsahuje informace o datovém zdroji databáze, které lze najít v souboru `src/main/application/META-INF/server-ds.xml`.

Soubor `persistence.xml` konfiguruje vytvoření struktury databáze na základě entit z balíčku model. Soubor `.gitlab-ci.yml` popis kontinuální integrace pro repositář server. Soubor `Dockerfile` popis vytvoření Docker obrazu pro tento server. Soubor `ear/target/ear.ear` musí před vytvořením obrazu existovat.

### 4.3.3 REST API

Podmodul obsahuje dva důležité typy tříd *Mixins* a *Resource* a konfigurační třídy *CorsFilter*, *ExceptionHandler*, *JaxRsActivator* a *MyObjectMapperProvider*.

*CorsFilter* je JAX-RS Filtr přidávající do HTTP odpovědi tři hlavičky CORS mechanismu. CORS je mechanismus přidávající do HTTP specifikace novou sadu hlaviček umožňující HTTP agentovy přistoupit ke zdrojům z jiného serveru, než z kterého je poskytována zdrojová stránka. [18]

*ExceptionHandler* je JAX-RS ExceptionMapper odchytávající výjimku *EntityAlreadyExistsException*, která nastává, pokud se nižší vrstvy snaží zapsat do databáze entitu, která tam už existuje. Tato výjimka není ponechána bez povšimnutí, ale je převedena do HTTP odpovědi v podobě kódu 409. HTTP kód vyjadřuje konflikt [19]

*JaxRsActivator* je jeden ze způsobu jak zaregistrovat JAX-RS Resource [20]. Specifikace nabízí 3 způsoby:

- Bez třídy dědící od třídy *Application*.
- S třídou dědící od třídy *Application* bez přiřazení konkrétnímu serverletu.
- S třídou dědící od třídy *Application* s přiřazením ke konkrétnímu servletu.

Serializaci objektů do těla HTTP odpovědi mají na starosti poskytovatelé, kterých je celá řada. Já si vybral poskytovatele Jackson, který umí transformovat POJO objekty bez zbytečných anotací, přesto však nabízí anotace pro

komplexnější ovlivnění transformace. Rozhodl jsem se pro přímou transformaci entit z datové části, alternativně jsem mohl vytvořit třídy podle návrhového vzoru DTO. Potřeboval jsem ovlivnit transformaci anotacemi, což by znamenalo kvůli poskytovateli ovlivnit nejspodnější část aplikace, což je označováno za anti-pattern. Naštěstí Jackson má funkcionalitu *Mixins*, což jsou třídy přidávající anotace jiným třídám jejich spárováním. Párování probíhá ve třídě *MyObjectMapperProvider*.

REST API jsem vytvořil podle obecně uznávaných doporučení [21]. Některé ukázky HTTP dotazů je možné nalézt ve složce *JetBrainRestRequests*. Ukázky je možné importovat do nástroje Test RESTfull Web Service a spouštět je. Všechny dotazy je potřeba směřovat na cestu `/rest/rest/` například `http://127.0.0.1:8889/rest/rest/zones`. Následuje dokumentace popisující všechny možné HTTP dotazy. Každý bod obsahuje stručný popis, ty HTTP dotazu, formální popis a příklad použití.

- Seznam všech zón. GET. param: ZONE_ID={0,1,2,3,4,5,6,7,8,9,10}

  `zones[?param[&param...]]`

  `http://127.0.0.1:8889/rest/rest/zones?1=0&2=7&3=3`

- Detail zóny. GET.

  `zones/{ZONE_ID}`

  `http://127.0.0.1:8889/rest/rest/zones/1`

- Přidání zóny. POST. Tělo dotazu je pole ve formátu JSON. coordination: { "lat": FLOAT, "lng": FLOAT }

  `zones`

  ```
  [coordination, coordination, ...]
  ```

  `http://127.0.0.1:8889/rest/rest/zones`

  ```
  [{ "lat": 50.480987, "lng": 14.141818},
  { "lat": 50.480974, "lng": 14.139061}]
  ```

- Seznam všech atributů. GET.

  `specifications`

  `http://127.0.0.1:8889/rest/rest/specifications`

- Seznam všech atributů pro konkrétního uživatele a zónu. GET.

  `specifications/{USER_ID}/{ZONE_ID}`

  `http://127.0.0.1:8889/rest/rest/specifications/0/2`

- Vytvoření nového atributu. POST. zone: { "id": ZONE_ID }

  `specifications`

```
{"name": STRING,
 "creator": { "id": USER_ID },
 "zones": [zone, ...]}
```

`http://127.0.0.1:8889/rest/rest/specifications`

```
{"name": "NAROCNOST",
 "creator": { "id": 0 },
 "zones": [{ "id": 2 }]}
```

- Přidání ohodnocení. POST.

  `rates`

  ```
  {"zone": { "id": ZONE_ID, "type": ZONE_TYPE},
   "specific": { "id": SPEC_ID },
   "creator": {"id": USER_ID},
   "rate" : FLOAT}
  ```

  `http://127.0.0.1:8889/rest/rest/rates`

  ```
  {"zone": { "id": 2, "type": "ROUTE" },
   "specific": { "id": 1 },
   "creator": { "id": 0 },
   "rate" : 1}
  ```

- Aktualiazce ohodnocení. PUT.

