Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Multiplatformní mobilní klientská aplikace pro geosociální sítě

Václav Škorpil

Bakalářská práce

2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Václav Škorpil**
Osobní číslo:	**I18196**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Multiplatformní mobilní klientská aplikace pro geosociální sítě**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce bude v teoretické části charakterizovat a popsat geosociální sítě a analyzovat využitelnost jejich API pro vlastní mobilní aplikaci v oblasti cestovního ruchu. Součástí práce bude i přehled dostupných rozhraní API geosociálních sítí, jejich SWOT analýza.
Druhým cílem v praktické části bakalářské práce bude navrhnout a realizovat vlastní multiplatformní mobilní aplikaci (např. v Adobe PhoneGap/Xamarin/Flutter). Ta bude využívat API vybraných geosociální sítí, kde na základě geolokačních technologií uživateli poskytne s využitím mapových podkladů informace o blízkých bodech zájmu (POI) z geosociálních sítí.

Rozsah pracovní zprávy: **35**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

- ARNAUDO, D., MONACO, N. *Data Analytics for Social Media Monitoring*. Washington, D.C., United States: National Democratic Institute (ndi.org), 2020.
- KYSELA, J.: *Comparison of Web Applications Geolocation Service*. In: IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI 2014). Budapešť: Óbuda University, 2014. ISBN 978-1-4799-5338-7.
- KYSELA, J., HORÁLEK, J., HOLÍK, F.: *Measuring Information Quality of Geosocial Networks*. In: New Trends in Intelligent Information and Database Systems. Springer, 2015. ISBN 978-3-319-16211-9.

Vedoucí bakalářské práce: **Ing. Jiří Kysela, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2020**
Termín odevzdání bakalářské práce: **14. května 2021**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 26. února 2021

Prohlašuji:
Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 4. 2021 Václav Škorpil

Poděkování

Tímto bych rád poděkoval své rodině a přítelkyni za podporu při studiu i v době psaní bakalářské práce. Také bych poděkoval mému vedoucímu práce Ing. Jiřímu Kyselovi, Ph.D. za pomoc a konzultace při psaní bakalářské práce.

Anotace

Bakalářská práce se zabývá analýzou geosociálních sítí a využitelností jejich aplikačních rozhraní pro multiplatformní mobilní aplikaci a následné vytvoření takovéto aplikace. V práci jsou analyzována 3 aplikační rozhraní a ke každému z nich je vytvořena SWOT analýza. Aplikace je vytvořena ve frameworku Flutter s použitím návrhového vzoru BloC a vrstvené architektury. Aplikace slouží k vyhledávání restaurací a zjištění informací o nich, v blízkosti stanovené lokace. Uživatel také může filtrovat výsledky dle kategorií, vzdálenosti, cenovou hladinou nebo textem.

Klíčová slova

POI, Yelp, GraphQL, cestovní ruch, multiplatformní aplikace, mobilní aplikace, geosocíální sítě, Flutter, geosocíální aplikace.

Title

Multiplatform Mobile Client Application for Geosocial Networks.

Annotation

Thesis aims to analyse usability of geosocial application interfaces in multiplatform mobile application and implement such application. In thesis are analysed 3 geosocial network's application interfaces. Application is built on top of Flutter framework, with usage of BloC design pattern and domain driven architecture. Purpose of application is to search restaurants and information about them near specified location. User can also filter results by category, distance, price level and text.

Keywords

POI, Yelp, GraphQL, tourism, multiplatform application, mobile application, geosocial network, Flutter, geosocial application.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

HTTP	Hypertext Transfer Protocol
CI/CD	Continuous integration and continuous deployment
API	Application programming interface
REST	Representational state transfer
POI	Point of interest
SDK	Software developement kit
SWOT	Strenghts, weaknesses, opportunities, threats
JSON	JavaScript object notation
XML	Extensible markup language
IP	Internet protocol
BloC	Business logic component
DevOps	Developement and operations
VCS	Version control system
URL	Uniform resource locator

ÚVOD

Dostupnost mobilních zařízení s geolokačními službami a mobilním internetem, výrazně ovlivnila oblast cestovního ruchu. Velkým přínosem byl vznik geosociálních sítí, které umožňují jednoduše vyhledávat a získávat informace o bodech zájmů. V dnešní době, pokud se člověk nachází v nové lokaci, kde to nezná a dostane chuť na kafe nebo něco k jídlu, nezřídka sáhne po mobilním telefonu a použije nějakou aplikaci geosocíální sítě, která mu pomůže se lepé rozhodnout kam zajít.

Cílem práce je, analyzovat dostupná aplikační rozhraní geosociálních sítí a vytvořit multiplatformní aplikaci využívající aplikační rozhraní geosociální sítě. V prvních 2 kapitolách se autor zabývá geosociálními sítěmi a aplikačními rozhraními 3 geosociálních sítí. Foursquare, Yelp a Google Places. U aplikačních rozhraní těchto sítí, jsou pro aplikaci důležitá hlavně data poskytovaná aplikačním rozhraním, jejich dostupnost, cena a smluvní podmínky.

Další kapitoly se zaměřují na mobilní aplikaci. Aplikace je realizována ve frameworku Flutter, ke kterému je probrán teoretický základ ve 3. kapitole. V aplikaci jsou zobrazovány body zájmu z aplikačního rozhraní geosociální sítě Yelp. K datům je přistupováno přes GraphQL, které je stručně popsáno ve 4. kapitole. Pátá kapitola se zabývá samotným návrhem a vývojem mobilní aplikace. Při vývoji a návrhu je kladen důraz na uživatelské rozhraní, jeho přívětivost a plynulost. Součástí této kapitoly je i vydání aplikace na Google Play.

1 GEOSOCIÁLNÍ SÍTĚ

1.1 Popis a charakteristika

Geosociální sítě jsou sítě, které slouží k propojení lidí stejně jako sociální sítě, a navíc používají geolokační technologie. Předpokladem pro vznik geolokačních sítí, jsou zařízení umožňující získávat data o poloze uživatele. Geolokace se může používat k různým účelům, nejčastěji se používají k svázání příspěvku s určitým místem. Takto poté vznikají takzvané body zájmu, často se označují jako POI, z anglického sousloví Point of interest.

Body zájmů mohou být restaurace, bary, hotely, nebo také zajímavá místa. Uživatelé poté mohou psát recenze, dávat hodnocení, přidávat fotografie ke konkrétnímu bodu zájmu. Takovéto geosocíální sítě většinou slouží k doporučování podniků, kde působí například firmy Foursquare a Yelp. Mohou také sloužit k doporučování turistických míst, což umožňuje například Trip Advisor.

Geosociální sítě našly také uplatnění v seznamování, kde slouží k vyhledávání lidí, kteří se chtějí seznámit v okolí, tohoto přístupu využívá například Tinder. Takovými sítěmi se v této práci více zabývat nebudu. [7]

1.2 Vytvoření geosociální aplikace

Vytvoření vlastní geosociální sítě v oblasti turismu od základu, by bylo velmi náročné. Pro geosociální síť v oblasti turismu, je kritické množství dat, které uživatelé nahrají na geosociální síť, aby měla uspokojivé a použitelné výsledky. K ukládání takovéhoto množství dat, je potřeba velké úložiště a komplexní databázová struktura.

Většina úspěšných, geosociálních sítí, nabízí veřejné aplikační rozhraní, které zprostředkovává přístup k datům geosociální sítě. Takové to rozhraní je poté možné použít ve vlastní aplikaci.

1.3 Rozdíly v poskytovaných aplikačních rozhraní

V poskytovaných aplikačních geosociálních rozhraní jednotlivých služeb, mohou být velké rozdíly v následujících oblastech.

- **Poskytovaná data** – jednotlivé rozhraní se mohou lišit kvalitou, strukturou i počtem poskytovaných dat. Některé aplikační rozhraní například poskytují pouze jeden obrázek pro POI. Nebo se mohou lišit ve výsledcích pro konkrétní oblast, některé mají více POI, nebo aktuálnější POI.
- **Licenční smlouva** – Služby si stanovují různé podmínky, pro použití jejich aplikačního rozhraní. Například mohou požadovat použití jejich grafických prvků, zakázat použití aplikačního rozhraní v určité oblasti podnikání, nebo vynutit zobrazování bodů zájmu pouze na konkrétních mapách.
- **Architektura aplikačního rozhraní** – Většina aplikačních rozhraní je implementovaná jako REST API, ale je možno se také setkat s GraphQL.
- **Cena** – Služby poskytující aplikační rozhraní se mohou lišit i cenou. Některá API jsou úplně zdarma, některá plně placená. API může být také v základu zdarma, ale pro využívání některých částí, si musí uživatel zaplatit. Takto mohou být zpřístupněny klidně i celé endpointy, nebo pouze počet fotek, či recenzí.

2 PŘEHLED DOSTUPNÝCH APLIKAČNÍCH ROZHRANÍ GEOSOCIÁLNÍCH SÍTÍ

2.1 Geosociální síť Yelp

2.1.1 Základní informace

Yelp je geosociální síť pro hodnocení podniků v mnoha různých odvětví. Vznikl v roce 2004 a od té doby se na síti nastřádalo cca 224 milionů recenzí a připojilo se k ní 30 milionů unikátních zařízení. Yelp působí v 32 zemích po celém světě. [8]

2.1.2 Yelp API

Velkou výhodou Yelpu pro vývojáře je, že nabízí GraphQL API. GraphQL je moderní přístup k aplikačnímu rozhraní, který nechává klienta deklarovat strukturu odpovědi. Tímto se docílí toho, že klient dostane vždy informace, které potřebuje a nemusí při tom volat několik endpointů. Yelp nabízí také klasické REST API, tím se nebudu dále zabývat, jelikož čerpá ze stejné datové základny. Je však nutno zmínit, že REST API vrací 3 fotografie podniku, kdežto GraphQL, vrací pouze jednu fotografii. [9]

2.1.3 Cena a smluvní podmínky

API je poskytováno zdarma. Aplikace musí pro zobrazování hodnocení používat styl hvězdiček Yelpu. Také na každé obrazovce, kde jsou použita data z Yelpu, by měl být odkaz na profil podniku na webové stránce Yelp.com. Aplikace také nesmí uchovávat žádná data stažená z API více než 24 hodin. [10],[11]

2.1.4 Dostupné endpointy

GraphQL má pouze jeden endpoint https://api.yelp.com/v3/graphql. Z tohoto endpointu lze získat všechna potřebná data, tím že specifikujeme v těle GraphQL dotaz.

Objekt Business

Yelp GraphQL dotazy mohou vracet různé objekty, z kterých je poté možné vybrat atributy, které se mají vrátit v odpovědi. Pro aplikaci v oblasti turismu je nejdůležitější objekt *bussines*. Níže popíšu jeho nejdůležitější atributy.