  `rates/{RATE_ID}`

  ```
  {"rate": FLOAT}
  ```

  `http://127.0.0.1:8889/rest/rest/rates/1`

  ```
  {"rate": 9}
  ```

### 4.3.4 Byznys metody

Byznys metody jsem implementoval jako EJB třídy. Byznys metody jsou nezávislé na reprezentaci, takže je možné je využívat v REST API a v budoucnu v balíčku web aniž by byly potřeba upravovat. Jak jsem již říkal, byznys metody jsou společně s datovou částí aplikace v podmodulu ejb. Tato realizace koliduje s návrhem a bylo by rozumnější tyto dvě logické komponenty rozdělit do vlastních podmodulů. To jsem však neučinil z důvodu jednodušší realizace.

Ve výsledku jsou metody pouze mezivrstvou REST API a datové části aplikace, jelikož všechny složitější procesy jsem byl schopen realizovat SQL dotazy.

### 4.3.5 Repositáře

Repositáře slouží k získávání výsledků z datového úložiště. Výsledky jsou množiny entit, které jsou uloženy v H2 databázi.

### 4.3.6 Entity

Entity jsou objektová reprezentace struktury úložiště. Na základě entit je při nasazení aplikace na server vytvořena struktura databázových tabulek, které jsou následně inicializovány souborem `import.sql`.

## 4.4 Uživatelská část

Jedna instance uživatelské části je provozována u poskytovatele Gitlab funkcionalitou Pages na adrese `https://bakalarska-prace.gitlab.io/client/`. Odpověď serveru by měla být okamžitá, ovšem může chvíli trvat vykreslení obsahu stránky 4.3, který se načítá ze serverové části aplikace.

Primárním úkolem uživatelské části je poskytnout prezentaci resp. sběr geografických dat, které jsou získávány resp. ukládány v serverové části aplikace.

Uživatelská část bakalářské práce je nejrozsáhlejší částí projektu a věnoval jsem jí největší úsilí, bohužel jsem musel některé části z návrhu při implementaci omezit. Projekt jsem vypracovával tím způsobem, aby bylo možné jej provozovat bez nutnosti placených služeb, což přineslo některý omezení návrhu.

Hlavním cílem bakalářské práce bylo na základě analýzy vytvořit vzorovou aplikaci. Z toho důvodu má aplikace pouze střídmý vzhled, ale přesto splňuje návrh. Postranní menu základního rozložení se ukázalo jako přebytečné a proto obsahuje pouze odkaz na informace o projektu a odkaz na domovskou stránku. Ostatní tlačítka se nachází v akčním menu v hlavičce.

Uživatelská část byla optimalizována pro internetový prohlížeč Chrome. Google je hlavním propagátorem architektury PWA tudíž podpora technologií, které PWA vyžaduje, je největší v jejich prohlížeči Chrome. Z toho důvodu doporučuji testovat aplikaci právě v tomto prohlížeči, aby bylo možné využít všech moderních technologií. Podpora ostatních vyhledávačů poroste časem, tím jak budou dodatečně implementovat chybějící technologie.

### 4.4.1 Lokální spuštění

Aplikaci je možné spustit třemi způsoby:

1. Vlastní instalace do Apache serveru.
2. Ruční spuštění v Docker kontejneru.

```
$ bower install
$ docker build -t musiltad/client .
$ docker run -it --rm -p 8888:80 musiltad/client
```

3. Automatické spuštění v Docker kontejneru použitím konfigurace *Apache in docker* v editoru WebStorm.

### 4.4.2 Struktura

Http servery v základu odbavují provoz na základu adresářové struktury a výchozího souboru s názvem `index.html`. Projekt využívá této vlastnosti pro zavedení 4 URL adres `/`, `/add-zone`, `/detail-zone` a `/filter`. Každá z těchto složek obsahuje zaváděcí soubor `index.html`, který zajišťuje zavedení: scriptu `/service-worker.js` 4.4.7, scriptu `https://cdnjs.cloudflare.com/ajax/libs/webcomponentsjs/1.0.20/webcomponents-lite.js`, json souboru `manifest.json` 4.4.12 a hlavní komponenty vykreslující obsah stránky.

Aplikace byla rozdělená do komponent pomocí sady čtyř standardů souhrnně označovaných jako Web Components. Všechny tyto čtyři standardy jsou na sobě nezávislé a lze použít jejich libovolnou podmnožinu, avšak jejich vzájemné použití je nejčastější.

- HTML Template - je standard popisující element *template*. Obsah elementu se na stránce nesmí vykreslit. Používá se pro dodání kusu html kódu do stránky pro pozdější použití. [8]

- Custom elements - je standard pro vytváření vlastních html tagů pomocí javascriptu. Název tagu musí obsahovat pomlčku. [5]

- Shadow DOM - je standard pro popisující způsob zapouzdření část DOMu. Zapouzdřený DOM nelze ovlivnit zvenčí a zároveň obsah zapouzdřeného DOMu nemůže ovlivnit vnější okolí. [6]

- HTML Imports - je standard rozšiřující funkci elementu *link* o popsání závislostí stránky na dalších zdrojích. [7]

Složka `components` obsahuje definice všech elementů vytvořených v rámci práce. Podsložka `components/main` obsahuje elementy potřebné zavaděči.

Soubor `.gitlab-ci.yml` popis kontinuální integrace pro repositář client.

Soubor `Dockerfile` popis vytvoření Docker obrazu pro tento server.

```
components ..... Vlastní vytvořené elementy
    main ..... Elementy zavaděčů
    buttons ..... Elementy tlačítek
    inputs ..... Elementy vstupů
bower_components ..... Knihovny třetích stran
images ..... Loga ČVUT
scripts ..... Všemožné javascripty
.idea ..... Konfigurační soubory editoru WebStorm
```

### 4.4.3 Úvod

Na úvodní stránce má uživatel možnost vidět seznam evidovaných zón. Seznam je tvořen kartami, na většině plochy zobrazují náhled do mapy. Modře jsou zvýrazněné trasy a červeným puntíkem je označená trasa definovaná bodem v mapě. Pod náhledem do mapy je možné vidět název zóny. Název zóny je reprezentován jejím identifikačním číslem z databáze tak, aby jej uživatel nemusel zadávat. Pod názvem zóny je obecné ohodnocení. Obecné hodnocení se počítá aritmetickým průměrem přes všechny ohodnocení zóny.

Seznam je možné prostřednictvím tlačítka v pravém horním rohu přepnout na zobrazení tras v mapě 4.1. Trasy jsou podobně jako v náhledu vyznačeny barvou. Poskytovatelem mapy je společnost Google a integraci do aplikace jsem provedl elementy *google-map*, *google-map-directions* a *google-map-point*.[7]

Na stránce je možné najít odkazy na všechny podstránky, tak jak jsem popsal v návrhu.

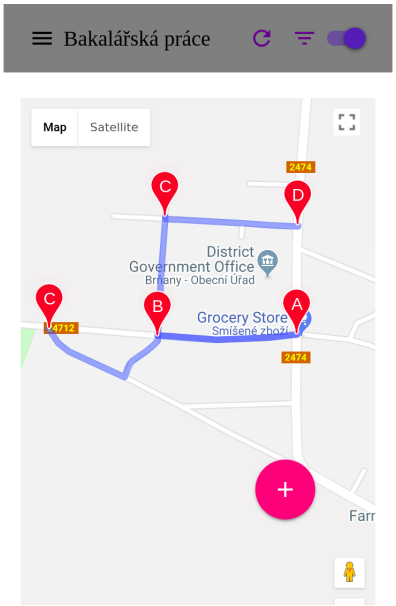
Struktura této stránky je definována elementem *app-teddy-main-index* jehož definici je možné najít v souboru `components/main/index.html`.

### 4.4.4 Filtr

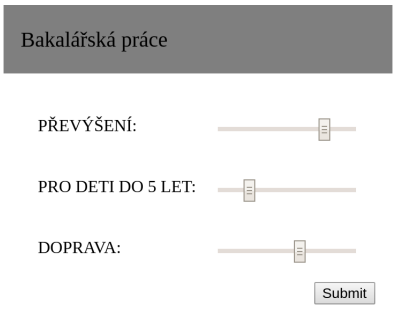
Na podstránce filtru 4.2 se zobrazí všechny atributy evidované v systému. Každý atribut má na své pravé straně vstup ve formě posuvníku, kterým lze určit nejmenší možnou hodnotu, kterou musí zóna v daném atributu splnit. Pokud mě nezajímá, jak se zóně v daném atributu daří, nechám ho na nule, což splňuje každá zóna, i ty které ještě v daném atributu nebyly ohodnoceny. Atributy je možné hodnotit na škále od 0 do 10. Posuvník vlevo vyjadřuje hodnotu 0 a naopak vpravo hodnotu 10.

Struktura této stránky je definována elementem *app-teddy-main-filter* jehož definici je možné najít v souboru `components/main/filter.html`.

[7] Zde je možné najít bližší popis všech poskytovaných elementů společností Google `https://www.webcomponents.org/author/GoogleWebComponents`




Obrázek 4.1: Úvodí stránky




Obrázek 4.2: Podstránky filtru


Obrázek 4.3: Podstránky detailu zóny

### 4.4.5 Detail

Na podstránce detailu 4.3 lze najít podrobnější informace o zóně. Mezi podrobnější informace o zóně jsem vybral informace na souhrnu plus jméno uživatele, který zónu přidal a typ zóny.