- **name** [String] – Název bodu zájmu.
- **id** [String] – Unikátní identifikátor.
- **alias** [String] – Unikátní přezdívka yelpu pro tento bod zájmu.
- **is_closed** [Boolean] – Určuje zda je bod zájmu permanentně uzavřen.
- **url** [String] – URL bodu zájmu na webu Yelp.com.
- **phone** [String] – Telefonní číslo.
- **display_phone** [String] – Telefonní číslo, formátováno pro zobrazení v aplikaci.
- **review_count** [Int] – Počet uživatelských recenzí.
- **price** [String] – Cenová úroveň podniku. Atributu může nabývat hodnot *$, $$, $$$, $$ $$*. Atribut může také hodnoty *null*, pokud není známa cenová úroveň.
- **distance** [Float] – Vzdálenost od specifikované lokace v požadavku.
- **location** [Location] – Adresa bodu zájmu.
- **coordinates** [Coordinates] – Souřadnice podniku.
- **photos** [array<String>] – List URL adres fotografií podniku.
- **hours** [Hours] – Otevírací hodiny podniku.
- **reviews** [array<Review>] – Uživatelské recenze podniku.

[12]

Dotaz search

Search slouží pro vyhledávání bodů zájmu. Vrací počet bodů zájmu a list bodů zájmu ve vyhledávané lokalitě splňující kritéria specifikovaná v parametrech. Nejdůležitější parametry přijímané tímto dotazem:

- **term** [String] – Volitelný. Výraz pro filtrování vyhledávání dle řetězce. Hodnota může například být *restaurace* nebo *jídlo*. Také to může být přímo název podniku, třeba *CrossCaffe*.
- **location** [String] – Požadovaný, pokud dotaz neobsahuje parametr *longtitude* a *latitude*. Řetězec obsažený v tomto parametru specifikuje geografickou oblast. Například *Praha* nebo *Pardubice*.
- **latitude** [Float] – Požadovaný, pokud dotaz neobsahuje parametr *location*. Zeměpisná šířka lokace, poblíž které mají být body zájmu vyhledávány..
- **longtitude** [Float] – Požadovaný, pokud dotaz neobsahuje parametr *location*. Zeměpisná délka lokace, poblíž které mají být body zájmu vyhledávány.

- **radius** [Int] – Volitelný. Udává doporučený poloměr vyhledávání. Výsledný rádius se může lišit v závislosti na hustotě bodů zájmů v oblasti. Maximální hodnota je 40 000 *m*.
- **limit** [Int] – Volitelný. Počet podniků vrácené ve výsledku dotazu. Výchozí hodnota je *20*. Maximální hodnota je *50*.
- **offset** [Int] – Volitelný. Hodnota specifikující odsazení začátku listu bodů zájmu vráceném v odpovědi. Je používán hlavně pro implementaci stránkování.
- **categories** [String] – Volitelný. Seznam kategorií oddělených čárkou.
- **open_now** [Boolean] – Volitelný. Pokud není specifikován, je nastaven na hodnotu *false*. V případě, že je nastaven na hodnotu *true,* vrací pouze aktuálně otevřené podniky.

[13]

Dotaz business

Dotaz pro získání detailu podniku z unikátního identifikátoru bodu zájmu. Má jediný parametr *id* a vrací objekt *business*.

Další dotazy

Yelp poskytuje dotazy pro vyhledání událostí a získání detailu události. Také lze na Yelpu vyhledávat body zájmu dle telefonního čísla. V neposlední řadě poskytuje dotaz pro získání hodnocení a dotaz pro získání kategorií.

[9]

2.1.5 SWOT analýza

Obrázek 1: SWOT analýza Yelp API. Zdroj: Vlastní.

2.2 Geosociální síť Foursquare

2.2.1 Základní informace

Foursquare poskytuje několik geolokačních služeb, zejména Pilgrim SDK, Pinpoint a pro mou práci nejzajímavější, Places API. Places API se budu dále zabývat blíže. Foursquare také provozuje stejnojmennou aplikaci dostupnou pro Android, iOS i web. Mezi nejzvučnější firmy, využívající služeb Foursquare jsou například Samsung, Uber, Airbnb, ale také i Twitter. Foursquare poskytuje přes 62 milionů POI ve více než 190 zemích. [2]

2.2.2 Cena a smluvní podmínky.

Foursquare nabízí osobní plán který je zdarma, tento plán omezuje maximální počet API dotazů na 99500 u normálních endpointů a 500 u prémiových endpointů. Tento plán může být využíváno pouze pro nekomerční použití. Další plán odstraňuje denní kvóty, ale stojí 599 $ a navíc se platí malá částka za každý dotaz. [3]

2.2.3 Places API endpointy

Venue search
Venue search endpoint vrací seznam objektů *venue* v blízkosti specifikované lokace. Adresa endpointu je *https://api.foursquare.com/v2/venues/*search. Endpoint přijímá několik parametrů, vyjmenuji a popíšu ty nejdůležitější.

- **ll** [String] – Povinný**,** pokud není použit parametr *near*. Zeměpisná šířka a zeměpisná délka lokace uživatele. Například *40.74224-73.99386*
- **near** [String] – Povinný, pokud není použit parametr *ll*. Textový řetězec specifikující název oblasti poblíž, které se má vyhledávat.
- **radius** [Int] – Limituje výsledky vracené tímto endpointem, na maximální vzdálenost od zadané lokality. Maximální možná hodnota je *100 000 m.*
- **query** [String] – Textový řetězec, který bude porovnáván s názvy objektů *venues*.
- **limit** [Int] – Počet výsledků. Maximální hodnota *50*.
- **categoryId** [String] – Seznam unikátních identifikátorů kategorií. Jednotlivé unikátní identifikátory jsou odděleny čárkou.

Endpoint vrací seznam objektů *venue*. Objekt *Venue* se skládá z následujících atributů.

- **id** [String] – Unikátní identifikátor podniku.
- **name** [String] – Název podniku.
- **location** [Location] – Lokace podniku. Objekt obsahující atributy: *adress [String], crossStreet [String], lat [Float], lng [Float], distance [Float], postalCode [String], cc [String], city [String], state[String], country[String], formatedAdress [FormatedAdress].*
- **categories** [Category] – Pole kategorií, do kterých spadá tento podnik. Jedna kategorie bude mít atribut *primary*, takto je označena hlavní kategorie pro tento podnik. Objekt

Category obsahuje attributy: *id [String], name [String], pluralName [String], shotName [String], icon [Icon], primary [Boolean].*

[5]

Detail

• Detail endpoint patří do skupiny prémiových endpointů. Slouží pro získání detailu podniku. Má jediný parametr, kterým je *VENUE_ID* [String]. Vrací objekt *venue,* který kromě atributů, obsažených v objektu *venue* z předchozího endpointu, navíc obsahuje tyto atributy.

- **url** [String] – URL webové stránky podniku.
- **hours** [Hours] – Obsahuje objekt *Hours*, kdy je podnik otevřen v lidsky čitelném formátu.
- **popular** [Hours] – Obsahuje objekt *Hours*, v kterém lidi podnik nejčastěji navštěvují**.**
- **price** [Price] – Objekt obsahující hodnotu cenové hladinu a zprávu popisující cenovou hladinu.
- **rating** [Int] – Hodnocení podniku. Hodnota od 0 do 10.
- **description** [String] – Popis podniku, poskytnutý manažerem podniku.
- **hereNow** [Int] – Počet lidí nacházející se na tomto místě.
- **photos** [Object] – Vrací počet obrázků obsažených v odpovědi a pole objektů *photo*.
- **bestPhoto** [String] – Fotografie, kterou Foursquare zhodnotil jako nejlepší pro tento podnik na základě uživatelských hodnocení a interního algoritmu.

Další endpointy

Foursquare poskytuje spoustu dalších endpointů. Je zde endpoint pro získání doporučených podniků, fotografie, otevírací hodiny, podniky, které jsou teď v trendu, podobné podniky a spoustu dalších.

[5],[1]

2.2.4 SWOT analýza

Obrázek 2: SWOT analýza Foursquare API. Zdroj: Vlastní.

2.3 Geosociální aplikační rozhraní Google Places

2.3.1 Cena

Places API je placená stylem "pay-as-you-go", to znamená, že se platí podle toho, kolik aplikace udělá požadavků na API. Z této ceny poté Google strhne 200$, takže pokud se cena vejde do této částky, API je zdarma. Pole vracená API jsou rozdělena na 3 skupiny, kde každá skupina má jinou cenu.

- **Základní data** – Zdarma**.** Do této kategorie například spadají pole *business_status, formatted_address, geometry, icon,name, permanently_closed (deprecated), photos, place_id, plus_code, types.*
- **Kontaktní údaje** – Pokud počet dotazů nepřesáhne 100 000 cena je 3 $ na 1000 dotazů. Od 100 001 do 500 000 je cena 2.4 $ na 1000 dotazů. Do této kategorie spadají pole *formatted_phone_number, international_phone_number, opening_hours, website*.
- **Data o atmosféře v podniku –** Pokud počet dotazů nepřesáhne 100 000 cena je 5 $ na 1000 dotazů. Od 100 001 do 500 000 je cena 4 $ na 1000 dotazů. Do této kategorie spadají pole *price_level, rating, review, user_ratings_total*

[25]

2.3.2 Smluvní podmínky

Pokud je API používána, pro zobrazování informací o bodech zájmu na mapovém podkladu, jako mapový podklad musí být použity Google maps. Pokud jsou informace o bodech zájmu zobrazovány jinak než na mapovém podkladu, musí zde být logo „powered by Google“. [24]

2.3.3 Google Places API endpointy

Endpointy Google Places API jsou zajímavé tím, že je možné specifikovat, jaká všechna pole mají vracet. Lze také specifikovat v jakém formátu budou data vracena, to se dá udělat URL parametrem *vstup,* který může nabývat hodnot *json* pro výstup ve formátu JSON nebo *xml* pro výstup ve formátu XML. Endpointy přijímají parametry, ty budu označovány v URL jako *parametry.*

Společné parametry endpointů

Endpointy mají několik společných parametrů,

- **key** – Povinný. Google places API klíč. Tento klíč lze získat stejně jako Google maps API klíč. Jak získat Google maps klíč, popisují dále v bakalářské práci.
- **fields –** Volitelný. Tento parametr specifikuje, jaká všechna pole má server vrátit. Všechna pole, která může endpoint vrátit, je vypsán v podkapitole cena. Cena se poté počítá dle nejdražšího zvoleného pole.
- **language –** Volitelný. Kód jazyku, ve kterém by měly být výsledky dotazu.

Find Places

Endpoint find places přijímá textový vstup a vrací list objektů *place*. Vstup můžou být jakákoli textová data objektu *place*, například jméno, adresa nebo telefonní číslo. Tento endpoint má pouze omezený výčet polí, která vrací, chybí zde například *reviews* a další. Dotaz find place je dostupný přes následující HTTP URL.

https://maps.googleapis.com/maps/api/place/findplacefromtext/*vystup?parametry*

Následující vstupní parametry lze použít.

- **input** – Povinný. Textový vstup, dle kterého se bude hledat.
- **inputtype** – Povinný. Typ vstupu. Specifikuje, zda vstup je text nebo telefonní číslo. Musí být jedna z hodnot *textquerry* nebo *phonenumber.*
- **locationbias** – Volitelný. Preferování výsledků ve specifikované lokaci. Specifikovat lokaci lze následujícími způsoby.
  - Dle lokace IP – Bere lokaci dle lokace IP adresy. Lze specifikovat hodnout *ipbias*. Defaultní hodnota, pokud není parametr *locationbias* specifikován.

- Bod – Bod, který je specifikován zeměpisnou délkou a šířkou. Zadává se ve formátu *point:lat,lng.*
- Okruh – Specifikuje okruh poloměrem, zeměpisnou šířkou a délkou. Zadává se ve formátu *circle:radius@lat,lng.*
- Obdélníková oblast – Obdélníková oblast se specifikuje textovým řetězcem, který obsahuje 2 páry zeměpisné šířky a délky. První pár reprezentuje jih a západ, druhý sever a východ. Formát je následující *rectangle:south,west|north,east.*

[37]

Nearby search

Endpoint, který vrací výsledky v okolí specifikované lokace. Nachází se na následující HTTP adrese. *https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters*
Pro endpoint lze použít tyto parametry.

- **location** – Povinný. Lokace, kolem které bude vyhledáváno.
- **radius** – Povinný. Poloměr.
- **keyword** – Volitelný. Klíčové slovo sloužící k filtrování výsledků.
- ***minprice,maxprice*** – Volitelný. Omezuje výsledky dle cenové kategorie na takové, které spadají do tohoto rozsahu. Rozsah může být od 0 do 4, kde 4 je nejdražší.
- **rankby** – Specifikuje pořadí, v jakém budou výsledky vráceny. Možné hodnoty jsou *prominence*, *distance* a *location.*
- **opennow** – Vrací pouze zrovna otevřená místa.

[37]

Place details

Pokud je již známé id objektu *place,* tímto endpointem se lze doptat na více detailů. Tento endpoind vrací všechny pole objektu *Place.* Dotaz Place details je dostupný přes HTTP adresu *https://maps.googleapis.com/maps/api/place/details/vystup?parametry.*

Lze použít tyto parametry.

- **place_id** – Povinný. Unikátní identifikátor objektu *place.*
- **region –** Kód regionu ve lokace výsledku. Parametr neslouží jako úplná restrikce, pokud je lepší výsledek mimo region, bude vrácen tento lepší výsledek. Specifikuje se textovým řetězcem nejvyšší domény země.

[38]

Další endpointy

Google places API poskytuje ještě 2 další endpointy. Place autocomplete, ten slouží k automatickému našeptávání při vyplňování vstupu pro hledání místa. Dalším je place photos, ten slouží k doptání na lepší fotografie ke konkrétnímu objektu Place. [36]

2.3.4 SWOT analýza API

Obrázek 3: SWOT analýza Google Places API. Zdroj: Vlastní.

3 TEORETICKÉ ZÁKLADY FRAMEWORKU FLUTTER

3.1 Úvod

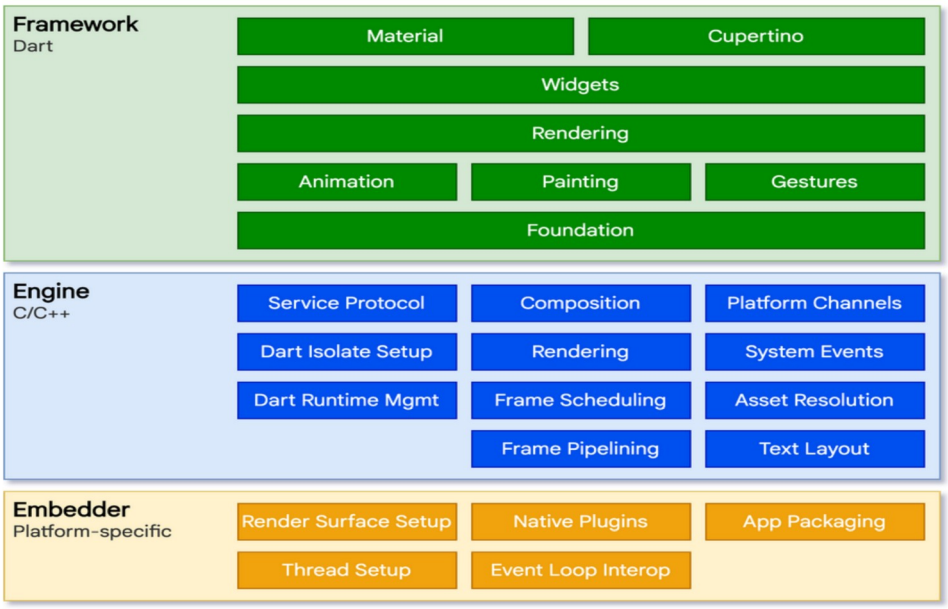
Flutter je open-source multiplatformní framework pro vývoj aplikací. Flutter 1.0 vyšel v roce 2018 a umožňoval pouze vývoj pro Android a iOS. [15] S příchodem Flutter 2.0 v roce 2021, Flutter přidal do stabilní verze i podporu webové platformy. Dále byla vydána do beta verze podpora vývoje aplikací pro Windows a Linux. [16] Flutter je dále používán operačním systémem Fuchsia pro tvorbu uživatelského rozhraní. [17]

3.2 Programovací jazyk Dart

Dart je programovacím jazykem ve kterým se píšou aplikace ve frameworku Flutter. Byl vytvořen s tímto záměrem, z tohoto důvodu se při návrhu kladl důraz na to, aby byl dobrý pro psaní klientských aplikací. Jeho syntaxe je podobná jazykům jako C nebo Java. Dart je staticky typovaným jazykem, ale umožňuje i používání dynamických proměnných použitím typu *dynamic*. Od verze 2.12 používá null safety, to znamená, že proměnná nemůže nabývat hodnoty *null*, pokud to není explicitně definováno. Podporuje reaktivní a asynchronní programování. [20]

3.3 Architektura frameworku

Flutter je navržen jako rozšířitelný vrstvený systém. Každá vrstva se skládá z nezávislých knihoven. Žádná vrstva nemá privilegovaný přístup k vrstvě pod ní.



Obrázek 4: Vrstvy Flutteru. Zdroj [18]

3.3.1 Embedder

Embedder slouží, jako přístupový bod k platformě, na které Flutter běží. Slouží ke koordinaci s operačním systémem zařízení a pro přístup ke službám operačního systému. Embedder je pro každou platformu jiný a je psán v jazyku nativním pro danou platformu. Pro Android je

napsán v jazyce Java a C++. Pro iOS a MacOS je psán v jazyce Objective-C a Objective-C++. Pro Windows a Linux v jazyce C++. Použitím embedderu, kód Flutteru může být přímo integrován do existující aplikace jako modul, ale většinou je kód Flutteru obsahem celé aplikace. Flutter zahrnuje několik embedderů pro běžně používané platformy, ale existují i embeddery vytvářeny nezávisle na Flutter týmu.

3.3.2 Engine

Vrstva engine, je jádrem Flutteru. Je z největší části napsána v jazyce C++. Slouží k rasterizaci a skládání scén. Poskytuje nízko-úrovňovou implementaci hlavního aplikačního rozhraní Flutteru. Toto zahrnuje grafiku, textový rozložení, práci se soubory a sítí, architekturu pro pluginy. Engine je zpřístupněn Flutter frameworku přes knihovnu dart:*ui*, která zabaluje třídy jazyka C++ do tříd jazyka Dart.

3.3.3 Framework

Při tvoření aplikace, programátor většinu času interaguje s touto vrstvou, která poskytuje moderní, reaktivní framework napsaný v jazyce Dart. Obsahuje komponenty pro specifické platformy, komponenty pro rozložení a základní knihovny. Flutter framework se skládá z následujících vrstev.

- **Foundation** – Zde se nachází základní třídy jako animace, barvení a gesta.
- **Renderovací vrstva** – Vytváří abstrakci pro vytváření rozložení. S touto vrstvou lze sestavit strom rendrovatelných objektů. S těmito objekty lze dynamicky manipulovat. Při manipulaci s objekty, se strom automaticky obnovuje a rozložení reflektuje provedené změny.
- **Widgetová vrstva** – Tato vrstva je abstrakcí pro kompozici. Každý rendrovatelný objekt v rendrovací vrstvě má korespondující objekt ve widgetové vrstvě.
- **Material a Cupertino knihovny** – Tyto knihovny poskytují základní prvky pro jednoduchou implementaci Material a iOS Cupertino designu.

Flutter framework je celkem malý. Většina vysoko-úrovňových funkcionalit je zpřístupněna formou balíčků. Sem patří funkce jako vibrace, kamera, webview, http a mnoho dalších. O tyto základní balíčky se stará Flutter tým.

[18]

3.4 Kompilace jazyka Dart

Kompilátor jazyka Dart umožňuje spouštět kód různými způsoby. Na mobilních zařízeních a osobních počítačích jazyk Dart používá virtuální stroj s just-in-time kompilací a také ahead-of-time kompilací, pro vytváření nativního strojového kódu. Na webu Dart používat kompilátor pro vývoj dartdevc a produkční kompilátor dart2js. Oba překládají jazyk Dart do jazyku JavaScript.

Při vývoji je používán just-in-time kompilátor. Tento kompilátor umožňuje rychlé iterační cykly a tím urychluje i samotný vývoj aplikací. Je mnohem pomalejší, ale nabízí výhody v používání specifických nástrojů jako hot reload, hot restart nebo DevTools.

- **Hot reload** – Funguje tak, že změněný kód je vložen do běžícího virtuálního stroje. Virtuální stroj aktualizuje třídy a Flutter framework automaticky znovu sestaví aktuální strom widgetů. Tato změna trvá v rámci vteřin a tím napomáhá rychle a jednoduše experimentovat, stavět UI, odstraňovat chyby a přidávat nové funkce.
- **Hot restart** – Ve většině případů stačí hot reload. Ve speciálních případech je potřeba hot restart. Ten provede to samé, co hot reload a navíc restartuje aplikaci, čímž obnoví její stav.
- **DevTools** – DevTools je sada ladících a debugovacích nástrojů pro Dart a Flutter. S DevTools se dá procházet rozložení uživatelského rozhraní a aktuální stav aplikace, diagnostikovat záseky v uživatelském rozhraní, ladit výkon procesoru, sledovat využívání sítě, analyzovat kód a velikost aplikace.

[22]

3.5 Asynchronní programovaní

Asynchronní programování je realizováno asynchronními funkcemi a třídami *Future, Stream.* Asynchronní programování je vhodné, používat pro případy, kdy čekáme na nějaká data zvenčí, nebo než se provede náročná funkce. Po příklad bych zde uvedl volání API, kde aplikace musí čekat, než server odpoví a až poté je znám výsledek. Dalším příkladem, nechť je čekání na uložení dat do lokální databáze.

Asynchronní programování také umožňuje reaktivní programování, kde například widget poslouchá na události datového toku a pokaždé když se objeví nová událost, widget je překreslen s hodnotami obsaženými v datovém toku.