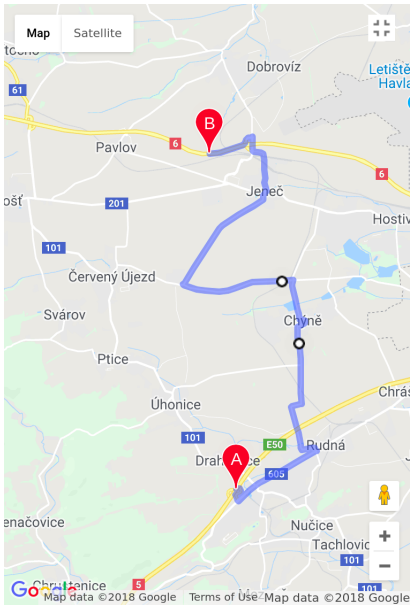
Pokud jsou k zóně přiřazeny nějaké atributy, je možné je najít vpravo od detailních informací. Zeleně jsou označeny ty atributy, podle kterých uživatel již ohodnotil zónu a červeně ty podle kterých ještě ne. Přesunutím posuvníku na jinou pozici se provede aktualizace ohodnocení uživatelem. Hodnoty na posuvníku vyjadřují stejnou škálo jako v 4.4.4. V případě že zde není dostupný atribut, který je již v systému evidován, je na administrátorovi, aby jej přiřadil k zóně 4.5.

Pod seznamem atributů je vstupní pole pro přidání nového atributu. Přidávat je možné pouze ještě neexistující atributy.

Struktura této stránky je definována elementem *app-teddy-main-detail* jehož definici je možné najít v souboru `components/main/detail.html`.

### 4.4.6 Přidání

Stránka 4.4 umožňuje přidávat nové zóny typu trasa do systému. Jako popis mechanizmu vybrání nové trasy uvedu příkladem: Prvním kliknutím do mapy vyberete počátek trasy a druhým její konec. Trasa se může skládat až z dvanácti mezikroků. Mezikrok přidáte kliknutím do vybrané trasy a jejich pře-




Obrázek 4.4: Podstránky přidání nové zóny

táhnutím se změní tvar trasy. Uložení trasy se provede kliknutím na fajfku v pravém dolním rohu.

Struktura této stránky je definována elementem *app-teddy-main-add* jehož definici je možné najít v souboru `components/main/add.html`.

### 4.4.7 Instalovatelnost

Instalovatelnost je jednou z hlavních předností uživatelské části aplikace. Zodpovědnost za nabídnutí instalace uživateli nese internetový prohlížeč, který má ovšem několik požadavků, které musí být splněny před nabídkou. Každý prohlížeč podporující tuto vlastnost může mít různé požadavky, ale neměly by se zásadně lišit. Jakmile jsou splněny požadavky, nabídka se projeví ve formě vyskakovacího okna v dolní části aplikace v prohlížeči Chrome s tlačítkem *PŘIDAT NA PLOCHU*. Instalaci lze manuálně vyvolat v kontextové nabídce v pravém horním rohu volbou *Přidat na plochu*.

Požadavky internetového prohlížeče Chrome na uživatele:

1. Aplikace ještě není nainstalována.
2. Uživatel má dostatečnou účast na aplikaci. V zásadě stačí, aby uživatel strávil na stránce alespoň 30 vteřin. [8]
3. Dostatek místa na disku

[8] `https://www.chromium.org/developers/design-documents/site-engagement`

Požadavky internetového prohlížeče Chrome na aplikaci:

1. Odkazovaný Web app manifest obsahuje následující položky: *short_name* nebo *name*, *icons* velikosti 192px a 512px, *start_url*.

2. Stránka je poskytována přes HTTPS.

3. Stránka zaregistruje script Service Worker, který poslouchá událost *fetch*.

Důležitá poznámka: od verze internetového prohlížeče Chrome 68 [9] nebude prohlížeč nabízet vyskakovací okno automaticky, ale tato zodpovědnost bude přenesena na programátora. [22]

Listing 4.1: Ukázka Web app manifest.

```
{
  "name": "Bakalarska prace",
  "short_name": "Bakalarka",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "white",
  "theme_color": "white",
  "icons": [{
    "src": "images/icons/symbol/symbol_cb/
       symbol_cvut_plna_samostatna_verze_cb.png",
    "sizes": "128x128 144x144 152x152 192x192 256x256",
    "type": "image/png"
  },{
    "src": "images/icons/symbol/symbol_cb/
       symbol_cvut_plna_samostatna_verze_cb.svg",
    "sizes": "144x144",
    "type": "image/svg"
  }]
}
```

### 4.4.8 Bezpečnost

Web je provozován u poskytovatele Gitlab funkcionalitou Pages, který automaticky všechny stránky poskytuje zabezpečeným protokolem HTTPS.