3.5.1 Asynchronní funkce

Asynchronní funkce je označena klíčovým slovem *async* a návratovým typem *Future*. V asynchronní funkci, mohou být volány další asynchronní funkce. Pokud není žádoucí návratová hodnota *Future,* je možné, použít klíčové slovo *await*, funkce potom počká na místě, dokud *Future* nemá výsledek.

3.5.2 Třída Future

Třída *Future* reprezentuje nějaký běžící proces, který ještě není dokončen a jeho hodnota bude známa někdy v budoucnu. *Future* může skončit buď s hodnotou nebo s errorem. Přijímatel *Future* může registrovat funkce, které určují, co se stane v momentě, kdy je dostupná hodnota nebo error. *Future* může mít zaregistrováno více funkcí, které se provolají při objevení hodnoty. Pokud tomu tak je, každá funkce je vyhodnocena nezávisle na sobě.

3.5.3 Datové proudy

Dalším prvkem asynchronního programování v jazyce Dart jsou datové proudy. Datový proud je sekvence asynchronních událostí a poskytuje způsob pro práci s touto sekvencí. Každá událost, která se v datovém proudu objeví, je buď konkrétní hodnota nebo error. Pokud datový proud odeslal všechny události, pošle poslední událost typu *done*, která notifikuje přijímatele o konci datového proudu.

Na události v datových proudech lze poslouchat funkcí *listen*. Funkce *listen* vrací objekt *StremSubscription*, což je objekt, který zprostředkovává události. Tento objekt může být i použit pro zastavení odposlechu, nebo k dočasnému pozastavení přijímání událostí.

Datové proudy je možno transformovat a tím vytvořit nový datový proud. Například je možné jeho výsledky transformovat na jinou třídu, vytvořit filtrovaný datový proud, nebo přeskočit určitý počet událostí.

Existují 2 typy datových proudů.

- **Single-subscritption** – Umožňuje mít pouze jednoho odběratele. Neodesílá události, pokud nemá žádného odběratele a přestane odesílat události, pokud odběratel přestane poslouchat.
- **Broadcast** – Umožňuje mít více odběratelů. Odesílá události hned co jsou připraveny nezávisle na tom, zda někdo poslouchá. Odběratel se může připojit v jakémkoli bodě životního cyklu datového proudu. Odběratel, který se připojí, získá až další událost, kterou datový proud odešle. Pokud datový proud už nemá žádné události, odešle událost typu *done* a odpojí všechny odběratele. Ty tedy tuto událost už nedostanou. Pro vytvoření datového proudu typu *broadcast*, je třeba zavolat na instanci třídy *Stream* funkci *asBroadcast()*.

[20],[21],[22]

3.6 Widgety

Widgety jsou základními stavebními kameny při vytváření uživatelského rozhraní ve frameworku Flutter. Widget je reprezentovaný neměnnou třídou. Tvoření uživatelského rozhraní je založeno na principu kompozice. Widgety mohou reprezentovat strukturální prvky, stylistické prvky (Fonty, barevná schémata), prvky pro rozložení (sloupce, řady, odsazení), reaktivní prvky a další věci. Tyto jednotlivé widgety se poté zanořují do sebe a skládá se z nich strom widgetů. Každý widget je zanořen do svého předka a může přijímá jeho *BuildContext*. Taková to struktura pokračuje, až úplně nahoru ke kořenovému widgetu, který je typicky *MaterialApp.*

Pro vytvoření vizuální reprezentace widgetu, je potřeba přepsat funkci *bulid()*. V této funkci se většinou nacházejí další widgety. Flutter framework poté rekurzivně volá tuto funkci, aby získal vizuální reprezentaci celého stromu.

Ve Flutter se nacházejí 2 hlavní třídy widgetů. *StatefullWidget* a *StatelessWidget*. Většina widgetů je neměnná a jejich stav se nemění, například ikona nebo text. Pokud chci vytvořit takovýcto widget, musí dědit ze třídy *StatelessWidget*.

Pokud widget musí měnit svůj stav, například na základě uživatelského vstupu, takovýto widget je stavový. Stavový widget se vytváří děděním ze třídy *StatefullWidget*. Protože jsou widgety neměnné, *StatefullWidget* drží svůj stav v jiné třídě, která dědí z třídy *State*. Stavové widgety nemají metodu *build()*, místo toho jejich vizuální reprezentace je vytvářena metodou

build() objektu *State*. Pokud je třeba změnit stav widgetu, stačí zavolat metodu *setState()* na objektu *State*, ta notifikuje Flutter framework, aby znovu vytvořil vizuální reprezentaci tohoto widgetu.

Díky tomu, že stavové widgety mají svůj stav v jiné třídě, ostatní widgety se k nim chovají úplně stejně jako k bezstavovým widgetům, aniž by se staraly o to, jak udržet jejich stav. [18]

3.7 Mixiny

Dart umožňuje používání mixinů, pro větší možnost recyklace kódu. Mixin je třída, která nemá žádný konstruktor. Mixin se používá podobně jako dědičnost, akorát místo klíčového slova *extends* používá klíčové slovo *with*. Pokud není žádoucí, aby bylo možno z mixinu vytvořit instanci, je potřeba při deklaraci mixinu, použít klíčové slovo *mixin* místo klíčového slova *class*. Mixiny lze také omezit, aby byly použity jen na určité třídě. To se dělá tak, že se za název třídy, přidá klíčové slovo *on* a název třídy, na kterou to chceme omezit. [21]

```
class Musician {
  // ...
}
mixin MusicalPerformer on Musician {
  // ...
}
class SingerDancer extends Musician with MusicalPerformer {
  // ...
}
```

Obrázek 5: Vytvoření mixinu v jazyce Dart. Zdroj [21].

4 DOTAZOVACÍ JAZYK GRAPHQL

4.1 Historie

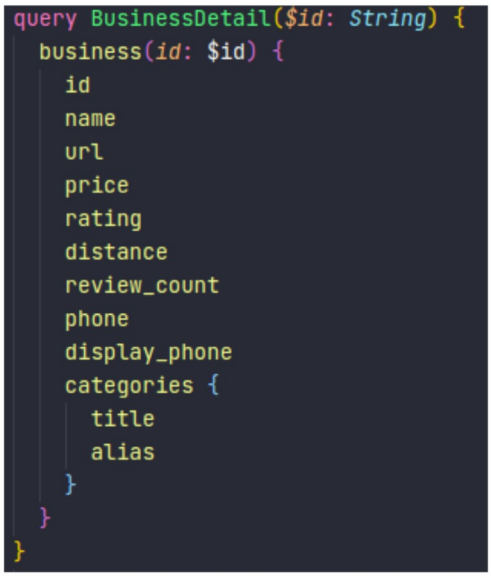
GraphQL bylo vytvořeno firmou Facebook v roce 2012. Facebook potřeboval lepší přístup k datům, který by mohl používat ve svých produktech a službách. Také chtěl vytvořit systém srozumitelný pro vývojáře, designery a zároveň netechnické uživatele. Facebook GraphQL nejdříve používal interně a v roce 2015, ho umožnil používat veřejnosti.

4.2 Základní informace

GraphQL je dotazovací jazyk. Slouží pouze pro servírování dat klientovi. Nezajímá se o to odkud data pocházejí. Data mohou pocházet z databáze, mikroslužby, ale také z REST API. GraphQL většinou běží na protokolu HTTP. Operace jsou pouhé textové řetězce, takže není potřeba žádného speciálního nástroje pro dotazovaní na GraphQL.

4.3 Princip fungování

V GraphQL se pevně definuje schéma. Schéma obsahuje přesný popis dat, která GraphQL může vrátit. Musí být definovány i typy dat. Jsou 2 typy operací, které lze v GraphQL vytvářet. *Mutace* a *query*. *Query* Slouží k získávání dat z API, jsou obdobou pro volání typu GET na REST API. Při volání operace typu query na GraphQL, klient definuje, jaká data chce vrátit z definovaného schématu. Mutace na druhé straně, slouží k vytvoření změny v databázi, obdobně jako volání typu POST a DELETE na REST API.



Obrázek 6: GraphQL dotaz. Zdroj: Vlastní.

4.4 Výhody GraphQL

- Klient nezískává zbytečná data navíc.
- Klient nemusí volat více endpointů, pro získání entity. Například se často stává, že klient volá endpoint na REST API, který vrací množinu entit, akorát entitě na tomto end-

pointu chybí jeden atribut, který klient taky potřebuje. Musí potom navíc provolat endpoint pro každou entitu, aby získal tento atribut.

- Lze jednoduše doplňovat atributy do stávajícího schématu.
- Tím že je schéma objektové, lze jednoduše ukládat objekty v lokální databázi zařízení.

[26],[27]

5 NÁVRH A VÝVOJ MOBILNÍ APLIKACE

5.1 Požadavky na aplikaci

5.1.1 Funkční požadavky

- Aplikace bude zobrazovat dostupné body zájmu v okolí na mapovém podkladu.
- Aplikace umožní zobrazit detail bodu zájmu.
- Aplikace umožní filtraci bodů zájmů dle kategorií, místa, vzdálenosti.
- Aplikace bude využívat API geolokační sítě.
- Aplikace bude pro operační systémy iOS a Android.

5.1.2 Nefunkční požadavky

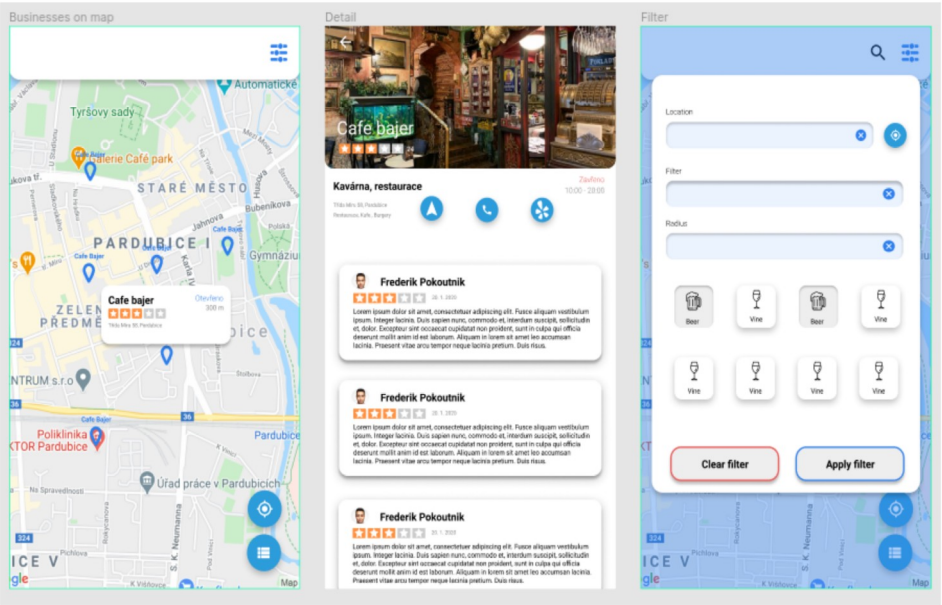
- Aplikace bude uživatelsky přívětivá.
- Aplikace bude spolehlivá.

5.2 Popis aplikace

Aplikace je realizována ve frameworku Flutter. Aplikace má tři obrazovky. Hlavní obrazovkou je mapa, na které budou zobrazovány body zájmu. Pro práci s mapou, jsem vybral Google maps a data pro body zájmu budou získávány z Yelp GraphQL API. Další obrazovkou bude filtr bodů zájmu. Ve filtru si uživatel může zvolit kritéria, dle kterých chce body zájmu filtrovat. Při uložení filtru mu budou body zobrazeny na mapě. Filtr je uložen v lokální databázi. Při vypnutí a zapnutí aplikace, hodnoty zadané ve filtru jsou obnoveny. Po kliku na bod zájmu na mapě, se uživateli zobrazí malý informační dialog. Po kliku na dialog se zobrazí obrazovka detailu. Na obrazovce detailu bude zobrazen detail bodu zájmu s detailními informacemi o bodu zájmu. Načítání obrazovky detailu bude realizováno takzvaným “shimmer“ efektem.

5.3 Návrh grafického designu

Při návrhu aplikace je vhodné nejdříve vytvořit grafický design. Tento krok není nutný, ale pomůže k získání představy o vzhledu výsledné aplikace, rozložení obrazovek, umístění ovládacích prvků a způsobech zobrazení dat uživateli. Díky grafickému designu je i jednodušší, si představit a navrhnou infrastrukturu aplikace, ještě před začátkem samotného vývoje. Grafický design byl realizován v nástroji Figma.



Obrázek 7: Původní grafický design. Zdroj: Vlastní.

5.4 Úvod do balíčku BloC

5.4.1 Základní informace

BloC je je návrhový vzor, který slouží k oddělení prezentační vrstvy od logiky aplikace. Tento návrhový vzor lze ve frameworku Flutter jednoduše implementovat s použitím balíčku bloc a flutter_bloc. Balíček bloc se snaží, aby změny stavu byly předvídatelné tak, že reguluje, kdy se změna stavu může objevit. Napomáhá tím také ke konzistenci řešení stavu v aplikaci tím, že změny stavu jsou řešeny vždy stejným způsobem. Bloc je navržen tak, aby splňoval následující požadavky.

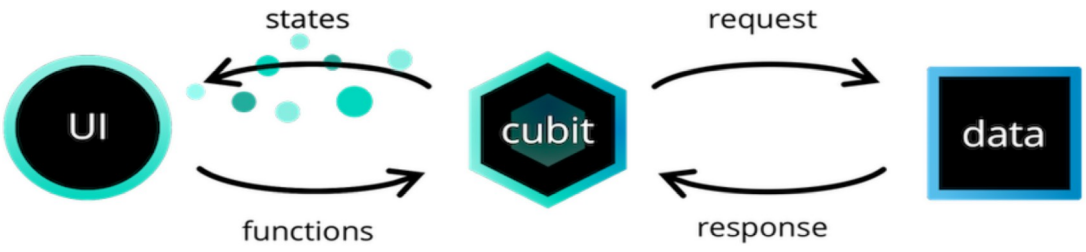
- Možnost zjistit v jakém stavu se aplikace nachází v jakémkoli časovém bodě.

- Jednoduše otestovat všechny možné případy, které mohou nastat, pro zajištění maximální spolehlivosti aplikace.
- Možnost zaznamenat každou uživatelskou interakci.
- Maximalizace efektivity a možnosti opětovného použití komponentu.
- Bezproblémový souběh více vývojářů, používáním stejných konvencí a vzorů.
- Vytvářet rychlé a responsní aplikace.

5.4.2 Princip fungování

Balíček bloc poskytuje 2 třídy pro implementaci návrhového vzoru BloC. *Cubit a Bloc*. Třída *Cubit* je jednodušší na používání a je vhodná pro případy, kdy se stav aplikace mění přímo v reakci na vstup uživatele. Třída *Bloc* nabízí širší možnosti práce s příchozími událostmi. Obě třídy vycházejí z třídy *BlocBase* a jsou dost podobné tím, jak fungují. Třídu *Bloc* v aplikaci není používána, proto se nadále budu věnovat pouze třídě *Cubit*.

Cubit poskytuje datový proud stavů, do kterého se na základě volaných veřejných funkcí cubitu, přidávají události. Aktuální stav *cubitu* se dá zjistit z getteru *state*. Odeslání nového stavu do datového proudu, se dělá funkcí *emit(State novyStav)*. Uvnitř potomka třídy *Cubit* je možné, sledovat změny stavu přepsáním metody *onChange()*, která poskytuje předchozí a následující stav. Při změně stavu, se zavolá tato metoda a až poté, se v datovém proudu objeví nový stav.



Obrázek 8: Interakce UI s Cubitem. Zdroj [23].

5.4.3 Poskytované widgety

Balíček flutter_bloc poskytuje, widgety a funkce pro práci s Cubitem.

- **BlocProvider** – Třída *BlockProvider* slouží pro poskytnutí instance potomka třídy *Cubit*. Instance, je posílána dolů stromem widgetů pomocí třídy *BuildContext*. *BlocPro-*

vider se umístí na vrch stromu. Parametrem *create* se specifikuje instance potomka třídy *Cubit,* kterou *BlockProvider* bude poskytovat. Takto poskytovaný potomek třídy *Cubit,* lze zpřístupnit v jakémkoli potomku tohoto widgetu pomocí rozšiřující funkce na třídě *BuildContext context.read<Cubit>().*

- **BlocBuilder<Cubit, State>** – *BlocBuilder* slouží k znovu sestavení widgetu, pokaždé co se objevý nový stav. Parametrem *builder* se pomocí funkce s návratovou hodnotou *Widget* specifikuje, jak má být widget při změně stavu sestaven. Pokud není žádoucí, aby se widget znovu sestavoval při každé změně stavu, lze specifikovat parametrem *buildWhen*, v jakých případech se má znovu sestavit. *BuildWhen* přijímá funkci s parametry *State oldState, State newState* s návratovou hodnotou *boolean*.
- **BlocListener<Cubit, State>** – Třída *BlocListener* slouží k vytváření vedlejších efektů při změně stavu. Parametrem *listener* se specifikuje funkce, která je provolána, pokud se změnil stav potomka třídy *Cubit*. Pokud není žádoucí, aby se funkce provolávala pokaždé, co se změní stav, je možné toto chování specifikovat parametrem *listenWhen*.
- **MulitBlocProvider** – V aplikaci je většinou více potomků třídy *Cubit,* které je potřeba poskytovat. Několik zanořených instancí třídy *BlocProvider* vypadá ošklivě a je nepřehledné. Z tohoto důvodu byl vytvořen widget *MultiBlocProvider*. Parametrem *providers* je specifikována množina instancí třídy *BlocProvider* a funguje to potom stejně, jako kdyby byly tyto instance vnořeny do sebe.
- **MultiBlocListener** – Stejné použití jako u třídy *MultiBlocProvider* s tím, že třída *MultiBlocListener* je pro množinu instancí třídy *BlocListener.*

Tyto widgety stačí pro většinu případů. Existují však speciální případy, které nelze hezky vyřešit s použitím těchto widgetů. Nechť je příkladem widget, který se sestavuje na základě několika potomků třídy *Cubit*. V takovémto případě, by muselo být zanořeno více instancí třídy *BlocBuilder* do sebe, což vypadá ošklivě a není to moc přehledné. Z tohoto důvodu balíček bloc poskytuje 2 rozšiřující funkce na třídě BuildContext. Tyto funkce se používají uvnitř funkce *builder* ve widgetu *Builder*.

- **Funkce watch<Cubit>().state** – Funkce *watch* naslouchá na změny stavu potomka třády *Cubit,* pokud se tak stane, reprezentace widgetu je překreslena s novým stavem.

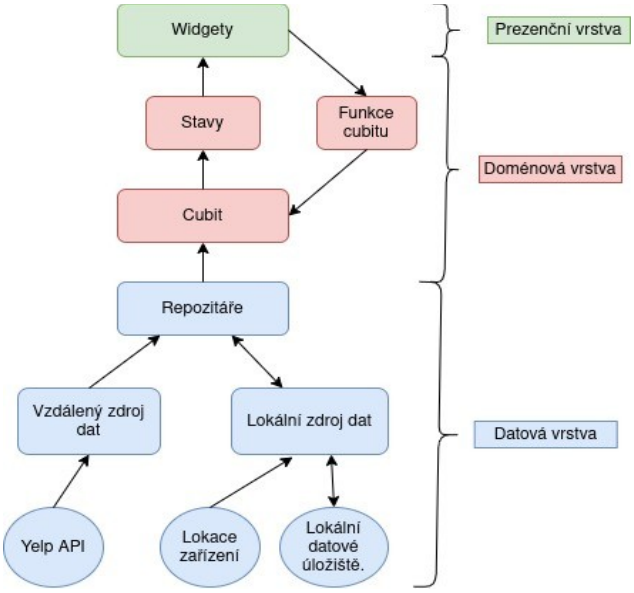
Ve funkci *builder* může být těchto funkcí použito více, stav je poté překreslen pokaždé kdy se změní stav potomka třídy *Cubit* v jakékoli funkci.

- **Funkce select<Cubit>(Unit Function(Cubit))** – Funkce *select* sleduje změnu konkrétní hodnoty stavu. Funguje stejně jako *BlocBuilder* se specifikovaným parametrem *buildWhen*.

[23]

5.5 Architektura aplikace

Aplikace je rozdělená do 3 vrstev. Každá vrstva je pro jiný typ tříd. Vrstvy si své služby navzájem zpřístupňují přes funkce veřejného rozhraní. Vrstva vždy může komunikovat pouze s nadřazenou, nebo podřízenou vrstvou.



Obrázek 9: Diagram komunikace mezi vrstvami. Zdroj: Vlastní.

5.5.1 Prezenční vrstva

V prezenční vrstvě se nacházení všechny widgety, styly, barvy. V prezenční vrstvě by neměla být žádná logika. Vstupy uživatele jsou předávány funkcemi potomkům třídy *Cubit*. Widgety pouze reagují na stavy potomků třídy *Cubit*, které přicházejí z doménové vrstvy.

5.5.2 Doménová vrstva

V doménové vrstvě je řešena veškerá logika aplikace. V aplikaci to je místo, kde se nacházejí všichni potomci třídy *Cubit* a jejich stavy. Doménová vrstva slouží jako prostředník mezi prezenční a datovou vrstvou. Typicky pokud uživatel vloží nějaký vstup, provolá se funkce potomka třídy *Cubit*, který zavolá nějakou datovou službu, počká na výsledek a odešle ho jako nový stav. Prezenční vrstva poté reaguje na tento nový stav, změnou v uživatelském rozhraní.

5.5.3 Datová vrstva

Datová vrstva poskytuje zdroje dat, přes rozhraní doménové vrstvě. V datové vrstvě jsou entity, repositáře a zdroje dat. Pro přístup k datové vrstvě jsou používaný repositáře. Repositář je třída zaobalující a slučující volání konkrétních zdrojů dat. Taky zde jsou odchytávány vyjímky a chybové stavy, které jsou transformovány na potomky třídy *Failure* a předány dále v levé straně monády *Either*. Repositáře jsou zprostředkovávány, přes jejich abstraktního předka. Každý repositář má abstraktního předka pro definici funkcí a implementační třídu. To zaručuje větší flexibilitu při změně zdroje dat. Pokud by byl změněn poskytovatel geosociálního API, změny by se týkaly pouze této vrstvy a ostatní vrstvy by to nemělo nijak ovlivnit.