[9] Vypuštění se očekává v květnu 2018

Listing 4.2: Ukázka Service Worker.

```
self.addEventListener('install', function (event) {
    // spusteno pouze pri instalaci noveho service
       workera
});

self.addEventListener('activate', function (event) {
    // spusteno pouze pri aktivaci noveho service
       workera
    // nastane pouze pri zavreni vsech oken pouzivajici
       starou verzi
});

self.addEventListener('fetch', function (event) {
    // Bypase / forward / CSR / edit response / edit
       request / ...
});
```

### 4.4.9 Nezávislost na internetovém připojení

Druhou předností aplikace je právě nezávislost na internetovém připojení. Od nativní aplikace očekáváme, že bude fungovat i bez internetového připojení a zároveň rychle. Těchto dvou vlastností lze dosáhnout několika přístupy, které lze navzájem kombinovat. Zároveň je potřeba si uvědomit, jak budeme zacházet se staženými zdroji do prohlížeče a zároveň jak budeme odbavovat odchozí požadavky ze stránky.

V každém případě je použito Cache API, fungující jako mapa (klíč hodnota), kde klíčem je HTTP požadavek a hodnotou je HTTP odpověď. Cache je nejdříve zapotřebí naplnit aby dávalo smysl její následné využití. První seznam se zabývají způsoby naplnění a druhý seznam způsoby využití Cache API. [23]

Naplnění Cache:

1. Při instalaci service workera jako závislost - Při instalaci service workera je Cache inicializována všemi potřebnými závislostmi a pokud jakákoliv závislost není možná stáhnout, service worker není nainstalován.

2. Při instalaci service workera jako volitelná závislost - Podobné předchozímu bodu, ale navíc je instalace obohacena dobrovolnými závislostmi, které případně nezpůsobí selhání instalace.

3. Při aktivaci service workera - Používá se zejména pro aktualizace Cache v případě, že dojde k aktualizaci webu.

4. Na základě uživatelské interakce - Cache je naplněna na vyžádání uživatelem, například když uživatel chce uložit článek pro pozdější shlédnutí.

5. Pouze jednou na základě požadavku síťové komunikace - Cache je postupně naplňována stahovanými zdroji prohlížečem, v průběhu toho co uživatel prohlíží web. Ve výsledku mohou být offline poskytovány stránky, které uživatel již navštívil.

6. Pokaždé na základě požadavku síťové komunikace - Cache je aktualizována pokaždé, když dojde k požadavku na síťovou komunikaci.

7. Na základě inicializace komunikace serverem - Jsme zvyklí, že HTTP komunikace je vždy inicializována klientem, avšak tato bariéra je v dnešní době již překonána a existují technologie umožňující i opačnou komunikaci. Tento přístup využívá notifikace ze strany serveru, který slouží jako požadavek na aktualizování nebo dodání určitých zdrojů do Cache.

8. Při připojení zařízení do sítě - Pokud zařízení není momentálně připojeno do sítě, požadavky na síťovou komunikaci mohou být střádány. Poté co se zařízení připojí do sítě, mohou být nastřádané požadavky předány dále.

Implementoval jsem body 1 a 5.
Využití Cache:

1. Vracení odpovědí pouze z Cache - Přístup respektuje pouze to, co bylo již nahráno v Cache a pro ostatní zobrazuje chybové hlášení.

2. Ignorování obsahu Cache, pouze síťová komunikace - Přístup je opakem předchozího bodu a umožňuje fungování pouze v offline režimu.

3. Pokud nenaleznu odpověď v Cache, použij síťovou komunikaci - Pokud odpověď na HTTP požadavek není nalezena v Cache, použije síťovou komunikaci k získání odpovědi. Odpověď ze síťové komunikace ale není uložena do Cache.

4. Nejdříve síť, pokud ta selže, pak použij Cache - Tento přístup je opakem předchozího bodu.

5. Paralelně síťová komunikace a Cache, použij rychlejší odpověď - Cache je uložena na disku a jak je známo, přístup na disk je pomalý. Tento přístup vrátí odpověď, která přijde dříve.

6. Paralelně síťová komunikace a Cache, použij nejdříve Cache pak aktualizuj ze sítě - Metoda je vhodná pro zařízení s rychlím čtením z disku. Nejdříve vrátí odpověď nalezenou v Cache a při získání odpovědi ze sítě vyvolá automatickou aktualizaci stránky.

7. Informativní stránka o nedostupnosti - Tato metoda lze kombinovat se všemi předchozími a popisuje, jak by se aplikace měla zachovat, pokud není možné získat odpověď. V takovém případě je možné například vrátit vlastní chybové hlášení.

Pro komunikaci s Gitlab Pages jsem implementoval body 1 a 7 a pro komunikaci s Heroku a Google Maps jsem implementoval bod 4. 4.3

Na základě implementace bodů z předchozích seznamů aplikace částečně funguje v offline režimu. V offline režimu jsou zpřístupněny pouze stránky, které uživatel někdy navštívil v online režimu. Tím pádem lze prohlížet například detaily zón nebo použít konkrétní filtry, které již byly někdy prohlídnuty.

Cache API obsahuje 3 hlavní metody: *open* pro otevření nebo vytvoření mapy, *match* pro vrácení HTTP odpovědi na parametru v podobě HTTP požadavku, *put* pro vložení nového páru HTTP požadavek a HTTP odpověď.