5.6 Uživatelské rozhraní

5.6.1 Animace

Flutter nabízí 2 typy animací. Explicitní animace a implicitní animace.

- **Implicitní animace** – Ve frameworku Flutter je možné využívat implicitně animované widgety. Tyto widgety se samy při znovu sestavení animují mezi předchozím a následujícím stavem. Widgetu stačí zadat délku animace a křivku pro interpolaci animační hodnoty. Implicitní widgety jsou vhodné pro jednoduché, jednoúčelové animace, jako změna velikosti, změna barvy nebo změna pozice widgetu. V aplikaci jsou použity implicitně animované widgety pro změnu pozice informačního dialogu, nebo pro přechod mezi stavem načítání a obrazovkou s načtenými daty. 51

- **Explicitní animace** – Explicitně animované widgety jsou vhodné pro komplexní animace, jelikož poskytují plnou kontrolu nad průběhem animace. K vytvoření explicitní animace je potřeba několik prvků, které zde budou popsány.
  - **Statefull widget** – Animovaný widget musí být stavový, aby se mohl měnit jeho stav.
  - **TickerProviderStateMixin** – Třída držící stav widgetu musí implementovat *TickerProviderStateMixin*. Ten poskytuje tikání, podle kterého se widget přestavuje.
  - **AnimationController** – Animační kontrolér řídí animaci. V konstruktoru animačního kontrolér se nastaví délka animace. Animace se spouští funkcí *forward().* Funkce *reverse()* spustí animaci pozpátku. Animačnímu kontroléru je možnost přidat funkci provolávanou při změně stavu animace, nebo při změně hodnoty animace. Animační kontrolér spouští všechny animace, kterým je předkem.
  - **Tween** – Třída *Tween* je datová struktura držící počáteční a koncovou hodnotu animace.
  - **Animation** – *Animation* je abstraktní třídou. Pro vytvoření animace interpolované mezi hodnotami instance třídy *Tween,* je nutné zavolat funkci *animation(Animation animation)* na instanci třídy *Tween*. V aplikaci používám implementátora třídy *Animation CurvedAnimation*, tento implementátor slouží k vytvoření animace po specifikované křivce. Parametrem *parent* je mu předán animační kontrolér. Parametrem *curve* je specifikován interval začátku a konce a interpolační křivka. Interval je hodnota od 0 do 1 udávající, v jakém okamžiku animace začne a skončí.

[34]

5.6.2 Načítání dat

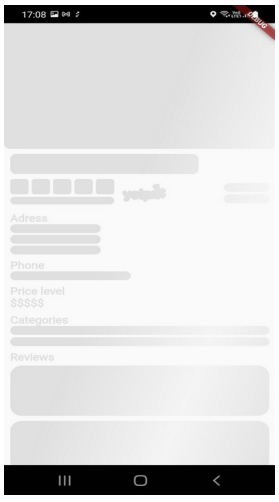
Pokud aplikace načítá data, uživatel by o tom měl být vhodně informován, aby si například nemyslel, že se aplikace zasekla a nesnažil se jí restartovat. V aplikaci jsou použitý 2 různé způsoby.

- **Indikátor načítání** – Indikátor načítání je nejednodušším způsobem, jak uživatele informovat o průběhu načítání dat. Ve Flutteru se dá zobrazit použitím widgetu *CircularProgressIndicator* nebo *ProgressIndicator*. Hodnotu načítání je možno, buď pří-

mo ovládat parametrem *value,* nebo parametr *value* nespecifikovat a indikátor bude animaci pouštět v nekonečné smyčce.

- **Efekt shimmer** – Termín shimmer efekt je používán pro obrazovku nebo komponentu, která má jednotnou barvu a přejíždí přes ní gradient jiné barvy. Vetšinou se používají odstíny šedé barvy a celkový efekt působí jako kov lesknoucí se na slunci. Tento efekt se dá použít buď pro celou obrazovku, či množinu komponent nebo pouze pro jednu komponentu. Efekt je vhodné použít pro načítání obrázků nebo obrazovky, kde je známa přibližná struktura výsledného rozložení po načtení. V mé aplikace se použití tohoto efektu výborně hodilo pro zobrazená načítání detailu bodu zájmu.

Pro implementaci efektu ve frameworku Flutter je možné využít balíček shimmer. Tento balíček obsahuje widget *Shimmer*, který vykresluje efekt na všechny jeho potomky s pozadím. Takže stačí vytvořit kopii obrazovky, na které má být efekt použit a všechny dynamické prvky jsou nahrazeny kontejnery. Poté stačí celou obrazovku zabalit do widgetu *Shimmer*. Pro výměnu této obrazovky za obrazovku s načtenými daty je vhodné použít widget *AnimatedSwitcher*, který zajistí plynulý přechod mezi widgety.



Obrázek 10: Efekt shimmer. Zdroj: Vlastní.

5.6.3 Chybové stavy

Uživatel je o chybových stavech informován pomocí takzvaných snackbarů. Snackbar je malé okénko, které na chvilku vyjede odspodu obrazovky a slouží k zobrazení krátkých zpráv

uživateli. Pro zobrazení snackbaru z widgetu, widget musí být ve stromu widgetů pod widgetem *Scaffold*. Poté stačí zavolat funkci *showSnackbar()* na instanci třídy *ScaffoldMessenger,* které je parametrem předán S*nackbar*.

5.6.4 Validace vstupních hodnot

GraphQL endpoint search vyžaduje, aby byla vyplněna buď lokace, nebo byly poskytnuty souřadnice. Pokud by nebylo vyplněné obojí, požadavek by selhal. Z tohoto důvodu je validováno ve filtrovacím dialogu vstupní pole lokace a tlačítko „moje lokace“. Pokud se uživatel pokusí použít filtr, který nesplňuje tyto podmínky, je mu zobrazena hláška, informujícího o této skutečnosti. Hláška se zobrazuje pod vstupním polem a je důležité, zde udělat pro ni dostatek místa, aby obrazovka při zobrazení neposkočila. Pro případ, že by uživatel nedával pozor, je hláška doprovozena velmi rychlou vibrací. Pro použití vibrací je potřeba balíček vibration.

5.7 Implementace Google Maps

5.7.1 Použití ve frameworku Flutter

Použití Google maps ve frameworku Flutter je velmi jednoduché. Slouží k tomu balíček google_maps_flutter. Nejprve je nutné získat Google maps API klíč. Pro získání klíče je potřeba, být zaregistrovat na stránce cloud.google.com, která slouží pro správu služeb poskytované Googlem. Po registraci je nutné založit projekt a v něm aktivovat Google maps pro iOS a Android. Poté je třeba klíč zadat pro Android do manifestu projektu AndroidManifest.xml. A pro iOS do třídy AppDelegate.m nebo AppDelegate.swift.

5.7.2 Práce s widgetem GoogleMaps

- Balíček google_maps_flutter poskytuje widgety pro práci s mapou. Nejdůležitější třídou je widget *GoogleMaps*. Ten je umístěn na místo ve stromu widgetů, kde má být zobrazena mapa. Dále zde uvedu důležité parametry konstruktoru widgetu *GoogleMaps* a k čemu slouží.

- **mapType** – Specifikuje typ mapy. Typy mapy je například terénní, normální, satelitní nebo hybridní.
- **myLocationEnable** – Specifikuje, zda se má zobrazovat indikátor mé polohy.
- **initialCameraPosition** – Počáteční pozice kamery a její přiblížení.
- **markers** – Seznam instancí třídy *Marker,* které slouží pro zobrazí bodů na mapě.
- **OnMapCreated** – Funkce, která se zavolá po vytvoření mapy a předá parametrem vytvořenou instanci třídy *GoogleMapController.*

Další důležitou třídou je *GoogleMapController*. Tato třída slouží k ovládání mapy. Poskytuje funkce jako *animateCamera,* která posune kameru na specifikovanou lokaci animovaně, nebo *moveCamera,* která jenom přesune kameru na specifikovanou lokaci. [30]

5.7.3 Možnosti nastavení stylů mapy

V aplikaci je měněn styl mapy a bodů. Styl mapy lze nakonfigurovat ve webovém konfiguračním nástroji [19]. Konfiguraci z konfiguračního webového nástroje, lze poté vyexportovat do textového řetězce ve formátu JSON. V aplikaci se takto vyexportovaný styl, vloží jako parametr funkce *GoogleMapsController.setMapStyle()*.

Vzhled bodů lze změnit při konstruování instance třídy *Marker*, takže každý bod na mapě může vypadat jinak. Toho jsem využil, pro vytvoření speciální grafické reprezentace bodu na mapě pro každou podporovanou kategorii. Třída *Marker* v konstruktoru přijímá parametr *icon* typu *BitmapDescriptor*, ale ve frameworku Flutter jsou ikony používány jako součást fontu ve formátu .ttf. Pro získání instance třídy *BitmapDescriptor* z fontu, je tedy nutné vytvořit instanci třídy *Canvas* s parametrem *PictureRecorder*, na který se vykreslí ikona s určitou velikostí, odsazením a barvou. Z instance třídy *PictureRecorder* lze poté získat instanci třídy *BitmapDescritor*. Vytvoření bodu na mapě pro každou kategorii je dělané tak, že je nejdříve vykreslena ikona kapičky a poté je na ni vykreslena menší ikona kategorie.

5.8 Datová vrstva

5.8.1 Vzdálený zdroj dat

- V aplikaci jsou používány pro práci se vzdáleným zdrojem dat 3 balíčky. Balíček graphql, Artemis, dio.

- **grahpql** – Graphql je balíček, který je používán pro volání dotazů na GraphQL endpointu. Nejprve je nutné vytvořit instanci objektu *GraphQLClient,* v konstruktoru se nastaví výchozí úložiště pro cachovaná data a výchozí nastavení cachovací politiky. Také lze specifikovat parametr *link,* tím lze předat GraphQL klientovi instanci tříd *dio* a *AuthLink*. *AuthLink* slouží pro autorizaci, při každém požadavku na API připojí do hlavičky autorizační token. Vytváření požadavků na API se dělá funkcí *query(),* která přijímá graphql dotaz ve formě textového řetězce a mapu proměnných dotazu. 50
- **Artemis** – Artemis je generátor pro soubory typu .graphql. Ze souboru typu .graphql vygeneruje třídu v jazyce Dart. Tato třída přijímá parametry dotazu a lze z ní získat textovou reprezentaci dotazu. Používání generované třídy je vhodnější, jelikož je staticky typovaná. 51
- **dio** – Dio je http klient pro Dart. Dio podporuje interceptory. Které slouží k vykonávání operací před každým http požadavkem, nebo po odpovědi. Lze ho použít například pro logování http odpovědi do konzole. [33]

5.8.2 Lokální úložiště

Lokální úložiště v aplikaci používám pro uložení stavu filtru. Pro implementaci lokálního úložiště, je v aplikaci použit balíček hive. Hive je velmi rychlá, jednoduchá na používání, NoSQL databáze.

Data do databáze se ukládají následovně. Nejdříve je potřeba otevřít krabici *Box.openBox(String nazevKrabice)*, tato metoda vrátí instanci třídy Box. Do této krabice je možné vkládat objekty metodou *put(String key,Object objekt)*. Krabice se chová jako mapa, data jsou uložená párem kliče a hodnoty. Objekt lze získat zpátky metodou *get(String klic)*.

5.8.3 Lokace zařízení

Pro možnost získání lokace zařízení, je potřeba nejdříve získat oprávnění. V případě Androidu, se do manifestu projektu musí přidat oprávnění pro hrubou a jemnou lokaci. To zajistí následující 2 řádky.

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

V případě iOS je třeba přidat do souboru *info.plist* následující řádky.
<key>NSLocationWhenInUseUsageDescription</key>
<string>Vysvetleni proc aplikace potrebuje toto opravneni</string>

V aplikaci pro práci s lokací používám balíček geolocator. Geolocator obsahuje statickou třídu *Geolocator,* ta obsahuje funkce pro práci s lokací. Nejdříve je potřeba zjistit, zda má uživatel povolenou geolokační službu a potřebná oprávnění. Tohoto lze dosáhnout funkcemi s návratovou hodnotou *Future<bool> isLocationServiceEnabled(), checkPermissions().* Pokud uživatel nemá povolenou službu, nebo nemá potřebná oprávnění, lze ho požádat o přidělení oprávnění funkcí *requestPermissions().* Ta otevře nativní dialog s požadavkem o přidělení dostatečných práv. Pokud má uživatel dostatečná práva, lze funkcí *getCurrentPosition()* zjistit aktuální polohu uživatele. Ta obsahuje zeměpisnou šířku, zeměpisnou délku, nadmořskou výšku, přesnost, časovou značku, rychlost, přesnost rychlosti, iOS může zjistit i v jakém podlaží se uživatel nachází. [29]

5.9 Injektáž závislostí

Injektáž závislostí je technika pro řešení závislostí objektů. Závislosti se předávají přes konstruktory objektů a je jeden statický objekt zodpovědný za distribuci závislostí.

Pro injektáž závislostí je v aplikaci využíván balíček get_it a generátor injectable. Závislosti pro get_it jsou generovány na základě anotací. Dále budou anotace, které jsou používané v projektu a k čemu slouží.

- **@singleton** – Vytvoří instanci třídy, která je používána napříč aplikací. Při novém požadavku na vytvoření instance, předá tuto instanci.
- **@lazySingleton** – Chová se stejně jako singleton, akorát je líně inicializovaný. Líná inicializace znamená, že se instance vytvoří, až těsně předtím, než je poprvé použita. Tento princip urychluje vytváření instancí s velkým počtem závislostí, jelikož se nemusí vytvářet každá závislost hned.
- **@LazySingleton(as: AbstraktniTrida)** – Tato anotace způsobí to, že třída je injektovaná jako instance rozhraní které implementuje. V projektu tuto anotaci používám pro injektáž repositářů přes jejich rozhraní.
- **@injectable** – Vytvoří instanci třídy.

- **@factoryMethod** – Slouží k registraci asynchronně vytvářené instance. Touto anotací se označí funkce s návratovou hodnotou *Future<InjektovatelnaTrida>,* která bude použita místo konstruktoru. Používá se hlavně, pokud je potřeba při inicializaci objektu, provolat nějakou asynchronní metodu, protože v konstruktoru nelze volat asynchronní metody.
- **@module** – Anotací *@module* je označena abstraktní třída zodpovědná za injektování tříd třetích stran. Také se do této třídy vkládají asynchronní závislosti, které mají být před vytvořeny.
- **@preResolve** – V abstraktní třídě anotované *@module*, může být tato anotace použita, pro označení asynchronní závislosti, která má být před vytvořena. Používá se, pokud je potřeba zajistit, že asynchronní metoda volaná při konstrukci objektu doběhla a provedené změny, jsou platné.

[14]

5.10 Správa verzí zdrojového kódu

Díky správě verzí kódu lze sledovat historii změn kódu. Změny jsou vratné, takže pokud se někde udělala chyba, lze se vrátit v historii a opravit ji, nebo celou změnu vrátit. VCS také umožňuje jednoduché zálohování na vzdálené datové úložiště. Pro VCS používám nástroj Git, pro zálohování na vzdálené úložiště službu Github a pro práci s nástrojem Git nástroj Git Kraken.

5.10.1 Správa verzí s nástrojem Git

Git je open-source distribuovaný systém pro kontrolu verzí, navržen pro správu projektů všech velikostí rychle a efektivně. Git repositář lze spravovat příkazy psanými do emulátoru terminálu, nebo existují i grafické nástroje.

Princip fungování

Git funguje v podstatě jako malý souborový systém nad adresářem. Změny kódu jsou prováděny takzvanými commity. Commit je datová struktura, která obsahuje ukazatel na snímek repositáře v konkrétním časovém bodě, ukazatel na předchozí commit, uživatele, který jej vytvořil, zprávu a svoje id. Git je velmi efektivní, jelikož neukládá vždy všechny soubory znovu, ale pouze provedené změny v souborech.

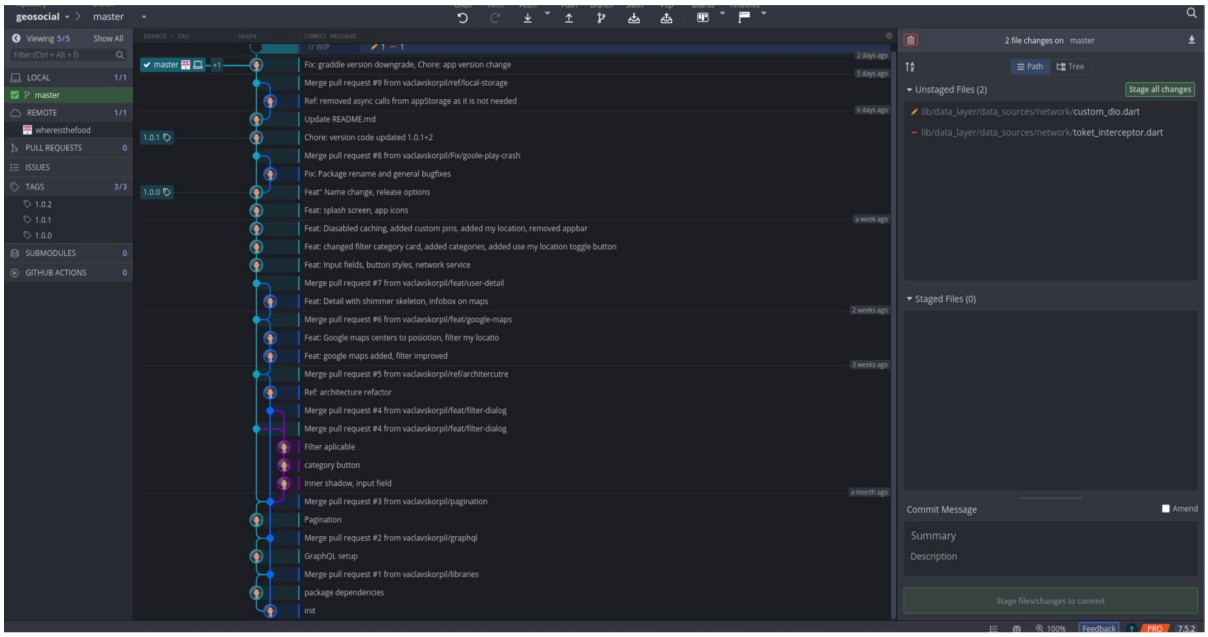
Důležitou funkcí podporovanou v Gitu je větvení. Větev je v podstatě pohyblivý ukazatel na commit. Při založení Git repositáře se automaticky vytvoří větev master. Při vytváření commitů, tato větev vždy ukazuje na poslední commit. Vytvořením nové větve se tedy vytvoří nový ukazatel na commit, na kterém byla větev vytvořena. Při práci s repositářem se vždy uživatel nachází v nějaké větvi, která se posouvá s jeho commity. Tento princip je důležitý pro souběh více uživatelů pracujících ve stejném Git repositáři. [28]

5.10.2 Datové úložiště Github

Github je webová služba pracující s Git repositáři. Na Githubu lze vytvořit projekt, do kterého jsou odesílány lokální změny v Git repositáři. Github umožňuje nejenom zálohování a synchronizaci repositáře, ale poskytuje i další nástroje pro jednodušší práci v týmech, nebo nástroje pro vytváření CI/CD potrubí. Při vývoji se tvoří commity lokálně a po každém logickém celku se odešlou na Github.

5.10.3 Grafické rozhraní Git Kraken

Git Kraken je grafický program pro práci s nástrojem Git. Lze propojit nejenom se službou Github, ale i s dalšími podobnými službami jako Gitlab, Bitbucket, nebo Azure DevOps. Díky tomu lze spravovat všechny repositáře vzdálené repositáře z jednoho prostředí. Jeho další výhodou je i přívětivé uživatelské rozhraní a přehledný graf commitů.



Obrázek 11: Git historie aplikace v programu Git Kraken. Zdroj: Vlastní.

5.11 Testování aplikace

5.11.1 Android

Aplikace byla testována na svém mobilním zařízení Samsung A40 s operačním systémem Android 10, v průběhu vývoje byl telefon aktualizován na operační systém Android 11. Důležité je testovat i balíček, který je vytvořen pro Google Play store, jelikož kompilátor zde dělá více optimalizací a může se chovat jinak.

5.11.2 iOS

Pro otestování aplikace na operačním systému iOS, je potřeba mít zařízení s operačním systémem MacOS pro vytvoření aplikace. Jelikož takovéto zařízení autor nevlastní, využil službu *CodeMagic.com*. CodeMagic je služba pro CI/CD, která navíc kromě samotného sestavení aplikace nabízí, možnost připojit se ke vzdálené ploše zařízení, na kterém sestavení probíhá. Zde je nainstalován iOS simulátor, na kterém lze otestovat aplikaci.

5.12 Umístění aplikace na Google Play

Google play je oficiální obchod pro aplikace se systémem Android. Pro vydání vlastní aplikace je nutné mít vývojářský účet, za který je poplatek 35 $. Účet i vydané aplikace se spravují ve webové aplikaci Google Play console. Aplikace je dostupná na Google Play pod názvem „Where is the food?“. Lze stáhnout také následujícím QR kódem.

Obrázek 12: QR kód pro stažení aplikace. Zdroj: Vlastní.

5.12.1 Prezentace aplikace

Pro reprezentaci aplikace v obchodě Google Play, je potřeba vytvořit snímky aplikace, ikonu aplikace a velké logo které je zobrazené v náhledy aplikace na Google Play. Pro vytvoření těchto materiálů autor použil webovou aplikaci Figma. Při vytváření loga se inspirovat ikonou, která často representuje sdílení. Do té poté přidal ikony jídla používané v aplikaci. Velké logo udělal tak, že rozšířil ikonu o další typy jídla a název aplikace.