### 4.4.10 Responzivita

Polymer je framework zjednodušující práci s architekturou PWA a technologií Web components. Lze jej využít pro vytváření celých aplikací, ale také pouze pro vytváření vlastních znovupoužitelných elementů. Polymer poskytuje několik vlastních elementů pro vytváření responzivního designu. Sadu těchto elementů jsem použil pro vytvoření základního rozložení aplikace. [10] Důležité elementy kterých jsem využil jsou *app-drawer-layout*, *app-drawer*, *app-header-layout*, *app-header* a *app-toolbar*.

Postranní menu se ukázalo jako nepotřebné, ale pro ukázku jsem ho ponechal. Na velkých obrazovkách je neustále zobrazováno a na malých obrazovkách ho lze rozvinout prostřednictvím tlačítka v levém horním rohu.

Karty na seznamu zón se na velkých obrazovkách skládají vedle sebe, ale na malých obrazovkách se vyskládají pod sebe.

### 4.4.11 Rychlá odezva

Aplikace se skládá kompletně ze statického kódu, tudíž její první načtení je téměř okamžité. Značná část stránky je sdílena s ostatními podstránkami (například soubory definující rozložení stránky), tudíž jsou načítány z Cache, díky čemuž jsou ještě nenavštívené stránky načteny mnohem rychleji.

Načítání aplikace by bylo možné ještě více urychlit technologií Server Push protokolu HTTP2. Technologie Server Push odesílá v rámci HTTP odpovědi nejen požadovaný zdroj, ale také zdroje spjaté s požadovaným zdrojem. [24] Abych ovšem mohl tuto technologii použít, musel bych využít jiného poskytovatele webového serveru než Gitlab.

[10] `https://www.webcomponents.org/element/PolymerElements/app-layout`

Listing 4.3: Ukázka implementace nezávislost na internetovém připojení.

```
if (requestURL.origin === location.origin)
    event.respondWith(caches
            .open(CACHE_NAME_ASSET)
            .then(function (cache) {
                return cache
                    .match(event.request)
                    .then(function (response) {
                        return response || fetch(event.
                            request).then(function (
                            response) {
                             cache.put(event.request,
                                response.clone());
                             return response;
                        });})
                    .catch(function () {
                        return cache.match('offline.html
                            ');
                    });}));
else
    event.respondWith(
        caches.open(CACHE_NAME_DATA).then(function (
            cache) {
             return fetch(event.request)
                 .then(function (response) {
                     cache.put(event.request, response.
                         clone());
                     return response;
                 })
                 .catch(function () {
                     return cache.match(event.request);
                 });}));
```

### 4.4.12 Vyhledatelnost

Vyhledatelnosti v internetovém vyhledávači jsem zatím bohužel nedosáhl. Internetový vyhledávač nabízí stránky na základě jejich obsahu, tak jak ho vidí uživatel. Pro analýzu obsahu stránky využívá takzvané *Crawlery*, kteří většinou neumějí spouštět javascriptové části webu. Vzhledem k tomu, že stránka využívá zásadním způsobem javascriptové technologie, Crawleři nejsou schopní stránku indexovat.

Nastíněný problém lze vyřešit metodami SSR nebo Prerendering, které

jsou ovšem rozsáhlou kapitolou sami o sobě a již pro ně nebyl prostor.

Funkce sdílení stránky problémem nebyl. Všechny URL odkazy jsem vytvořil takovým způsobem, aby je bylo možné přidávat do záložek. Všechny potřebné informace pro načtení stránky jsou obsaženy v URL parametrech odkazu.

## 4.5 Administrátorská část

Instance administrátorské části je provozována na stejné instanci jako serverová část na adrese `https://bakalarska-prace.herokuapp.com/h2admin` a platí pro ní stejná omezení 4.3. Z toho důvodu její první načtení může nějaký moment trvat.

Primárním úkolem aplikace je umožnit administrátorům spravovat přiřazení atributů zónám. Považuji tuto možnost za důležitou, jelikož by se jednoduše mohlo stát, že uživatelé budou přiřazovat zónám nesmyslné atributy. Takovému chování je zajisté potřeba zamezit.

Abych zjednodušil realizaci a nemusel vytvářet databázový server odděleně, zabudoval jsem aplikaci do serverové části. V ideálním případě jsem mohl aplikaci vytvořit jako samostatný server, to by ovšem z technických důvodů znamenalo, že by databázový server nemohl být zakořeněn do serverové části.