Obrázek 13: Ikona a velké logo aplikace. Zdroj: Vlastní.

5.12.2 Vydání aplikace

Nejprve je potřeba vytvořit aplikaci ve webové aplikaci Google Play console. Vyplnit k ní základní údaje a nastavení. Přidat prezentační materiály a snímky obrazovek pro různé velikosti telefonu. Aplikace Google Play console je velmi intuitivní a provede uživatele vším co má vyplnit. Následně je možné vytvořit vydání.

Zde se přidá podepsaný app bundle aplikace, informace o vydání, název vydání. Google před vydáním aplikaci vždy kontroluje a je nutné čekat, dokud ji neschválí. To může trvat od pár desítek minut až po několik dní. Pokud je aplikace schválena, je ihned vydána na Google Play store. Lze i nastavit, aby po schválení, aplikace nebyla vydána, ale uživatel jí poté vydal, kdy chce.

ZÁVĚR

Cílem práce bylo analyzovat dostupná aplikační rozhraní geosociálních sítí a vytvořit multiplatformní mobilní aplikaci využívající aplikační rozhraní geosociální sítě.

V první kapitole byly charakterizovány geosociální sítě obecně a ve druhé kapitole zanalyzovány aplikační rozhraní jednotlivých geosociálních sítí. Ke každému aplikačnímu rozhraní je vytvořena i SWOT analýza. Z analýzy autor dospěl k závěru, že Foursquare API je pro použití v malé aplikaci nevhodné, hlavně z důvodu její ceny. Google places vycházejí z analýzy nejlépe, ale nakonec pro aplikaci bylo použito aplikační rozhraní geosociální sítě Yelp, hlavně kvůli možnosti použití GraphQL.

V 3. kapitole je teoretický úvod do frameworku Flutter, jeho architekturu a základy jazyka Dart pro psaní klientských aplikací. Dále zde je objasněno základní názvosloví a prvky používané ve frameworku Flutter, které jsou dále v práci často používány. Ve 4. kapitole byl zpracován úvod do dotazovacího jazyka GraphQL. Autor v aplikaci s GraphQL pracuje pouze ze strany klienta, proto zde není hlouběji probrána serverová část GraphQL.

V páté kapitole jsou popsány důležité kroky v celém cyklu vývoje aplikace. Byl zde i udělán teoretický úvod do návrhového vzoru BloC a vrstvené architektury. Dále je zde popsána práce s animacemi, webovými službami a nakonec vydání aplikace.

Framework Flutter, by autor zhodnotil jako velmi dobrou volbu pro vývoj multiplatformní aplikace. Framework je jednoduchý na používání a je zde velká podpora komunity. Nestalo se mu, že by narazil na nějakou limitaci tohoto frameworku. V komunitě je velká podpora pro začínající vývojáře a lze naleznout spoustu tutoriálů, které se zabývají základní problematikou vývoje a návrhu aplikace. Samozřejmě framework má i nějaké vady na kráse, konkrétně problém s prvním načtením obrazovky, kde působí lehce zasekaně. Na tomto problému však tým vývojářů ve Flutteru pracuje a věřím, že se jim v budoucnu podaří vyřešit.

POUŽITÁ LITERATURA

[1] *Foursquare developers* [online]. New York City, Chicago, Seattle, San Francisco, Los Angeles: Foursquare, ©2021 [Cit. 20. 1. 2021]. Dostupné z: https://developer.foursquare.com/

[2] *Foursquare* [online]. New York City, Chicago, Seattle, San Francisco, Los Angeles: Foursquare, ©2021 [Cit. 20. 1. 2021]. Dostupné z: https://foursquare.com/

[3] *Foursquare developers* [online]. New York City, Chicago, Seattle, San Francisco, Los Angeles: Foursquare, ©2021 [Cit. 20. 1. 2021]. Dostupné z: https://developer.foursquare.com/places

[4] *Flutter documentation* [online]. Flutter, ©2021. [Cit. 20. 1. 2021] Dostupné z: https://flutter.dev/docs

[5] Foursquare. Endpoints. *Foursquare developers* [online]. New York City, Chicago, Seattle, San Francisco, Los Angeles: Foursquare, ©2021 [Cit. 20. 1. 2021]. Dostupné z: https://developer.foursquare.com/docs/places-api/endpoints

[6] KYSELA, J.: Comparison of Web Applications Geolocation Service. In: IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI 2014). Budapešť: Óbuda University, 2014. ISBN 978-1-4799-5338-7, ISSN 1860949X

[7] Kysela J. (2019) Analysis of Usability of Various Geosocial Network POI in Tourism. In: Applied Informatics. ICAI 2019. Communications in Computer and Information Science, vol 1051. Springer 2019, ISBN: 978-3-030-32474-2, ISSN 18650929

[8] Yelp. Fast Facts. *An introduction to Yelp Metrics as of December 31, 2020* [online]. Yelp Inc., ©2021. [12. 2. 2021] Dostupné z: https://www.yelp-press.com/company/fast-facts/default.aspx

[9] Yelp. Developers. *Yelp* [online] Yelp Inc., ©2021. [Cit. 2. 3. 2021] Dostupné z: https://www.yelp.com/developers/

[10] Yelp. Display requirements. *Yelp* [online] Yelp Inc., ©2021. [Cit. 2. 3. 2021] Dostupné z: https://www.yelp.com/developers/display_requirements

[11] Yelp. API Terms of Use. *Yelp* [online] Yelp Inc., ©2021. [Cit. 2. 3. 2021] Dostupné z: https://www.yelp.com/developers/

[12] Yelp. Business. *Yelp* [online] Yelp Inc., ©2021. [Cit. 18. 2. 2021] Dostupné z: https://www.yelp.com/developers/graphql/objects/business

[13] Yelp. Search. *Yelp* [online] Yelp Inc., ©2021. [Cit. 18. 2. 2021] Dostupné z: https://www.yelp.com/developers/graphql/query/search

[14] Milad Akarie. injectable. *Dart packages* [online]. Flutter, ©20. 3. 2021. [Cit. 18. 3. 2021]. Dostupné z: https://pub.dev/packages/injectable

[15] Concise Software. What is Flutter? Here is everything you should know. In: *Medium.com* [online]. 26. 8. 2019 [Cit. 8. 3. 2021]. Dostupné z: https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f

[16] Flutter. Flutter 2.0.0 release notes. *Flutter – Beautiful native apps in record time*. [online] Flutter, ©2021. [Cit. 18. 3. 2021]. Dostupné z : flutter.dev/docs/development/tools/sdk/release-notes/release-notes-2.0.0

[17] *Fuchsia* [online] Fuchsia, ©2021. [Cit. 10. 3. 2021]. Dostupné z: https://fuchsia.dev/

[18] Flutter. Flutter architectural overview. *Flutter – Beautiful native apps in record time*. [online] Flutter, ©2021. [Cit. 18. 3. 2021]. Dostupné z: https://flutter.dev/docs/resources/architectural-overview

[19] *Styling Wizzard: Google Maps APIs* [online] Google, ©2021. [Cit. 12. 4. 2021]. Dostupné z: https://mapstyle.withgoogle.com/

[20] *Dart programming language* [online] Google, ©2021. [Cit. 2. 4. 2021]. Dostupné z: https://dart.dev/

[21] Dart. A tour of the Dart language. *Dart programming language* [online] Google, ©2021. [Cit. 2. 4. 2021]. Dostupné z: https://dart.dev/guides/language/language-tour

[22] Dart. Dart overview. *Dart programming language* [online] Google, ©2021. [Cit. 2. 4. 2021]. Dostupné z: https://dart.dev/overview

[23] Felix Angelov. BloC. *Dart packages* [online]. BlocLibrary, ©18. 3. 2021. [Cit. 10. 4. 2021]. Dostupné z: https://pub.dev/packages/bloc

[24] Google. Places API policies. *Google developers* [online] Google, ©2021. [Cit. 10. 4. 2021]. Dostupné z: *https://developers.google.com/maps/documentation/places/web-service/policies*

[25] Google. Usage and billing. *Google developers* [online] Google, ©2021. [Cit. 10. 4. 2021]. Dostupné z: https://developers.google.com/maps/documentation/places/web-service/usage-and-billing

[26] Khalil Stemmler. What is GraphQL? GraphQL introduction. In: *apollographql.com/blog* [online]16. 2. 2021. [Cit. 10. 4. 2021]. Dostupné z: https://www.apollographql.com/blog/what-is-graphql-graphql-introduction/

[27] The GraphQL Foundation. Schemas and Types. *GraphQL | A query language for your API* [online] The GraphQL Foundation, ©2021. [Cit. 10. 4. 2021]. Dostupné z: https://graphql.org/learn/schema/

[28] CHACON, Scott, STRAUB Ben. *Pro Git* [online] 2. vyd. USA: Apress, 2014. [Cit. 10. 4. 2021]. Dostupné z: https://git-scm.com/book/en/v2/

[29] BaseFlow. geolocator. *Dart packages* [online]. Flutter, 14. 4. 2021. [Cit. 20. 4. 2021]. Dostupné z: https://pub.dev/packages/geolocator

[30] Flutter. google_maps_flutter. *Dart packages* [online]. Flutter, 6. 4. 2021. [Cit. 20. 4. 2021]. Dostupné z: https://pub.dev/packages/google_maps_flutter

[31] ZinoApp. graphql. *Dart packages* [online]. Flutter, 7 2. 2021. [Cit. 20. 4. 2021]. Dostupné z: https://pub.dev/packages/graphql

[32] Borges. artemis. *Dart packages* [online]. Flutter, 18. 2. 2021. [Cit. 20. 4. 2021]. Dostupné z: https://pub.dev/packages/artemis

[33] Flutter china. dio. *Dart packages* [online]. Flutter, 18. 3. 2021. [Cit. 20. 4. 2021]. Dostupné z: https://pub.dev/packages/dio

[34] Flutter. Flutter explicit animations. *Flutter – Beautiful native apps in record time*. [online] Flutter, ©2021. [Cit. 18. 3. 2021]. Dostupné z: https://flutter.dev/docs/codelabs/explicit-animations

[35] Flutter. Flutter implicit animations. *Flutter – Beautiful native apps in record time*. [online] Flutter, ©2021. [Cit. 18. 3. 2021]. Dostupné z: https://flutter.dev/docs/codelabs/implicit-animations

[36] Google. Overview. *Google developers* [online] Google, ©2021. [Cit. 10. 4. 2021]. Dostupné z: *https://developers.google.com/maps/documentation/places/web-service/overview*

[37] Google. Search. *Google developers [online] Google, ©2021. [Cit. 10. 4. 2021]. Dostupné z: https://developers.google.com/maps/documentation/places/web-service/search*

[38] *Google. Details. Google developers [online] Google, ©2021. [Cit. 10. 4. 2021]. Dostupné z: https://developers.google.com/maps/documentation/places/web-service/search*