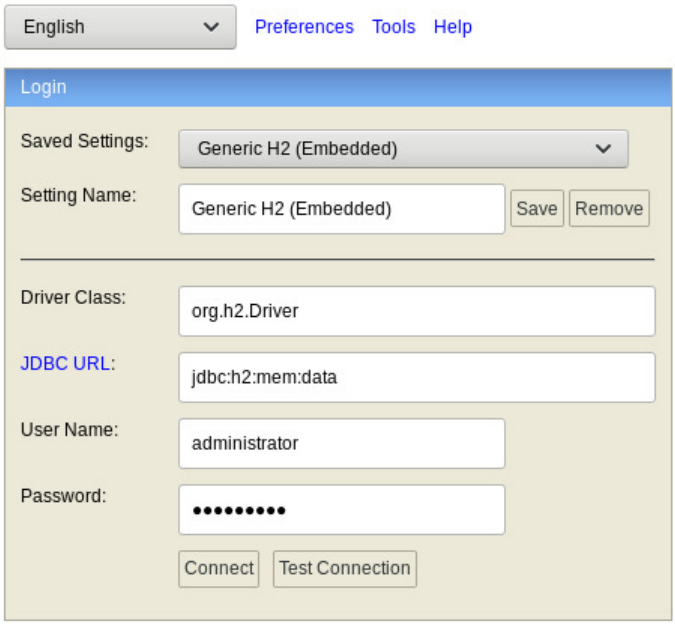
Aplikace je pouze nasazením H2 Console. H2 Console je grafický databázový klient poskytovaný webový serverem, který je tím pádem přístupný skrze internetový prohlížeč. [25] Na rozdíl od uživatelské části, zde jde o dynamický kód, jelikož neočekávám velkou vytíženost.

Po grafické stránce jsem klienta nijak neupravoval.

### 4.5.1 Přihlášení do aplikace

Abych administrátora webu oddělil od administrátora databáze (přihlašovacím jméno *sa*), vytvořil jsem nového uživatele *administrator* s heslem *123456789*. Vzhledem k tomu, že stále jde pouze o databázového klienta, je potřeba zadat URI adresu cílové databáze, která je *jdbc:h2:mem:data*. 4.5 Administrátorovi jsem omezil práva tak, aby mohl provádět všechny úpravy nad tabulkou *ZONE_SPECIFIC* a zobrazovat obsah tabulek *ZONE* a *SPECIFIC*. Uživatele a práva jsem vytvořil tímto způsobem 4.5.1

```
CREATE ROLE roleAdministrator;
CREATE USER administrator PASSWORD '123456789';
GRANT roleAdministrator TO administrator;
GRANT ALL ON ZONE_SPECIFIC TO roleAdministrator;
GRANT SELECT ON ZONE TO roleAdministrator;
GRANT SELECT ON SPECIFIC TO roleAdministrator;
```




Obrázek 4.5: Ukázka přihlášení administrátora

### 4.5.2 Editace přiřazení atributů k zónám

Editace přiřazení je dostupná prostřednictvím spuštění následujícího příkazu:

```
SELECT * FROM ZONE_SPECIFIC
```

Zobrazí se tabulka s přiřazeními, kterou je možné začít editovat prostřednictvím tlačítka *edit*.

# Závěr

Cílem bakalářské práce bylo seznámit se s problematikou multiplatformního vývoje mobilních aplikací. Na základě této analýzy vybrat jedno konkrétní řešení, navrhnout vzorovou aplikaci na téma sport a provést její realizaci. Vzorová aplikace měla sloužit uživateli k vyhledání tras podle specifických kritérií a přidávání nových tras. Dále měla uživateli umožnit hodnocení trasy podle specifických kritérií a přidávaní nových kritérií.

Bylo potřeba analyzovat tři metody přístupu k multiplatformnímu vývoji mobilních aplikací. Metody zahrnovaly celkem sedm frameworků, u nichž bylo potřeba pochopit jejich základní principy fungování. Do analýzy jsem zároveň vepsal popis těchto základních principů.

Na základě požadavků na aplikaci jsem provedl návrh, do kterého jsem popsal obecné vlastnosti nativních aplikací.

Výsledkem práce je webová aplikace instalovatelná do chytrých zařízení s operačním systémem Android, iOS a Widnows Phone. Aplikace umožňuje uživateli vyhledávat trasy a přidávat nové. Trasy je možné zobrazit na mapě nebo v jednoduchém seznamu. Trasám je možné přiřazovat atributy podle kterých je možné filtrovat, ale také vyjádřit spokojenost. Administrátorům aplikace je umožněno spravovat přiřazení atributů k trasám.

Aplikace není distribuována skrze obchody s aplikacemi, ale přes internetové prohlížeče. Aplikace byla optimalizována pro internetový prohlížeč Chrome.

Možností jak aplikaci rozšířit nebo modifikovat je celá řada. Vítanou vlastností by jistě bylo vyhledávání skrze internetový vyhledávač a vytvoření přívětivějšího designu. Administrátoři by především uvítali intuitivnější rozhraní pro správu přiřazování atributů k trasám. Aplikace by se také dala přesunout z poddomény společnosti Gitlab na vlastní doménu a servery na placené varianty tak, aby uživatel nemusel dlouho čekat na načtení obsahu. Uživatelský komfort by jistě navýšilo pojmenování tras a plánování výletů. Pro lepší informovanost uživatelů by přispěla integrace se službami třetích stran zaměřených na mapové podklady nebo dopravní informace. V neposlední řadě by bylo

vhodné vytvořit návod na použití aplikace.

# Literatura


[1] 5 ways the web will evolve in 2018. *V: O'Realy - elektrotechnický magazín a nakladatelství [online]*, leden 2018, [cit. 2018-5-10]. Dostupné z: https://www.oreilly.com/ideas/5-ways-the-web-will-evolve-in-2018

[2] formatkaka; mfuji09; Jakubem; aj.: Service Worker API. [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

[3] Caceres, M.; Christiansen, K. R.; Lamouri, M.; aj.: Web App Manifest. [online], [Accessed: 10.5.2018]. Dostupné z: https://www.w3.org/TR/appmanifest/

[4] chrisdavidmills; Sheppy; jamesdhurd; aj.: Add to Home screen. [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/Apps/Progressive/Add_to_home_screen

[5] Google, I.: Custom Elements. [online], [Accessed: 12.5.2018]. Dostupné z: https://w3c.github.io/webcomponents/spec/custom/

[6] Ito, H.: Shadow DOM. [online], [Accessed: 12.5.2018]. Dostupné z: https://www.w3.org/TR/shadow-dom/

[7] Glazkov, D.; Morrita, H.: HTML Imports. [online], [Accessed: 12.5.2018]. Dostupné z: https://www.w3.org/TR/html-imports/

[8] Apple; Google; Mozilla; aj.: HTML Living Standard. [online], [Accessed: 12.5.2018]. Dostupné z: https://html.spec.whatwg.org/multipage/scripting.html

[9] Google: Cards. [online], [Accessed: 10.5.2018]. Dostupné z: https://material.io/design/components/cards.html

[10] chrisdavidmills; cpxPratik; vaindil; aj.: input type='number'. [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number

[11] Coyier, C.: Star Ratings With Very Little CSS. [online], [Accessed: 10.5.2018]. Dostupné z: https://css-tricks.com/star-ratings/

[12] Biiinggg; Jedipedia; barryels; aj.: input type='range'. [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/range

[13] Mueller, T.; group, H.: Comparison to Other Database Engines. [online], [Accessed: 10.5.2018]. Dostupné z: http://www.h2database.com/html/features.html

[14] Ater, T.: *Building Progressive Web Apps*. O'Reilly Media, ISBN 9781491961643.

[15] chrisdavidmills; poshaughnessy; end3r; aj.: Introduction to progressive web apps. [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/Apps/Progressive/Introduction

[16] Wagner, J.: Why Performance Matters. [online], [Accessed: 10.5.2018]. Dostupné z: https://developers.google.com/web/fundamentals/performance/why-performance-matters/

[17] Inc., D.: WHAT IS CONTAINER. [online], [Accessed: 10.5.2018]. Dostupné z: https://www.docker.com/what-container

[18] jonicious; dhurlburtusa; AndreiIgna; aj.: Cross-Origin Resource Sharing (CORS). [online], [Accessed: 10.5.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[19] Fielding, R.; Irvine, U.; Gettys, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. [online], [Accessed: 12.5.2018]. Dostupné z: https://tools.ietf.org/html/rfc2616

[20] Bucek, P.; Pericas-Geertsen, S.: JAX-RS: Java™ API for RESTful Web Services. [online], [Accessed: 12.5.2018]. Dostupné z: http://download.oracle.com/otndocs/jcp/jaxrs-2_1-final-eval-spec/index.html

[21] Richardson, L.; Ruby, S.; Amundsen, M.: *RESTful Web APIs*. O'Reilly Media, ISBN 9781449359713.

[22] LePage, P.: Add to Home Screen. [online], [Accessed: 12.5.2018]. Dostupné z: https://developers.google.com/web/fundamentals/app-install-banners/

[23] Archibald, J.: The Offline Cookbook. [online], [Accessed: 12.5.2018]. Dostupné z: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/

[24] Wagner, J.: A Comprehensive Guide To HTTP/2 Server Push. [online], [Accessed: 12.5.2018]. Dostupné z: https://www.smashingmagazine.com/2017/04/guide-http2-server-push/

[25] Mueller, T.; group, H.: The H2 Console Application. [online], [Accessed: 13.5.2018]. Dostupné z: http://www.h2database.com/html/quickstart.html

Příloha **A**

# Seznam použitých zkratek

**OS** operační systém

**PWA** progresivní webové aplikace

**PRPL** push render pre-cache lazy-load

**API** application program interface

**UI** user interface

**AOT** Ahead-of-time

**JDK** java development kit

**JRE** java runtime environment

**CLI** command line interface

**PC** personal computer

**DOM** document object model

**EA** enterprise architect

**HTTP** hyper text transfer protocol

**HTTPS** hyper text transfer protocol secure

**REST** representational state transfer

**A2HS** add to home screen

**SSR** server side rendering

**IS** informační systém

**EU** evropská unie

**GDPR** general data protection regulation

**DAB** distributed application bundle

**RAM** random access memory

**WAR** web application resource

**EJB** enterprise java beans

**EAR** enterprise application archive

**JPA** java persistence api

**JSF** java server faces

**CORS** cross-origin resource sharing

**POJO** data transfer object

**JSON** javascript object notation

**SQL** structured query language

**URL** uniform resource locator

Příloha **B**

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `server` ..... zdrojové kódy implementace REST API
    - `client` ..... zdrojové kódy implementace klientské části
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS