FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webová aplikace Meet me now!
Student:	Vladyslav Kalachykov
Vedoucí:	Ing. Oldřich Malec
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Vytvořte prototyp webové aplikace pro vyhledávání událostí a rychlého setkání lidí mezi sebou. Cílovou skupinou jsou osoby, které si chtějí zahrát deskovou hru, zacvičit jógu či zajít na výlet. Aplikace umožní jednoduché vložení události a její zobrazení ostatním uživatelům.

Postupujte v těchto krocích:
- Formalizujte požadavky na výsledné řešení.
- Analyzujte již existující řešení problematiky a porovnejte jejich funkce s vašimi požadavky.
- Navrhněte design aplikace pomocí vhodných nástrojů.
- Zvolte vhodné technologie pro implementaci a jejich volbu řádně zdůvodněte.
- Na základě analýzy a návrhů implementujte funkční prototyp.
- Prototyp otestujte a zhodnoťte výsledné řešení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 28. června 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace Meet me now!

Vladyslav Kalachykov

Katedra softwarového inženýrství
Vedoucí práce: Ing. Oldřich Malec

23. června 2022

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

Praha dne 23. června 2022 . . . . . .

Odkaz na tuto práci

Kalachykov, Vladyslav. *Webová aplikace Meet me now!*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací funkčního prototypu aplikace pro tvorbu a vyhledávaní událostí. Na základě analýzy stávajících řešení a relevantních technologií byla vytvořena jednostránková webová aplikace *"Meet me now"*, která řeší problém snadného vyhledávání událostí ve zvolené lokalitě.

Pro napsání praktické části mé bakalářské práce byly zvoleny tyto technologie: Firebase Firstore jako databáze, Firebase Authentication pro autentizace uživatelů, Firebase Storage pro chránění obrázků. React byl zvolen jako knihovna pro napsání jednostránkové webové aplikace. Pro vyřešení problémů snadného vyhledávání událostí byly použity Google Maps jako technologie pro zobrazování událostí v blízkém okolí uživatele.

Klíčová slova React, Redux, webová aplikace, jednostránková webová aplikace, SPA, Firebase, Google Maps

Abstract

This bachelor thesis deals with the design and implementation of a functional prototype application for event creation and retrieval. Based on the analysis of existing solutions and relevant technologies, a single page application *"Meet*

me now" was created to solve the problem for easy event retrieval in the selected location.

For writing the practical part of the given bachelor thesis, the technologies chosen were Firebase Firstore as database, Firebase Authentication for user authentication, Firebase Storage for image protection. React was chosen as the library to write a single page application. To solve the problem of easy event retrieval, Google map was used as a technology to display events in the user's vicinity.

Keywords React, Redux, web application, single page application, SPA, Firebase, Google Maps

Obsah

Seznam obrázků

Kapitola 1

Úvod

Zábava je nedílnou součástí života všech lidí od nepaměti. Člověk přece nemůže být pouze pragmatická a praktická bytost, která jen pracuje a přináší světu užitek. Člověk potřebuje dovolenou, aby nabral sílu, naučil se něco nového, viděl se s přáteli, bavil se, zpíval písničky, vyprávěl si příběhy, navštěvoval koncerty oblíbených kapel a mnoho dalšího. To vše je nutné, aby lidé získali nové zážitky a emoce. Vždyť co jiného dělá náš život plnohodnotnější nežli emoce?

V roce 2020 však bohužel celou planetu postihlo neštěstí v podobě pandemie koronaviru. Vlády po celém světě začaly kvůli bezpečnosti lidí omezovat všechny velké akce, zavíraly restaurace, kina, rušily provoz tělocvičen, koncerty a lidem nezbývalo nic než vysedávat v bytě a chodit na procházky do parku. Ale lidé jsou společenské bytosti a možnost komunikovat a vidět se navzájem, a to nejen prostřednictvím notebooku, ale také osobně, je velkou nutností. Bez toho trpí lidé špatnou náladou nebo rovnou depresemi. A zhoršení duševního zdraví může přitížit zdraví fyzickému a mít negativní dopad na tělo a kvalitu života.

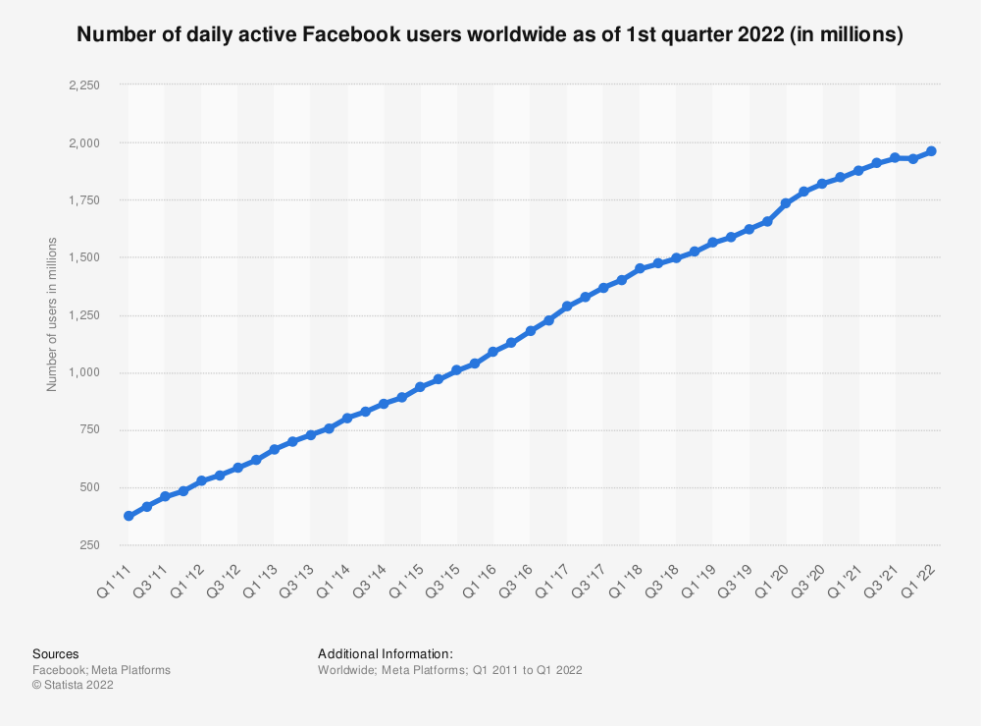
Cílem práce je navrhnout a implementovat prototyp aplikace, která uživateli usnadní proces vyhledávání událostí v jeho blízkosti, a to nejen v rozsahu konkrétního města. To bude provedeno analýzou stávajících zdrojů určených k tomuto účelu, výběrem hlavního výčtu požadovaných funkcí a doplněním těch chybějících. Budou vybrány použitelné technologie, které jsou schopny rychle, efektivně a účinně zajistit požadované funkce pro vývoj aplikace.

Výsledkem analýzy konkurenčních zdrojů a technologií potřebných k realizaci konceptu, bude jednostránková webová aplikace, která uživatelům umožní vytvářet události, vyhledávat je na mapě v blízkosti jejich polohy, zanechávat komentáře a hodnotit pořadatele. Projekt bude vytvořen s využitím potřebné architektury a struktury projektu, která umožní škálovatelnost a přidávání nových funkcí v budoucnu.

Kapitola 2

Analýza stávajících řešení

Pro analýzu byly zvolené webové stránky jako Facebook a Goout. Facebook má více než 1.9 miliardy denně aktivních uživatelů (viz obrázek 2.1).

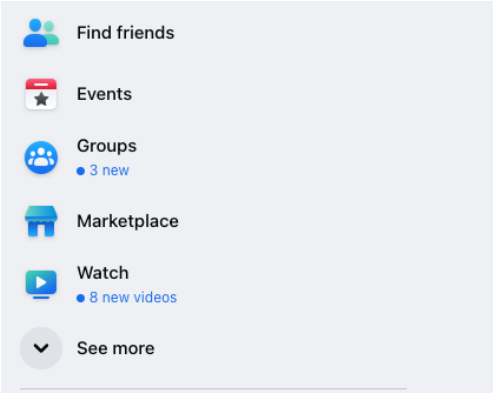


Obrázek 2.1: Počet denně aktivních uživatelů Facebooku na celém světě v 1. čtvrtletí roku 2022 [1].

2.1 Facebook

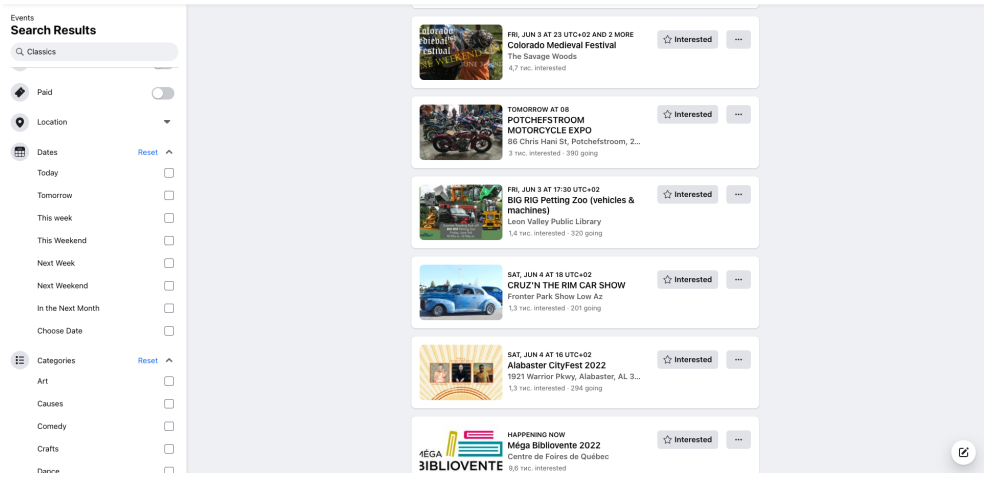
Jedním z příkladů, kde uživatelé mohou vyhledávat události, je sociální síť **Facebook**[1]. Na stránkách si uživatel může najít záložku *Events* (viz obrázek 2.2), kde se zobrazuje výběr nadcházejících událostí, které jsou zobrazeny jako karty. Vlevo může uživatel vybrat kategorii události (viz obrázek 2.3a). Při pokusu o výběr kategorie se stránka znovu načte, již otevřené události se zobrazí jako seznam a levá nabídka filtru se změní a otevře možnost výběru data. Dokud uživatel neklikne na kategorii, neotevře se ani filtr pro výběr data. Jediné, co události na Facebooku do té doby nabízejí, je tlačítko *Tento týden* (viz obrázek 2.3b) v horní části stránky. Jeden týden je příliš velké časové rozpětí. Pokud uživatel chce vybrat události, které se budou konat dnes, ale nemá zájem o výběr kategorie, tak neklikne na výběr kategorie, díky čemuž se mu nezobrazí nabídka s výběrem data a on může použít tlačítko *Tento týden*.

Další nevýhodou *Events* je propagace události pomocí reklamy. Ovlivňuje to, co uživatel uvidí a místo relevantních doporučení může uživatel dostat nějakou reklamu. Třetí nevýhodou je, že není možné vyhledávat události v užších lokalitách. *Events* nabízí seznam míst, ale na výběr jsou pouze města, případně čtvrti v rámci města. V kombinaci s tím, že *Events* obsahuje mnoho velkých událostí, které se mohou konat až za měsíc/půl roku/rok, by uživateli mohlo být znemožněno vyhledávat menší a aktuálnější události. Například návštěva parku na pivo, nebo setkání s milovníky psů ze sousedství při společném venčení.

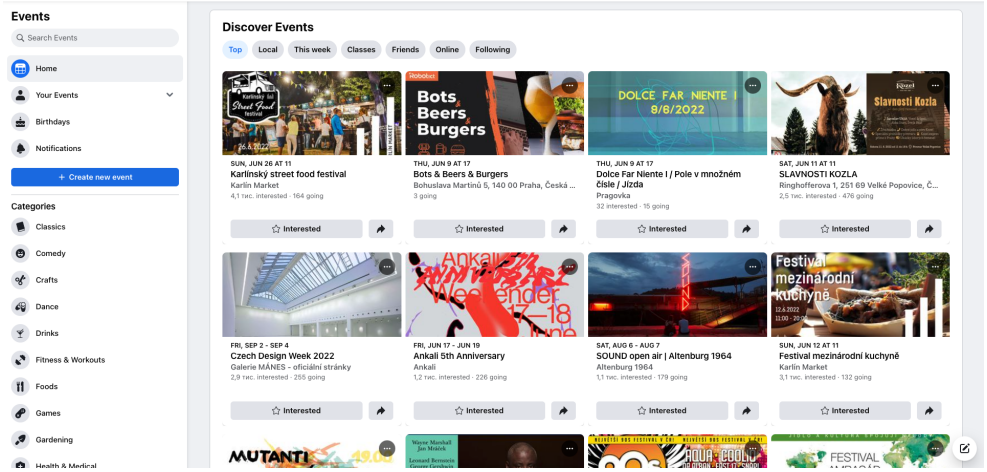


Obrázek 2.2: Facebook události.

[1] `https://facebook.com`



(a) Filtrace podle kategorie.



(b) Tlačítko *"Tento týden"*.

Obrázek 2.3: Filtry.

2.2 GoOut

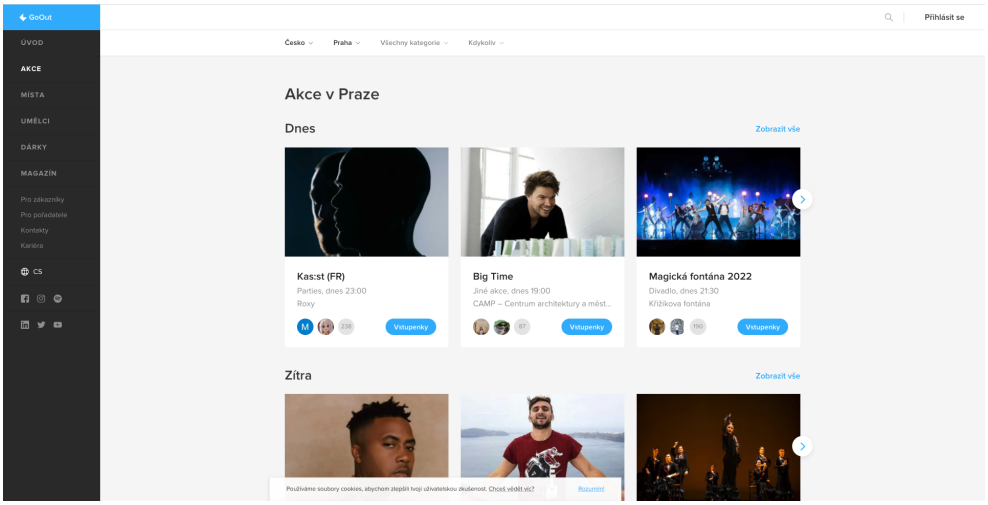
GoOut[2] je webová stránka pro vyhledávání a organizování akcí, která umožňu-

je vyhledávat události podle data, kategorie a místa (viz obrázek 2.4). Tato aplikace je pro uživatele intuitivnější než *"Facebook events"*, ale stále existuje několik problémů, kvůli kterým není tak snadné tuto stránku jednoduše používat k vyhledávání události v blízkosti, nebo k rychlému vytváření událostí. Jednou z nevýhod, stejně jako u *"Facebook events"*, je absence možnosti vyhledávání ve specifičtější lokalitě nežli město tak, jak je vidět na obrázku 2.6. Při vyhledávání čtvrti Karlín je možné vybrat pouze města nebo obce s po-

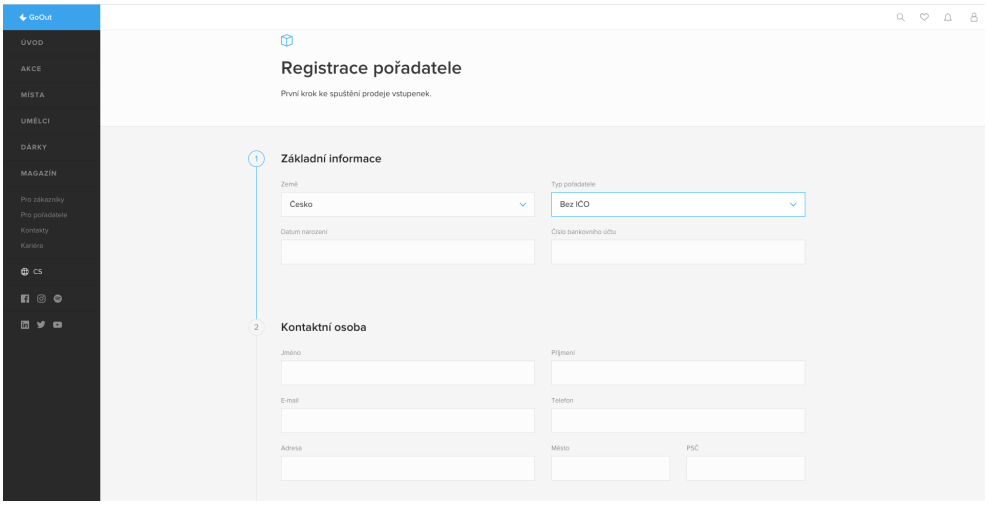
[2] `https://goout.net`

dobným názvem. Chybí výběr takové čtvrti v Praze, nebo vyhledávání čtvrti na mapě, což způsobuje nízkou efektivitu vyhledávání dle polohy.

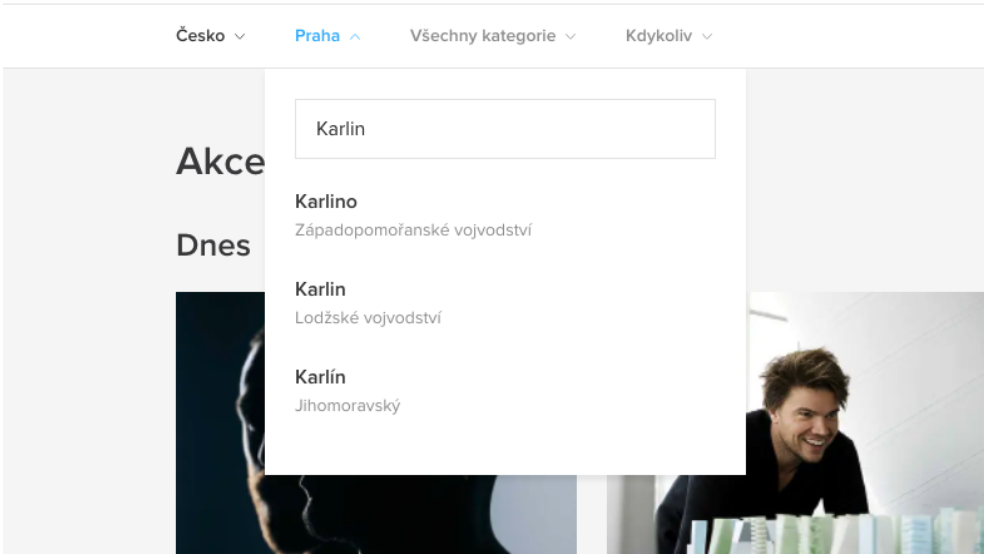
Druhou, ale neméně důležitou, nevýhodou je, že pro vytvoření události nestačí být jen uživatelem. Uživatel si musí vytvořit účet *"organizátora"*, kde vyplní buď své IČO, nebo číslo bankovního účtu (viz obrázek 2.5). Všechny události na tomto webu jsou placené, což činí vytváření rychlých akcí nepohodlným.



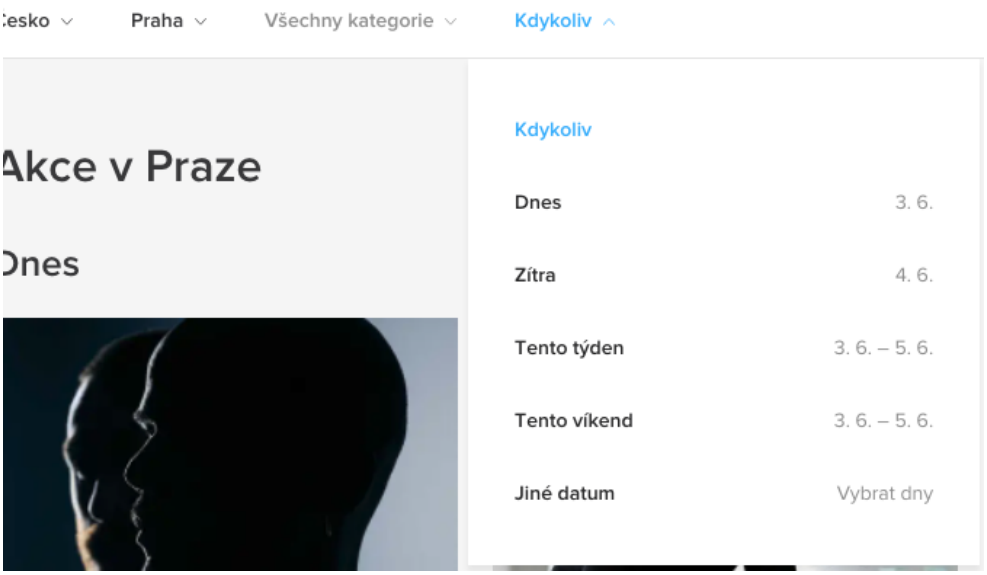
Obrázek 2.4: Seznam událostí.



Obrázek 2.5: Registrace účtu organizátoru.



Obrázek 2.6: Filtrace podle města.



Obrázek 2.7: Filtrace podle daty.

Kapitola 3

Analýza relevantních technologií

Před vytvořením webové aplikace je nutné zvolit si správné technologie pro realizaci nápadu.

Existují dva hlavní přístupy k vývoji webových aplikací: *jednostránkový* (**SPA**) a *vícestránkový* (**MPA**). Tyto přístupy se odlišují podle jednoduchosti vývoje, uživatelské přívětivosti a šířce možností vývoje aplikací.

Aplikace na jedné stránce, SPA, je jednostránková webová aplikace, která se načítá na jedinou stránku **HTML**. Kvůli dynamické aktualizace pomocí JavaScript není třeba během používání znovu načítat nebo načítat další stránky. V praxi to znamená, že uživatel vidí veškerý hlavní obsah prohlížeče a při posouvání nebo přechodu na jiné stránky se místo úplného znovunačtení jednoduše načtou požadované prvky. Při práci může mít uživatel pocit, že používá spíše desktopovou, aplikaci než webovou stránku, protože reaguje na všechny jeho akce okamžitě, bez zpoždění.

Níže jsou popsány výhody SPA v porovnání s jinými typy webových aplikací:

1. Vysoká rychlost - všechny zdroje se načítají v jedné relaci a data se jednoduše mění během akcí na stránce, což šetří mnoho času.

2. Flexibilní a responzivní uživatelské rozhraní - existuje pouze jedna webová stránka. Je snazší vytvořit bohaté rozhraní, ukládat údaje o relaci, spravovat stavy zobrazení a animace.

3. Zjednodušený vývoj - kód lze spustit z adresy `file://URL` bez použití serveru a k vykreslení stránky na straně serveru není potřeba žádný samostatný kód.

4. Ukládání dat do mezipaměti - aplikace odešle pouze jeden požadavek, shromáždí data a poté může pracovat offline.

Níže jsou popsány nevýhody SPA ve srovnání s jinými typy webových aplikací:

1. SEO vyžaduje řešení vykreslování na straně serveru, protože obsah se načítá pomocí technologie **AJAX**, která zahrnuje dynamické změny obsahu a pro optimalizaci je důležitá stabilita stránky.
2. Načítání prohlížeče - rámce na straně klienta jsou náročné, jejich načítání trvá dlouho.
3. Podpora JavaScriptu je nutná. Bez JS nelze využívat všechny funkce aplikace.
4. Úniky paměti v JavaScriptu - kvůli špatnému zabezpečení je SPA náchylnější k útokům a únikům paměti.

Vícestránkové aplikace (MPA) jsou vícestránkové aplikace, které pracují podle tradičního schématu. To znamená, že při každé drobné změně údajů, nebo nahrání nových informací se stránka aktualizuje. Tyto aplikace jsou náročnější než jednostránkové aplikace a jsou užitečné pouze v případě, že je třeba zobrazit velké množství obsahu. Níže jsou popsány výhody MPA ve srovnání s jinými typy webových aplikací:

1. Snadná optimalizace SEO - každou stránku aplikace je možné optimalizovat pro požadované klíčové dotazy.
2. Uživatelská přívětivost - díky jednoduchému rozhraní a klasické navigaci.

Níže jsou popsány nevýhody MPA v porovnání s jinými typy webových aplikací:

1. Úzká vazba mezi backendem a frontendem, takže je nelze vyvíjet souběžně
2. Komplexní vývoj - vyžadují použití frameworků na straně klienta i serveru, což prodlužuje dobu vývoje a zvyšuje rozpočet.

Po analýze potřeb aplikace *"Meet me Now"* jsou zdůrazněny následující body:

1. Potřeba bohatého a funkčního uživatelského rozhraní.
2. Potřebu rozhraní **API**, což znamená, že k sestavení aplikace se používají předem připravené bloky.
3. Rychlost.
4. Flexibilita.
5. Snadný a rychlý vývoj.
6. Ukládání dat do mezipaměti.

Na základě těchto bodů se dá určit, že pro vývoj aplikací by byla vhodnější SPA.

3.1 Frameworky

V současné době existují frameworky pro jednostránkové aplikace jako je *React*, *Angular*, *Vue*, *Meteor*, *Svetle*, *Ember*, *Knockout*. Níže uvedené obrázky 3.1 ukazují statistiky nejpoužívanějších freameworků pro tvorbu jednostránkové aplikace. Nejpopulárnější jsou React, Angular a Vue.

Angular jako framework se používá k vývoji velkých podnikových aplikací, protože je k němu dodávána řada architektonických řešení včetně TypeScriptu. To může být na jedné straně omezení, ale na druhé straně je to rychlé řešení standardního případu. Angular jako framework je vysoce specializovaný. Vstupní úroveň pro vývoj s Angular je velmi vysoká. Kromě znalostí průměrného vývojáře JS je nutné pochopit, jak je tento framework nastaven.

React je v podstatě knihovna a má velký arzenál souvisejících knihoven. Je vhodný pro jakýkoliv úkol. Lze jej použít pro MVP 3.2 a malé i střední projekty. Podpora jazyka TypeScript pomůže při vývoji v podnikové sféře. Lze jej také zvolit pro vývoj webových stránek s minimální obchodní logikou na přední straně.

Vue je vhodný pro středně velké aplikace a MVP. Vysoká rychlost při spuštění umožňuje vytvořit MVP v krátkém čase. Škálovatelnost aplikace Vue také pomůže dynamicky rozvíjet projekt. Verze 3.0 má plnou podporu jazyka TypeScript. Počáteční náklady na tento framework jsou nižší než u frameworku React díky uživatelsky přívětivému CLI a dobré sadě podporovaných knihoven. Snadná škálovatelnost je zase vyšší než u Angularu. Kromě toho je Vue jednodušší na naučení, takže společnosti kladou menší nároky na najímání vývojářů Vue.

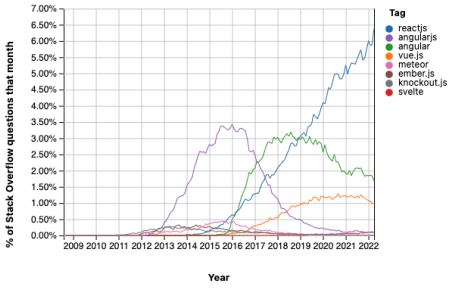
3.2 Srovnání na základě výkonu

Při vývoji webových aplikací je výkonnost produktu přímo závislá na **DOM**. Tento model představuje webovou stránku ve webových formulářích i v kódu. Pomocí DOM lze snadno spravovat webové stránky v případě aktualizací.

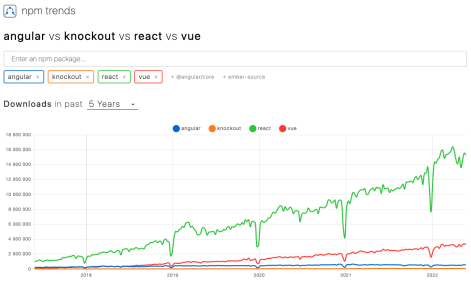
Angular běžně používá základní DOM. Překresluje se už v případě jediné změny. To je jeho velkou nevýhodou jednoduše proto, že základní DOM má velký vliv na výkon webové aplikace, zejména pokud jde o jednostránkové webové aplikace.

V tomto případě jsou React i Vue.js poměrně robustní, protože používají virtuální DOM. Zde je možné provádět změny, aniž byste ovlivnili DOM nebo jinou funkci aplikace.

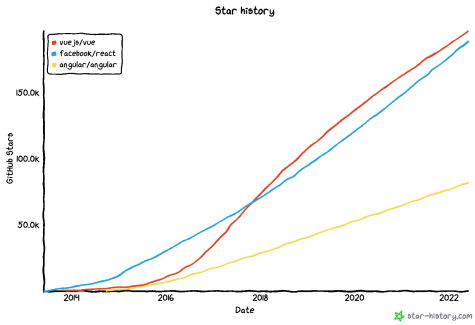
Virtuální DOM se porovná se snímkem základního DOM a poté se vykreslí pouze změněné prvky. Tento přístup zvyšuje výkonnost aplikace.



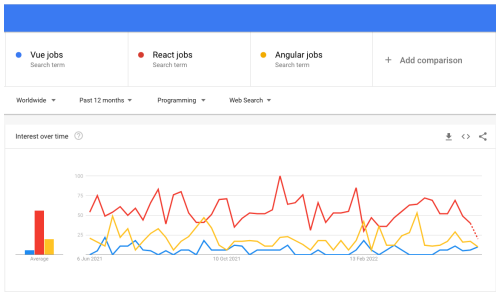
(a) Procento dotazů položených ohledně frameworky na StaskOverflow [2].



(b) Statistika stahování [3].

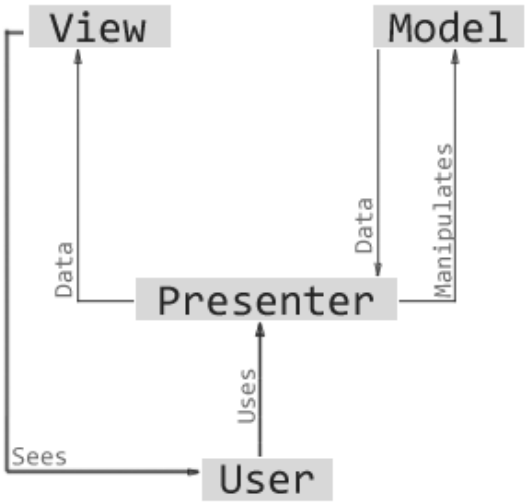


(c) Počet hvězd na Github u jednotlivých frameworku [4].



(d) Statistiky pracovních nabídek podle frameworku [5].

Obrázek 3.1: Statistika SPA frameworků.



Obrázek 3.2: MVP pattern.

3.3 Velikost vnějšího rámce

V případě projektu vývoje softwaru záleží na velikosti rámce. Rámce rozhraní totiž přímo souvisejí s výkonem produktu. Kromě toho je třeba načíst aplikaci a framework dlouho předtím, než je aplikace funkční.

Angular je nejnáročnější ze všech (asi 143K), následuje React (asi 43K) a poté Vue.js, který je s 23K nejlehčí ze všech.

Z analyzovaných frameworků pro psaní SPA jsem si pro psaní své webové aplikace vybral knihovnu React, protože je dostatečně jednoduchá a s její pomocí lze snadno a rychle napsat rozhraní, včetně opakovaně použitelných komponent, a na výstupu je možné získat vysoký výkon a škálovatelnost. V úvahu je třeba vzít i statistické údaje, které říkají, že při výběru mezi Reactem, Angular a Vue dávají vývojáři přednost Reactu.

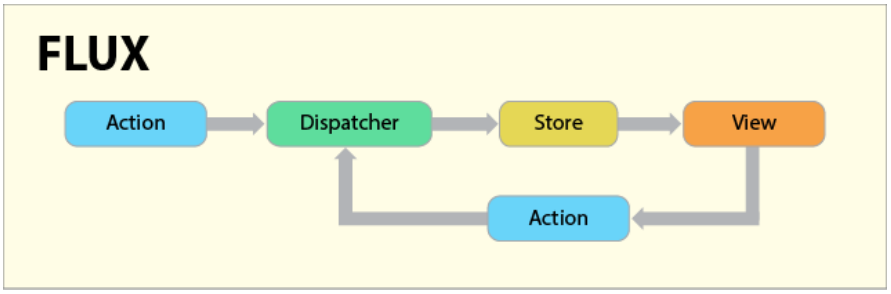
3.4 Architektura Flux

Flux je architektura, kterou vyvinul vývojový tým společnosti Facebook pro spolupráci s frameworkem React. Je důležité si uvědomit, že Flux není framework ani knihovna. Je to architektura, šablona nebo vzor, podle kterého se vytvářejí aplikace React. React a Flux jdou ruku v ruce.

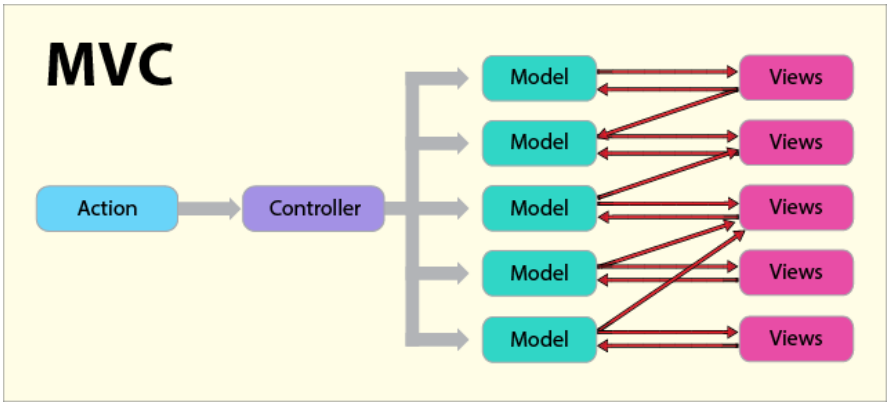
Většina programátorů zná přístup **MVC** (viz obrazek 3.4. Při navrhování aplikací pomocí tohoto modelu bude React zodpovědný za "View". Zjednodušeně řečeno, React bude zodpovědný za vnější rozhraní aplikace. V tomto případě bude Flux zodpovědný za "Model", což znamená šablonu, podle které

bude aplikace navržena, a propojení všech jejích modulů. MVC a Flux jsou však odlišné přístupy k vývoji.

Flux je v podstatě zodpovědný za vytváření datové vrstvy aplikace v jazyce JavaScript a také za chod na straně serveru (viz obrazek 3.3).



Obrázek 3.3: Flux pattern.



Obrázek 3.4: MVC pattern.

Nyní se musíme rozhodnout, jak implementovat architekturu Flux. Mezi nejoblíbenější implementace Fluxu patří *Redux*, *MobX*, *Recoil*.

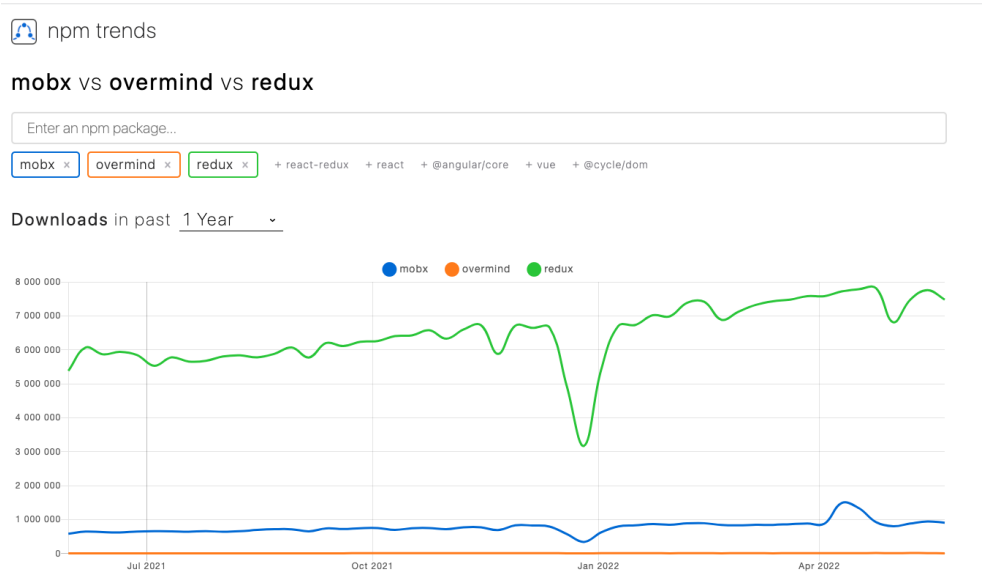
Statistiky stažení těchto řešení ukazují, že nejoblíbenějším řešením je Redux (viz obrázek 3.5). Knihovna Redux má navíc ve srovnání s konkurencí nejmenší velikost, pouhých 1,6 KB.

Z výše uvedených důvodu byl vybrán Redux.

3.5 Databáze

Pro ukládání dat aplikace, v případě této bakalářské práce *"user"*, *"event"*, je nutné mít databázi jako úložný prostor. Dále budou probrané některé příklady databází.

Oracle Database je objektově-relační systém pro správu databází (DBMS) od společnosti Oracle. Slouží k vytvoření struktury nové databáze, jejímu na-



Obrázek 3.5: Statistiky stahování implementace Flux [6].

plnění, úpravě obsahu a zobrazení informací. Každá tabulka obecně obsahuje údaje týkající se podobných subjektů. Každá tabulka má svůj název, který odpovídá tomu, jaké informace jsou v tabulce uloženy.

Tabulky se skládají z řádků a sloupců. Každý sloupec má jedinečný název, který také označuje typ uložených informací. Každý řádek obsahuje informace o jednom objektu. Tabulka obsahuje určitý počet sloupců, ale může mít libovolný počet řádků.

Práce s takovým úložištěm je mnohem složitější než s běžnou tabulkou. Počet záznamů se může pohybovat v milionech. Ruční čtení informací je téměř nemožné, a proto se pro práci s databázemi používá speciální programovací jazyk, který se nazývá SQL.

MongoDB je dokumentově orientovaný systém správy databází, který nevyžaduje popis schématu tabulky. Je považován za jeden z klasických příkladů systémů NoSQL.

Mezi funkce této databáze patří:

1. Napříč platformami. DBMS je vyvinut v programovacím jazyce C++, takže jej lze snadno integrovat pod jakýkoliv operační systém (Windows, Linux, MacOS atd.).

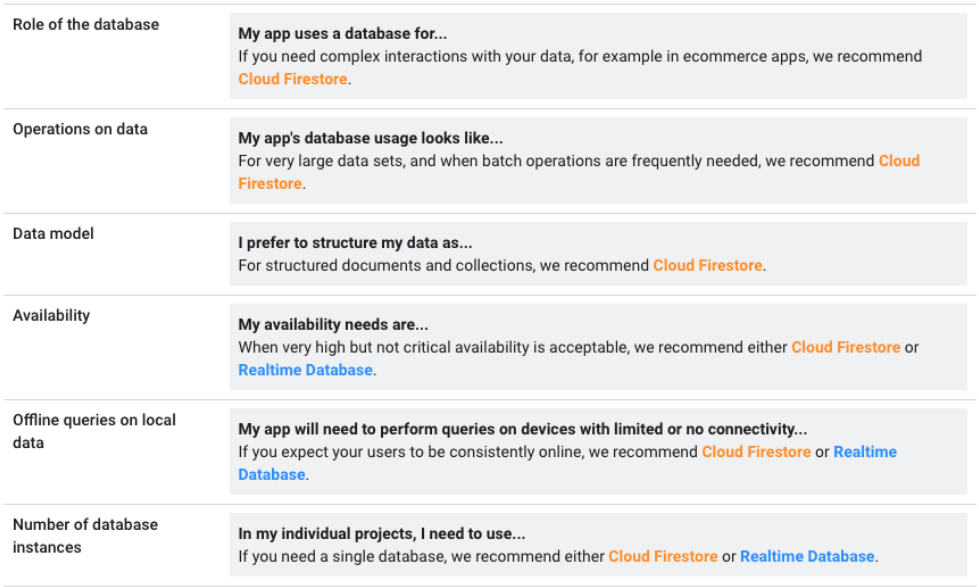
2. Formát dat. MongoDB používá vlastní formát ukládání informací - binární objektovou notaci JavaScript (BSON), která je založena na jazyce JavaScript.

3. Dokument. Zatímco relační databáze používají řetězce, MongoDB používá dokumenty, které ukládají hodnoty a klíče.

4. Místo tabulek používá MongoDB kolekce. Obsahují různé typy datových sad

5. MongoDB používá k ukládání velkých dat vlastní technologii GridFS, která se skládá ze dvou kolekcí. První obsahuje názvy souborů a metadata souborů. Druhá část ukládá segmenty informací, jejichž velikost nepřesahuje 256 Kb.

3.6 Firebase



Obrázek 3.6: Test výběru databáze.

Firebase je cloudová služba vytvořená společností *Google*. Hlavní výhodou je, že umožňuje vývojáři, aby se nerozptyloval tvorbou backendu, tj. softwarové části projektu skryté před uživatelem, např. kódu serveru. Mezi poskytované funkce patří dvě databáze: **Firebase Realtime Database** a **Cloud Firestore**. Firebase funguje také jako úložiště souborů. Úložiště **Firebase Storage** zajišťuje spolehlivé nahrávání a stahování souborů pro aplikaci. Cloudové úložiště Google podporuje ukládání videa, zvuku nebo jakéhokoliv jiného typu souboru. Obsah cloudového úložiště je bezpečně chráněn vlastním bezpečnostním systémem. Další poměrně důležitou funkcí, kterou Firebase může poskytnout, je **Firebase Authentication**, ve které je možné ověřování uživatelů aplikace pomocí hesla a e-mailu. Firebase Authentication podporuje

také otevřený ověřovací protokol *OAuth 2.0*, který používají společnosti Google, Twitter a Facebook. Systém ověřování Firebase je integrován přímo do databáze [7].

Vzhledem ke všem funkcím, které nabízí Firebase, vyberu databázi z toho, co nabízí tato služba, protože kromě toho pro svou aplikaci budu také potřebovat možnost přihlásit se do aplikace pomocí uživatelského jména a hesla, stejně jako s účtem Google. K ukládání fotografií uživatelů se bude používat cloudové úložiště Google.

Firebase Realtime je databáze NoSQL hostovaná v cloudu [8]. Data jsou uložena ve formátu JSON a synchronizována v reálném čase pro každého připojeného klienta. Rozhraní API databáze v reálném čase je určeno pouze pro operace, které lze provádět rychle.

Cloud Firestore je flexibilní a škálovatelná databáze pro vývoj mobilních zařízení, webů a serverů od Firebase a Google Cloud [9]. Stejně jako databáze Firebase Realtime synchronizuje data mezi klientskými aplikacemi pomocí naslouchání v reálném čase a nabízí offline podporu pro mobilní zařízení a web, takže lze vytvářet responzivní aplikace, které běží bez ohledu na latenci sítě nebo připojení k internetu. Cloud Firestore taky může provádět bezproblémovou integraci s ostatními produkty Firebase a Google Cloud, a to včetně cloudových funkcí.

Podle datového modelu NoSQL ukládá úložiště Cloud Firestore data v dokumentech obsahujících pole mapovaná na hodnoty. Tyto dokumenty jsou uloženy v kolekcích, což jsou kontejnery pro dokumenty, které můžete použít k organizaci dat a vytváření dotazů. Dokumenty podporují mnoho různých datových typů, od jednoduchých řetězců a čísel až po složité vnořené objekty. Je možné vytvářet vnořené kolekce v rámci dokumentů a vytvářet hierarchické datové struktury, které se s růstem databáze škálují. Datový model úložiště Cloud Firestore podporuje jakoukoli strukturu dat. Kromě toho jsou dotazy ve službě Cloud Firestore expresivní, efektivní a flexibilní. Lze vytvářet mělké dotazy pro načtení dat na úrovni dokumentu, aniž by byla potřeba načítat celou kolekci nebo vnořené podsbírky. K dotazům nebo kurzorům lze také přidat třídění, filtrování či omezení a rozdělit tak výsledky na stránky.

Aby nebylo nutné po každé aktualizaci dat načítat znovu z databáze, lze přidat posluchače v reálném čase. Přidání posluchačů v reálném čase do aplikace bude upozorňovat pomocí snímku dat vždy, když se data, kterým klientská aplikace naslouchá, změní a bude přijímat pouze nové změny.

Na základě popisu, analýzy a testu (viz obrazek 3.6), který je navržen pro výběr správné databáze na stránce dokumentace [10], bude jako databaze zvolen Cloud Firestore.

3.7 UI

K vytvoření vzhledu aplikace bude používáno **Material UI**. Material UI je framework React JS, který poskytuje hotová řešení pro rychlý a poměrně jednoduchý vývoj webových stránek. Základní principy:

1. Grafičnost - založená na polygrafickém designu, kánonech typografie, budování hierarchií.
2. Metaforičnost - textury, stíny a vyvážení světla jsou převzaty z reálného světa.
3. Přehlednost - všechny ikony jsou intuitivní.
4. Dynamičnost - uplatňují se sekvenční transformace a přechody.

3.8 Map

Aplikace, která je představena v praktické části bude potřebovat mapu, na které se budou zobrazovat události. Nejoblíbenější mapou používanou uživateli internetu jsou Mapy Google. Za tímto účelem poskytuje Google Cloud Platform různé funkce prostřednictvím rozhraní API. V daném případě budu používat rozhraní JavaScript API Maps pro zobrazení samotné mapy, rozhraní Places API pro zobrazení míst (jako jsou kavárny, zajímavosti atd.) na mapě a rozhraní Geocoding API pro možnost z názvů konkrétních míst získat jejich souřadnice. K tomuto účelu slouží knihovna `react-google-maps/api`, která poskytuje velmi jednoduché propojení s JavaScriptovým rozhraním API Google Maps a umožňuje jej používat v aplikaci jako komponenty React [11].

Kapitola 4

Návrh

V této části je popsán návrh webové aplikace. Níže jsou probraný funkční a nefunkční požadavky. Taky se popisuje proces tvorby uživatelského rozhraní

4.1 Funkční požadavky

.

F1 Možnost zaregistrovat účet na základě uživatelského jména a hesla.

F2 Možnost použití aplikace pomocí uživatelského účtu.

F3 Možnost přistupovat k aplikaci po přihlášení pomocí svého účtu Google.

F4 Možnost zobrazování událostí pomocí seznamu karet.

F5 Možnost zobrazování událostí pomocí mapy. Události se na mapě zobrazí jako značka.

F6 Možnost vytvářet vlastní události.

F7 Možnost filtrovat zobrazené události podle datum.

F8 Možnost označit události, které se mu líbí.

F9 Možnost zobrazit události, které jsou označené jako seznam.

F10 Možnost na stránce události zanechat komentář.

4.2 Nefunkční požadavky

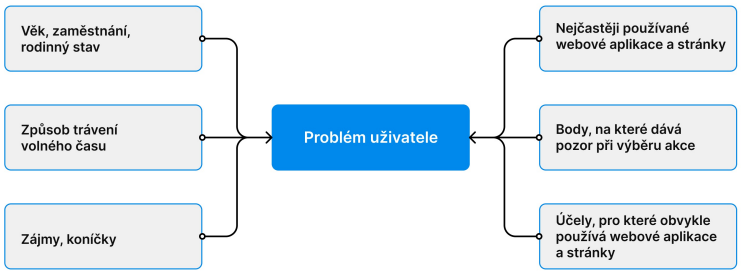
N1 Možnost práce s aplikací jenom pro registrované uživatele.

N2 Možnost používat aplikace pomocí prohlížeče Google Chrome, Firefox, Safari, Microsoft Edge.

N3 Aplikace nebude ukládat hesla do databáze, o což se postará služba Firebase Authenticaction.

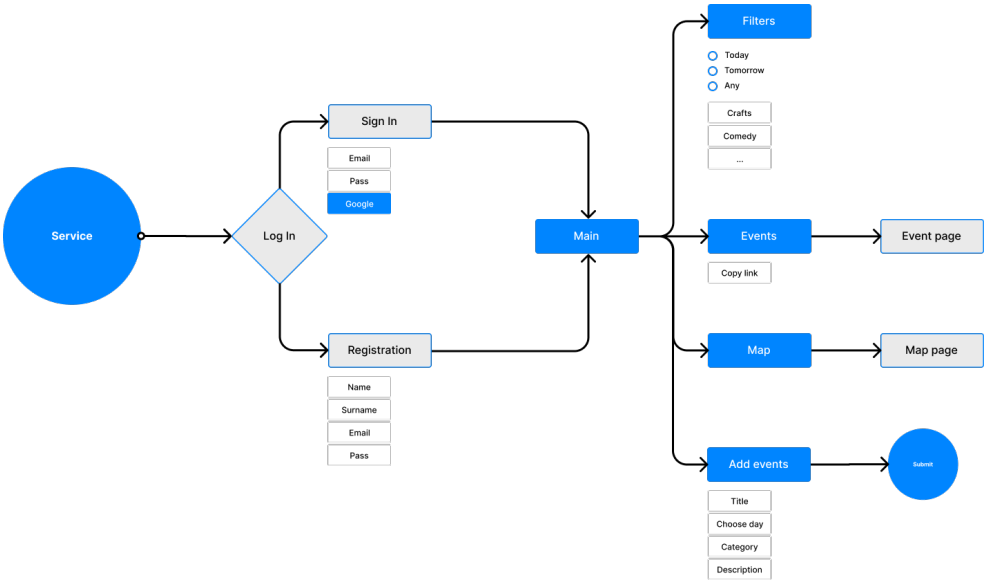
4.3 Návrh uživatelského rozhraní

V první fázi byl proveden výzkum, který pomohl určit cílové publikum a potřeby uživatele. Na základě těchto aspektů bude postaven koncept uživatelského rozhraní. Ke zjištění cílového publika a potřeb uživatelů byla použita metoda rozhovoru, během níž byly účastníkům kladeny otázky uvedené níže v diagramu 4.1. Pomocí rozhovoru se povedlo zjistit potenciální uživatelův problém, který bude vyřešen při tvorbě webové aplikace, který je právě rychlé a snadné vyhledávaní událostí ve své blízkosti, a to nejen v rozsahu konkrétního města. Na základě uživatelských odpovědí je vidět, že aplikaci pro vyhledávání událostí používají uživatelé všech věkových kategorií bez ohledu na sociální postavení, což znamená, že rozhraní by mělo být snadno použitelné a mělo by mít podobnosti s jinými obdobnými aplikacemi, aby nabízelo pro uživatele maximální pohodlí.



Obrázek 4.1: Rozhovor k určení cílového skupiny.

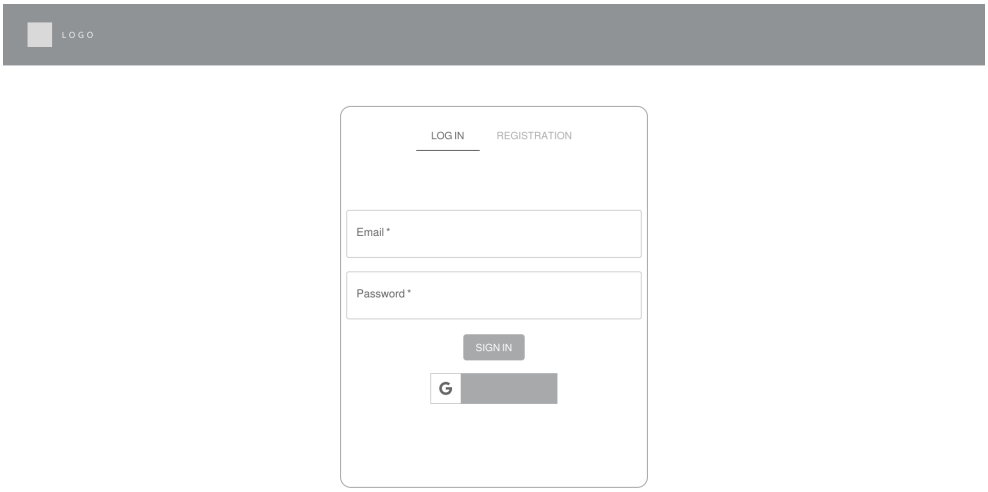
Po provedení výzkumu, prostudování jejich potřeb a stanovení úkolů, které je třeba vyřešit, byl vytvořen předpoklad, jak konkrétní uživatel bude budoucí aplikaci používat. Na zaklade vyzkumu byl vytvoren User Flow (viz obrazek 4.2). **User Flow** - vizuální reprezentace sledu akcí, které uživatel provádí, aby dosáhl svého cíle. User Flow je založen na pořadí akcí, které musí uživatel provést. To pomáhá pochopit, zda všechny procesy v produktu mají logický závěr. Při pohledu na tok uživatelů okamžitě pochopíte, co je podstatou řešení, které aplikace nabízí.



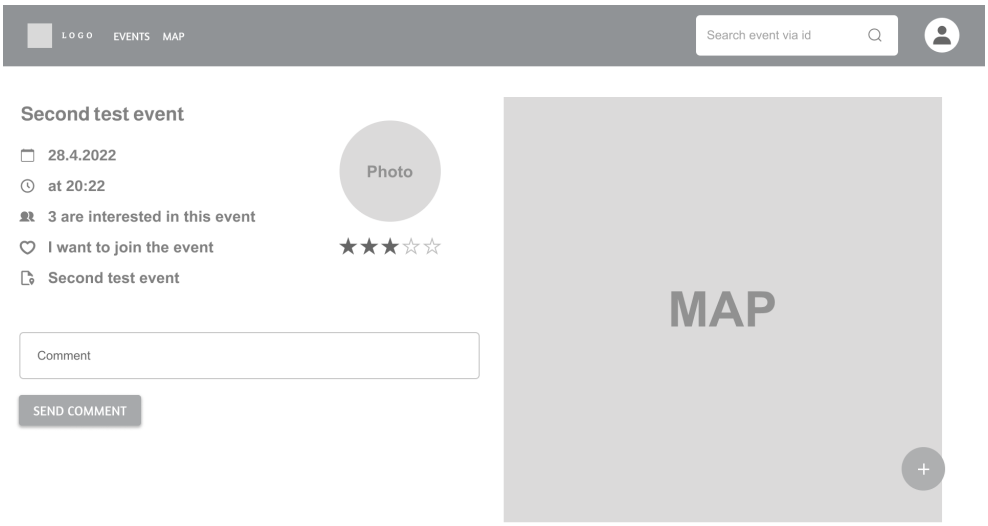
Obrázek 4.2: User Flow.

Ve třetí fázi byly ve Figmě vytvořeny warframy, které jsou nízkoúrovňovou ukázkou designu aplikace. Vytváření warframů poskytuje věcné zobrazení funkčnosti, popis dostupných funkcí, popis různých scénářů. Tato metoda umožňuje zobrazit strukturu a popis interakce uživatele s vyvíjeným produktem (viz obrázky 4.3).

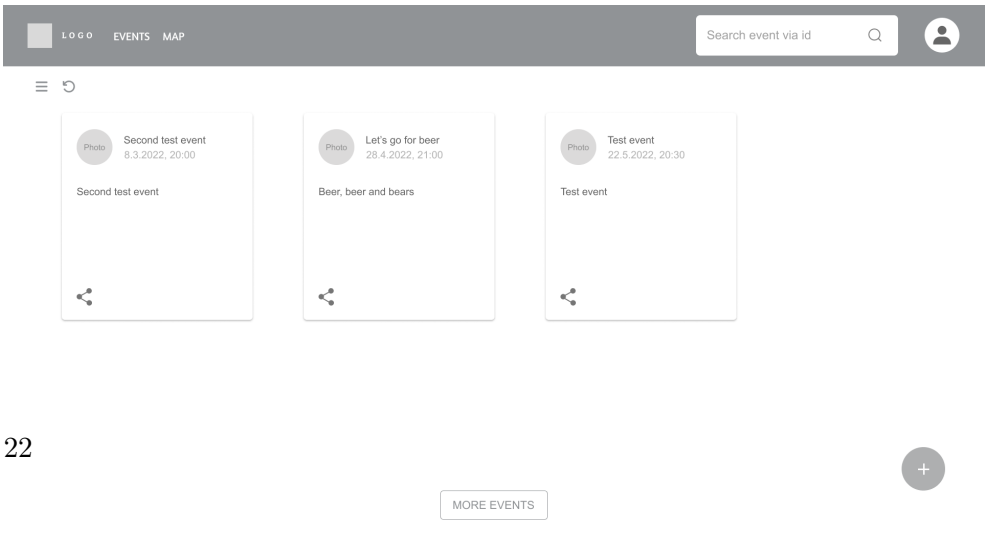
Poslední fází prototypování je vývoj **High-fidelity** prototypů (prototypy s velkou detailizací). Tento druh prototypu již vypadá jako hotová aplikace s vybraným stylem, *pixel perfect* prvky a je převeden do vývoje (viz obrázky 4.4).



(a)

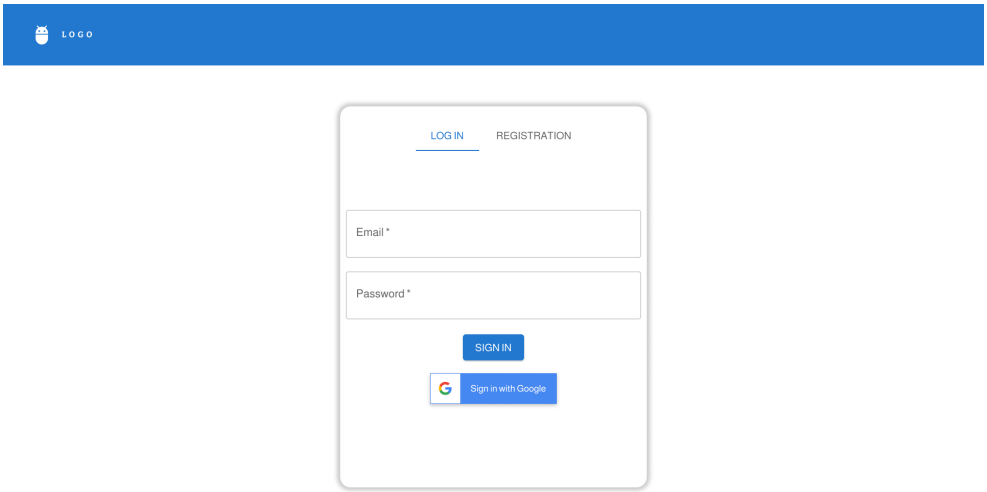


(b)

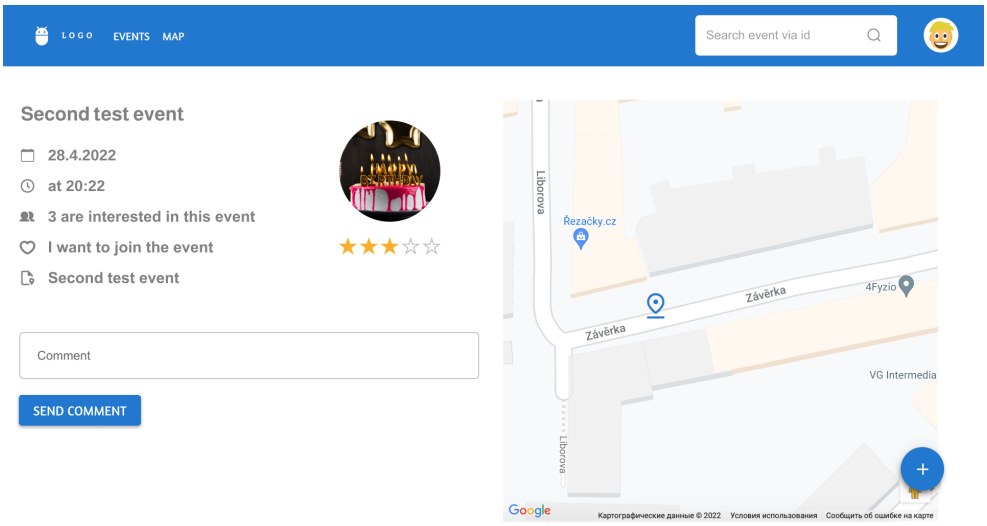


(c)

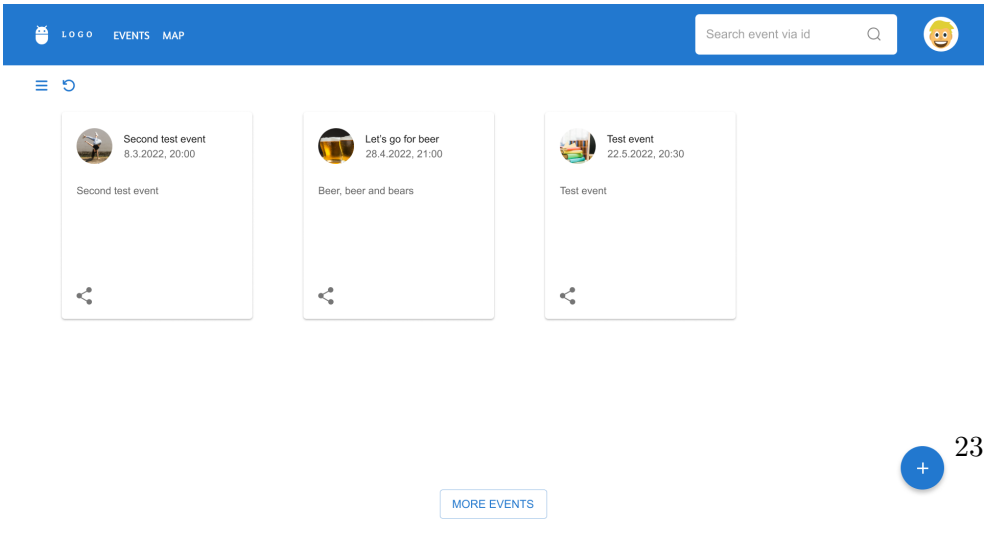
Obrázek 4.3: Figma warframy.



(a)



(b)



(c)

Obrázek 4.4: High-fidelity prototypy.

Kapitola 5

Realizace

Níže uvedený diagram (viz obrázek 5.1) znázorňuje strukturu projektu. Základní kostra byla vytvořena při vytváření projektu příkazem `create-react-app`.

- `node_modules` ..... adresář se knihovnami
- `public` ..... adresář se soubory a obrázky
  - `assets` ..... adresář se statickými obrázky
  - `index.html` ..... html stránka projektu
- `src`
  - `components` ..... adresář se soubory jednotlivých React komponent
  - `constants` ..... adresář se konstanty
  - `firebase` ..... adresář se souborem pro inicializace Firebase
  - `redux` ..... adresář se souborami pro inicializace a spravování store
  - `utils` ..... adresář se souborami s instrumenty
  - `App.css` ..... soubor stylů
  - `App.js` ..... hlavní ReactjS komponenta
  - `index.css` ..... soubor stylů
  - `index.js` ..... soubor pro inicializace React]S aplikace
- `.env` ..... soubor pro nastavení parametrů aplikace
- `.gitignore` ..... seznam souborů, které by se neměly ukládat do git
- `firebase.json` ..... konfigurační soubor pro Firebase
- `firestore.indexes.json` ..... soubor s konfiguracemi pro indexování
- `firestore.rules` ..... soubor s pravidly pro data ve Firestore
- `package.json` ..... soubor se závislostmi

Obrázek 5.1: Struktura projektu.

Veřejná složka obsahuje složku `assets/markers`, která obsahuje statické fotografie značek, které budou zobrazovat událost na mapě. Každá značka je pojmenována podle kategorie, kterou bude reprezentovat.

Další složkou je `src/components`, kde jsou uloženy komponenty React. Každá komponenta se nachází v příslušné složce spolu se soubory stylů pro

příslušnou komponentu.

Ve složce `src/components/autocomplete` je uložena komponenta, která je zodpovědná za vyhledávací panel míst a zajímavostí na mapě.

Složka `src/components/events` obsahuje několik sad komponent, které do určité míry slouží pro zobrazovaní události. První sada zobrazuje události v seznamu karet, druhá sada události na mapě, třetí sada existující událost, když si ji uživatel chce prohlédnout, a čtvrtá sada zobrazuje obrazovku pro vytvoření události, která se skládá ze tří částí. První část je stránka s informacemi o události, například název, datum, čas, kategorie a popis. Na druhé stránce může uživatel vybrat místo události. Třetí stránka umožňuje uživateli zobrazit náhled informací, které zadal při vytváření události.

Ve složce `src/header` se nachází komponenta která zobrazuje záhlaví stránky projektu. V záhlaví bude fotografie uživatele s možností otevřít kontextovou nabídku, což je tlačítko pro výběr zajímavých aktivit a tlačítko pro odhlášení z účtu. V záhlaví je také tlačítko pro volbu zobrazení událostí jako seznamu karet, nebo na mapě.

Ve složce `src/login` se nacházejí komponenty, které zobrazuji přihlašovací stránku a stránku pro registrace uživatelů.

Složka `src/map` obsahuje komponentu mapy, která se používá k zobrazení událostí na mapě nebo na stránce, kde se bude událost konat. Ve složce `src/modalWindow` se nachází komponenta odpovědná za modální okno v projektu. Tato komponenta může přijímat jiné komponenty a zobrazovat je.

Složka `src/constants` obsahuje soubor s konstantami potřebnými v aplikaci.

Složka `src/firebase` obsahuje soubor, který inicializuje *Firebase*, *Firebase Cloude Firstore*, *Firebase Authentication*, *Google Authentication Provider* a *Firebase Storage*.

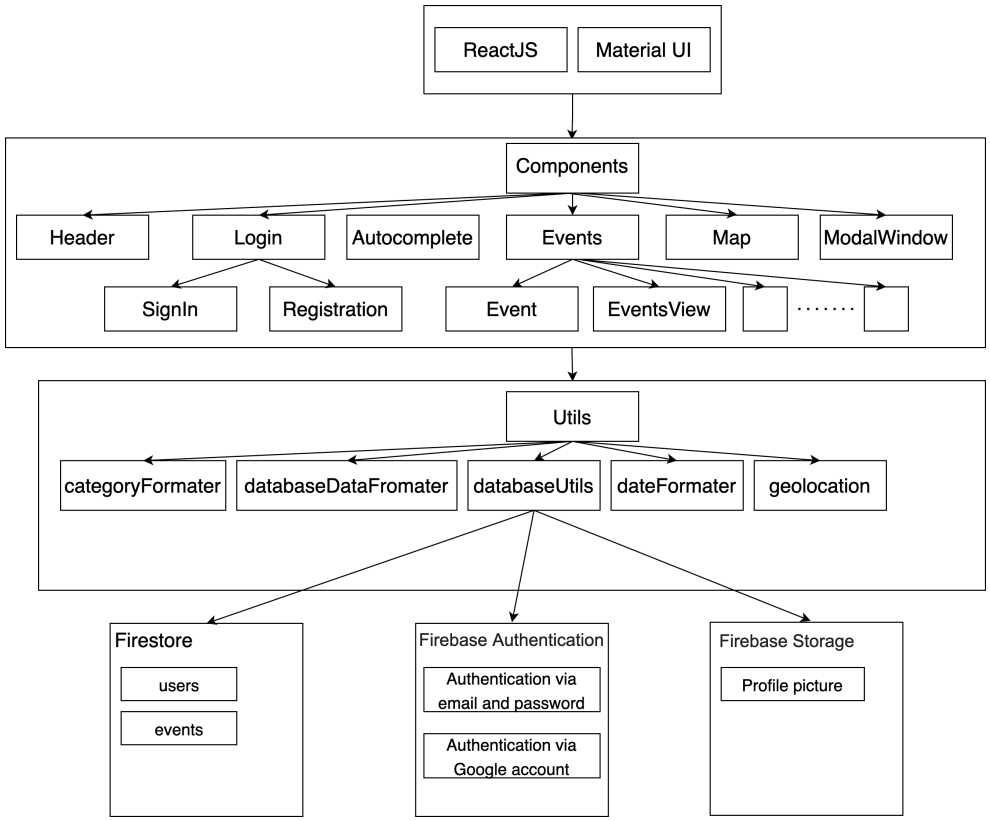
Složka `src/redux` obsahuje soubory pro inicializaci úložiště Redux a reduktory, které slouží pro provádění změn stavu.

Složka `src/utils` obsahuje pomocné soubory, které jsou zodpovědné za kompilaci objektu, který má být přidán do databáze, poskytnutí cesty ke statické ikoně tokenu na základě kategorie události, poskytnutí data v požadovaném formátu, poskytnutí souřadnic aktuální polohy uživatele a nakonec soubor, který zpracovává operace s databází.

5.1 Architektura

Níže uvedený diagram znázorňuje architekturu aplikace (viz obrázek 5.2). Vše začíná u Reactu a Material UI, které jsou použity v každé komponentě. Dále následují komponenty, které tvoří uživatelské rozhraní. Informace zobrazené těmito komponentami jsou uloženy ve službách Firebase. Za tímto účelem byl vytvořen soubor `databaseUtils` a pomocí funkcí tohoto souboru získávají

komponenty informace potřebné pro zobrazení. V souboru `databaseUtils` komunikují funkce s Firebase Firestore a získávají informace o událostech a uživatelích. Pomocí funkcí, které používají ověřování Firebase, se uživatel přihlašuje do aplikace pomocí uživatelského jména a hesla, nebo prostřednictvím účtu Google. Tento soubor také poskytuje funkce pro ukládání a načítání uživatelských fotografií z úložiště Firebase.



Obrázek 5.2: Architektura.

5.2 Použité technologie a funkce

5.2.1 Mapa

V aplikaci *"Meet me now"* má uživatel možnost interakce s mapou. Za tímto účelem byla do aplikace přidána knihovna `react-google-maps/api`. Tato knihovna poskytuje hotové řešení pro použití rozhraní **Google Maps JavaScript API**. Za funkce nabízené platformou Google Cloud Platform, jejíž součástí je stejné rozhraní Google Maps JavaScript API, je však třeba zaplatit. Google naštěstí poskytuje zkušební dobu 90 dní a také zůstatek 300 dolarů

za služby, které využívají různá API používaná uživatelem [12]. Pro napsání prototypu aplikace bylo k vyřešení využito této služby.

Při pohledu do dokumentace byla vytvořena komponenta, která tuto knihovnu používá.

```
const {isLoaded} = useJsApiLoader({
    id: 'google-map-script',
    googleMapsApiKey: process.env.REACT_APP_GOOGLE_MAP_API_KEY,
    libraries
})
```

Obrázek 5.3: Inicializace mapy.

Pro inicializaci mapy byl použit hook `JsApiLoader` (viz obrazek 5.3), který přijímá za parametry *id*, *API klíč* a *seznam knihoven*. Id bude mít wrapper pro mapu, až bude integrována do HTML. API klíč, který se zobrazuje v Google Cloud Platform po aktivaci Maps JavaScript API. Knihovna která bude s mapou pracovat je `const libraries = ['places'];`, načte požadovanou knihovnu `places`, která je zodpovědná za zobrazování názvů různých míst a zajímavostí na mapě. Aby mapa mohla tuto knihovnu používat v platformě Google Cloud Platform, je třeba aktivovat rozhraní Places API.

Pro použití mapy lze snadno přidat komponentu `GoogleMap`, jak je znázorněno na obrázku5.4.

```
<GoogleMap mapContainerStyle={containerStyle}
           center={props.center}
           zoom={17}
           onLoad={onLoad}
           onUnmount={onUnmount}
           options={defaultMapOptions}
           onClick={props.onClick}>
    {/* Markers */}
    {markers}
</GoogleMap>
```

Obrázek 5.4: GoogleMap komponenta.

Pro potřeby aplikace používám `prop`-y:

1. `mapContainerStyle` pro stylování kontejneru mapy.
2. Center pro nastavení souřadnic, na kterých se mapa zobrazí při načtení.
3. Přiblížení.

4. `onLoad` je funkce, ve které změním zaměření mapy, pokud `props.center`.
5. `onUnmout`.
6. `options` se předává objektu s nastavením funkčnosti mapy.
7. `onClick` se předává funkci, která zpracovává kliknutí levým tlačítkem myši na mapě. V případě této aplikace se kliknutím na mapu při vytváření události umístí značka. Při jiném použití mapy se nic nestane.

Tato komponenta obsahuje také ukazovatele5.5, které se přidávají jako pole ukazovatelů.

```
<Marker key={uuid()} icon={{url: iconUrl}} title={title}
                     position={{
                         lat: parseFloat(props.markers[i].lat),
                         lng: parseFloat(props.markers[i].lng)}}
                     onClick={() => {onMarkerClick(props.markers[i].id)}}/>
```

Obrázek 5.5: Ukazovatel komponenta.

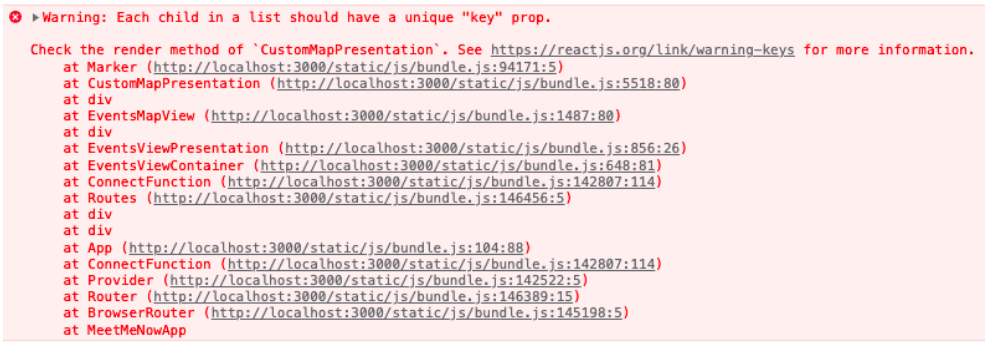
Mezi rekvizity této komponenty patří např.:

1. `Key` je vyžadován pro každý prvek pole.
2. `Title` zobrazí název události a datum, když ukazatel myší najede na značku.
3. `Position` přebírá objekt se souřadnicemi umístění značky.
4. `onClick` přijímá funkci, která zpracuje kliknutí levým tlačítkem myši na značku. V případě této aplikace přesměruje kliknutí na značku uživatele na stránku události, na které se značka zobrazila.

5.2.2 Uuid

Při zobrazování seznamu je třeba každé položce přiřadit hodnotu klíče. Klíče pomáhají systému React identifikovat, které prvky byly změněny, přidány nebo odstraněny. Musí být zadány tak, aby React mohl v průběhu času porovnávat položky pole. Pokud nejsou k prvkům seznamu přiřazeny klíče, aplikace ohlásí chybu, jaká je zobrazena na obrázku 5.6.

Oficiální dokumentace Reactu [13] nedoporučuje používat indexy polí jako klíče, pokud se pořadí položek může měnit. To bude mít negativní dopad na výkon a může způsobit problémy se stavem komponent. V praktické častí práce bylo použito **UUID** jako klíč položky seznamu. UUID je 128-bitová hodnota, která slouží k jednoznačné identifikaci objektu nebo entity na internetu [14]. K vygenerování `uuid` se využívá funkce `uuid()` 5.7, ktera poskytuje závislost `react-uuid`.



Obrázek 5.6: Key chyba.

```
function uuid() {
    const hashTable = 'abcdef0123456789'.split('')
    let uuid = []

    for (let i = 0; i < 35; i++) {
        if (i === 7 || i === 12 || i === 17 || i === 22) {
            uuid[i] = '-'
        } else {
            uuid[i] = hashTable[
            Math.floor(Math.random() * hashTable.length - 1)
            ]
        }
    }
    return uuid.join('')
}
```

Obrázek 5.7: uuid().

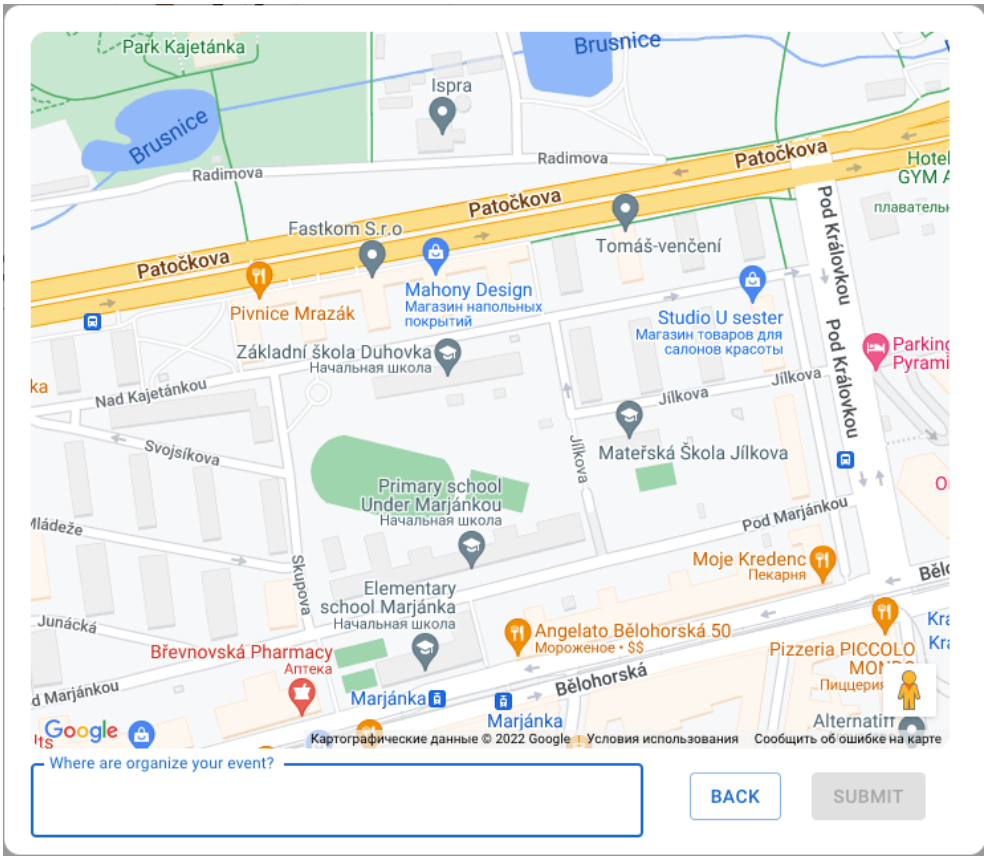
5.2.3 Autocomplete

Při vytváření události musí uživatel zadat místo, kde se bude událost konat. Bez tohoto nemůže uživatel dokončit proces vytváření události. Tlačítko *"Submit"* bude zablokováno (viz obrázek 5.8).

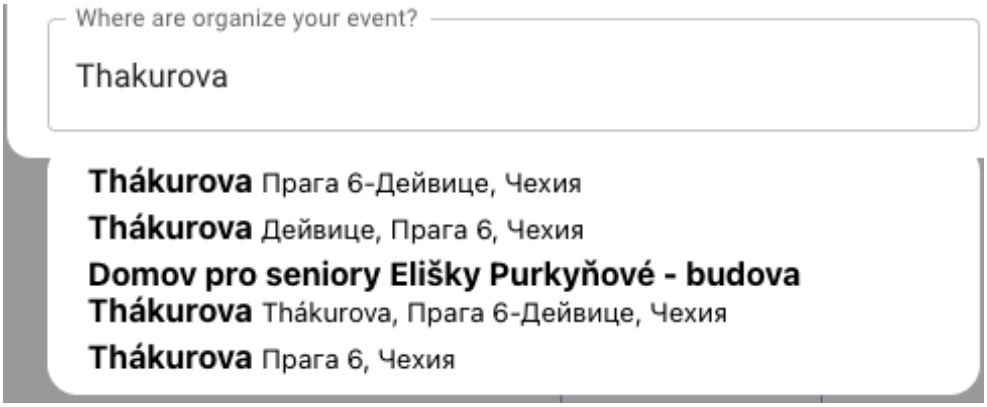
Do aplikace byla přidána závislost `use-places-autocomplete`, která usnadňuje vyhledávání míst na mapě. V dolní části mapy je vyhledávací lišta, která usnadňuje uživateli hledání místa. Závislost poskytuje React hook pro funkci automatického doplňování míst v Mapách Google, která pomáhá snadno vytvořit komponentu uživatelského rozhraní s funkcí automatického doplňování míst. Automatické dokončování s nápovědami k umístění vypadá jako na obrázku5.9.

Při výběru místa ze seznamu spustí komponenta, která má v sobě funkci automatického vyplňování, obslužnou funkci `handleSelect`.

Hlavní částí je funkce `getGeocode` (viz obrázek 5.10). Tato funkce získá



Obrázek 5.8: Mapa.



Obrázek 5.9: Automatické dokončování.

adresu místa vybraného uživatelem, pak pomocí rozhraní Geocoding API převede adresu na její souřadnice a tyto souřadnice předá funkci `props.changeMapCenter`, aby se zaměření mapy přesunulo na místo vybrané uživatelem. Geokódování je rozhraní API nabízené platformou Google Cloud

Platform, takže pro jeho fungování je třeba aktivovat rozhraní API Geocoding.

```
const handleSelect = ({ description }) => () => {
    setValue(description, shouldFetchData: false);
    clearSuggestions();

    // Get latitude and longitude via utility functions
    getGeocode( args: { address: description }).then((results : GeocodeResult[] ) => {
        try {
            const { lat, lng } = getLatLng(results[0]);
            console.log('📍 Coordinates: ', { lat, lng });
            props.changeMapCenter(lat, lng)
        } catch (error) {
            console.log('😱 Error: ', error);
        }
    });
};
```

Obrázek 5.10: Získání souřadnic z adresy na mapě.

5.2.4 Geolokace

Při používání aplikace bude uživatel opakovaně konfrontován s mapou. Někde se zobrazí značky všech existujících událostí, někde pouze jedna značka a stejně jako při vytváření události bude uživatel vyzván k výběru místa. Na takových místech jako je zobrazení všech značek, nebo při vytváření události, bude pro uživatele výhodné, když bude fokus mapy přiřazen poloze uživatele, například aby uživatel viděl události v okolí. S tím vám může pomoci rozhraní API pro určování zeměpisné polohy.

Rozhraní API Geolocation umožňuje uživateli sdílet svou polohu, pokud s tím uživatel souhlasí. Z důvodu ochrany osobních údajů bude uživatel požádán o povolení.

Rozhraní API pro určování zeměpisné polohy je přístupné prostřednictvím objektu `navigator.geolocation`. Pro získaní aktuální polohy uživatele se volí metoda `getCurrentPosition()`. Tím se iniciuje asynchronní požadavek na zjištění polohy uživatele a dotaz na polohovací hardware, aby se získaly nejnovější aktuální informace [15].

Obrázek 5.11 ukazuje implementaci této funkce pro vyhledání uživatele. Pokud z nějakého důvodu nelze zjistit aktuální polohu, vrátí tato funkce již přednastavené souřadnice.

5.2.5 Kopírování do schránky

Události jsou určeny ke spolupráci, takže může nastat situace, kdy uživatel bude chtít sdílet oblíbenou událost s přítelem. Pro usnadnění je naimplemen-

```
const defaultCenter = {
    lat: process.env.REACT_APP_PRAGUE_LAT,
    lng: process.env.REACT_APP_PRAGUE_LNG
}

export const getBrowserLocation = () => {
    return new Promise( executor: (resolve, reject) => {
        if ('geolocation' in navigator) {
            navigator.geolocation.getCurrentPosition(
                successCallback: (pos : GeolocationPosition ) => {
                    const  {latitude: lat, longitude: lng} = pos.coords;
                    resolve( value: {lat, lng});
                }, errorCallback: () => {
                    reject(defaultCenter);
                }
            );
        } else {
            reject(defaultCenter);
        }
    });
};
```

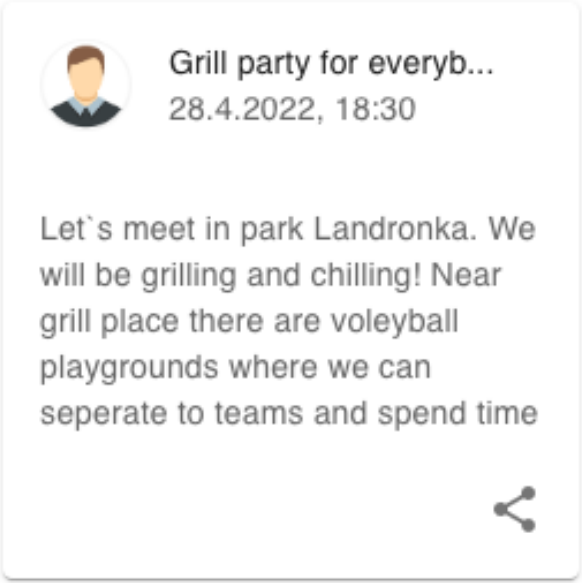
Obrázek 5.11: Získání polohy uživatele.

tovaná funkce na kopírovaní `id` události při jejich zobrazení jako seznamu karet.

Na obrázku 5.12 je uveden příklad karty události. Na každé takové kartě se v pravém dolním rohu nachází výrazné tlačítko, které po kliknutí zkopíruje ID události do schránky počítače uživatele. Tuto funkci poskytuje rozhraní Clipboard API.

Rozhraní **Clipboard API** [16] poskytuje možnost reagovat na příkazy schránky (vyjmout, kopírovat a vložit) a provádět asynchronní čtení a zápis do systémové schránky. Přístup k obsahu schránky je umožněn prostřednictvím rozhraní **Permissions API**: oprávnění k zápisu do schránky je stránkám automaticky uděleno, když jsou na aktivní kartě. K systémové schránce lze přistupovat pomocí příkazu `navigator.clipboard`. Tato funkce byla použita, aby uživatel mohl zkopírovat ID události.

Jakmile uživatel předá `id` události jinému uživateli, může uživatel toto `id` použít k hledání události prostřednictvím vyhledávacího panelu (viz obrázek 5.13).



Obrázek 5.12: Karta události.

Obrázek 5.13: Vyhledávací linka.

5.2.6 Firebase

5.2.6.1 Firebase Firestore

Jako umístění úložiště jsem zvolil databázi **Firebase Firestore**. Pro možnost snadného zápisu a načítání informace z databáze, je třeba myslet na jejich rozložení. Pro tento účel byly vytvořeny dvě kolekce:

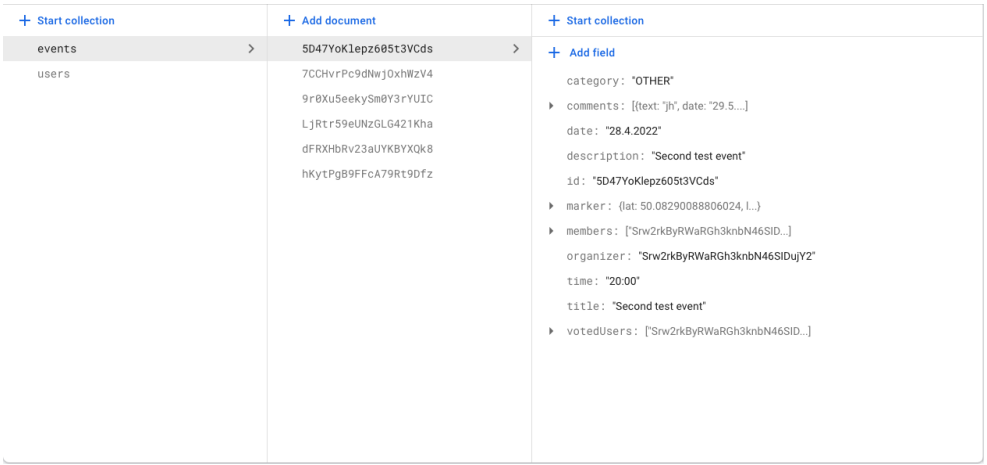
1. **Events**.
2. **Users**.

První sbírkou jsou události, které se ukládají jako dokumenty (viz obrazek 5.14).

Každý název položky je jedinečný klíč, který Firebase vytvoří pro novou položku.

Každá událost má pole pro ukládání informací:

1. `category` - je typ akce, např. DRINKS, GRILL a podobně.
2. `comments` - jedná se o pole komentářů. Každý komentář má pole `name`, `surname`, `text`, `time`, `date`, `uid` a `eventId`.



Obrázek 5.14: Eventy.

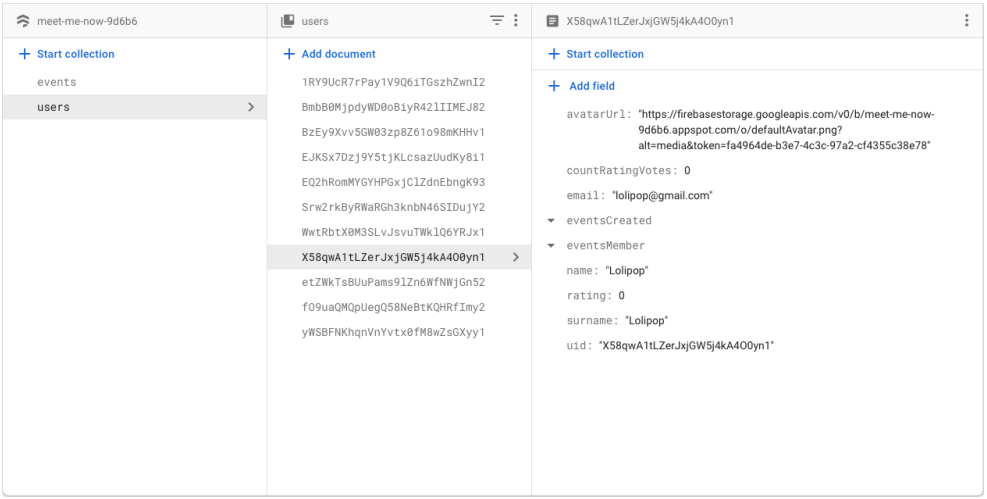
3. `date` - datum konání události.
4. `description` - popis události.
5. `id` - id události. Stejné jako to, co vygenerovala Firebase pro dokument, ve kterém se událost nachází.
6. `marker` - objekt, který ukládá souřadnice místa, kde se událost odehraje.
7. `members` - pole, které uchovává id uživatelů, kteří mají o událost zájem.
8. `organizer` - id uživatele, který událost zorganizoval.
9. `time` - čas události.
10. `title` - název události.
11. `votedUsers` - pole uživatelů, kteří hodnotili uživatele, který událost organizoval.

Druhá kolekce je zodpovědná za ukládání uživatelů (viz obrazek 5.15). Podobně jako v předchozím případě jsou v kolekci uživatelů uloženy dokumenty s osobními údaji.

Jako klíč dokumentu pro uživatele je použito hodnota `uid`, která je uživateli přiřazena při registraci pomocí ověřování Firebase. Dále jsou ukázaná pole, kde se chrání osobní udaje uživatele:

1. `avatarUrl` - zde je uložen odkaz na fotografii uživatele, kterou přidá při registraci.
2. `countRatingVotes` - do tohoto pole se ukládá počet hodnocení uživatele. Tato hodnota je nezbytná pro výpočet aritmetického průměru.

3. `e-mail` - e-mailová adresa.
4. `eventsCreated` - jedná se o pole, které uchovává id událostí vytvořených uživatelem.
5. `eventsMember` - jedná se o pole, které uchovává id událostí, o které má uživatel zájem.
6. `name` - jméno uživatele.
7. `surname` - jméno uživatele.
8. `rating` - hodnocení uživatelů.
9. `uid` - ID uživatele, které je přiděleno při vytvoření dokumentu, ve kterém budou uloženy informace o uživateli.



Obrázek 5.15: Usery.

Firebase Firestore nabízí bohaté funkce pro práci s databází. K účelům zápisu informace do databáze Firestore nabízí na výběr funkci `addDoc()` a `setDoc()`.

Pro tvorbu nebo změny dokumentu existuje funkce `setDoc()`. Pokud dokument neexistuje, bude vytvořen. Jestliže dokument existuje, bude jeho obsah přepsán nově zadanými daty. Při vytváření dokumentu pomocí funkce `setDoc()`, je třeba zadat identifikátor vytvářeného dokumentu. Daná aplikace nemá identifikátor pro dokumenty. Na to je vhodné nechat Cloud Firestore identifikátor automaticky vygenerovat, což lze provést zvolením funkce `addDoc()`. [17].

Obrázek 5.16 ukazuje příklad použití funkce `addDoc()` při přidávání nové události do databáze. Na řádku 171 se z poskytnutých informací vytvoří objekt

```
170 export const addEvent = async (title, date, time, category, description, organizer, marker) => {
171     const event = DatabaseDataFormatter.createEvent(title, date, time, category, description, organizer, marker);
172     const eventsRef = collection(database, path: 'events');
173     try {
174         const eventRef = await addDoc(eventsRef, event);
175         await updateDoc(doc(eventsRef, eventRef.id), data: {id: eventRef.id});
176         console.log('Creating new events is successed.')
177         return {
178             successed: true,
179             key: eventRef.id,
180             open: true,
181             severity: 'success',
182             text: 'Creation of a new event was successful'
183         };
184     } catch (e) {
185         console.log('Error during creating new events: ', e);
186         return {successed: false, open: true, severity: 'error', text: e.message};
187     }
188 }
```

Obrázek 5.16: `addEvent()`.

události. Na řádku 172 je vytvořen odkaz na kolekci událostí pomocí funkce `collection()`, která jako parametry přebírá odkaz na databázi a název kolekce. Na řádku 174 přijímá funkce `addDoc()` jako parametry odkaz na kolekci, do které bude přidán nový dokument, a informace o samotném dokumentu. Poté funkce vrátí odkaz na nově vytvořený dokument, ze kterého lze získat hodnotu `id`. Vrácená hodnota `id` je jedinečný identifikátor, který generuje úložiště Cloud Firestore.

Obrázek5.17 ukazuje příklad použití funkce `setDoc()`.

```
22 export const addUser = async (uid, name, surname, email, avatarImage) => {
23     const userRef = doc(database, path: 'users', uid.toString().trim());
24     const response = avatarImage ? await addUserAvatar(avatarImage) : {successed: false};
25     const avatarUrl = response.successed ? response.url : await getDefaultAvatarUrl();
26     const user = DatabaseDataFormatter.createUser(name, surname, email, uid, avatarUrl)
27     try {
28         await setDoc(userRef, user)
29         console.log('User successfully added.');
30     } catch (e) {
31         console.log('Error during adding new user: ', e.message);
32     }
33 }
```

Obrázek 5.17: `addUser()`.

Na řádku 23 je pomocí funkce `doc()` vytvořen odkaz na nový dokument. Parametry, které tam jsou předávány, jsou odkaz na databázi, název kolekce (v tomto případě název `users`) a jedinečný identifikátor dokumentu. Jako jedinečný identifikátor je použito `uid`, které bylo uživateli přiděleno při registraci a které bylo vygenerováno pomocí ověřování Firebase. Na řádku 26 je vytvořen objekt uživatele. A konečně na řádku 28 se používá funkce `setDoc()` k přiřazení objektu `userRef` s informacemi o uživateli k dokumentu.

Kromě toho, že do databáze lze přidávat nové dokumenty, je možné také měnit pole v dokumentech. K tomu slouží funkce `updateDoc()` úložiště Cloud

Firestore. Pokud dokument obsahuje pole typu pole, lze k přidávání a odebírání položek použít funkce `arrayUnion()` a `arrayRemove()`. Funkce `arrayUnion()` přidá do pole prvky, ale pouze ty, které ještě neexistují. Funkce `arrayRemove()` odstraní všechny instance každého daného prvku.

```
237 export const addCommentToEvent = async (eventId, uid, name, surname, text) => {
238     try {
239         const eventRef = doc(database, path: 'events', eventId);
240         const comment = DatabaseDataFormatter.createComment(eventId, uid, name, surname, text);
241         await updateDoc(eventRef, data: {comments: arrayUnion(comment)});
242         console.log('Comment successfully added to event.')
243     } catch (e) {
244         console.log('Error during adding comment to event: ', e.message);
245     }
246 }
```

Obrázek 5.18: `addCommentToEvent()`.

Na obrázku5.18 je příklad, kdy se pomocí funkcí `updateDoc()` a `arrayUnion()` přidává komentář do pole komentářů v dokumentu události. Na řádku 239 se zobrazí odkaz na dokument o události. Na řádku 240 se vytváří objekt komentáře a na řádku 241 se pomocí funkce `updateDoc()` a `arrayUnion()` přidává nově vytvořený komentář do pole komentářů.

Dále jsou popsány způsoby, jakými lze získat informace z databáze úložiště Cloud Firestore.

Data uložená v úložišti Cloud Firestore lze načíst dvěma způsoby [18]. Obě metody lze použít pro dokumenty, kolekce dokumentů nebo výsledky dotazů:

1. Volání metody pro získání dat.
2. Nastavení posluchače pro příjem událostí změny dat.

Pokud je nastaven posluchač, úložiště Cloud Firestore odešle posluchače a počáteční snímek dat a poté další snímek při každé změně dokumentu.

Pro jednorázové načtení dokumentu lze použít funkci `getDoc()`.

```
115 export const ifUserExist = async (uid) => {
116     const userRef = doc(database, path: 'users', uid)
117     const userSnap = await getDoc(userRef);
118     return userSnap.exists();
119 }
```

Obrázek 5.19: `ifUserExist()`.

Na obrázku 5.19 je příklad získání snímku dokumentu pomocí funkce `getDoc()`. Na řádku 116 se mi zobrazí odkaz na dokument s uživatelem. Na řádku 117 funkce `getDoc()` vrací snímek. Na řádku 118 je uveden příklad

funkce `exists()`, která je k dispozici pro každý snímek a tato funkce vrátí `true`, pokud dokument na předaném odkazu existuje, a `false` pokud neexistuje. Pro získání informací ze snímku jako objektu lze použit funkce `data()`, kterou má každý snímek.

Druhým způsobem získávání informací z databáze je nastavení posluchače dokumentu pomocí funkce `onSnapshot()`. Počáteční volání pomocí zadaného zpětného volání okamžitě vytvoří snímek dokumentu s aktuálním obsahem jednoho dokumentu. Při každé změně obsahu pak další volání aktualizuje snímek dokumentu.

```
58  useEffect( effect: () => {
59      let unsubscribe = null;
60      if (props.event.organizer && props.event.organizer !== '') {
61          unsubscribe = onSnapshot(doc(database, path: 'users', props.event.organizer), observer: snapshot => {
62              const user = snapshot.data();
63              props.setOrganizer(user);
64          })
65      }
66      return () => {
67          if (unsubscribe) unsubscribe();
68      }
69
70  }, deps: [props.event.organizer])
```

Obrázek 5.20: `onSnapshot()`.

Na obrázku 5.20 je uveden příklad, kde se funkce `onSnapshot()` používá k nastavení posluchače na dokument s informacemi o konkrétním uživateli. Na řádku 61 předávám první parametr funkci `onSnapshot()` pro odkaz na dokument pomocí funkce `doc()` a druhý parametr pro předání funkce, která obdrží snímek při každé změně v dokumentu a zpracuje jej. Na řádku 67 se píše, že při demontáži komponenty bude posluchač zavolán pomocí funkce `onSnapshot()`, aby byl deaktivován.

Úložiště Cloud Firestore zajišťuje výkonnost dotazů tím, že pro každý dotaz vyžaduje index. Indexy potřebné pro nejjednodušší dotazy se vytvářejí automaticky. Při pokusu spustit složený dotaz s návrhem rozsahu, který neodpovídá existujícímu indexu, se zobrazí v konzole prohlížeče chybová zpráva. Chybová zpráva5.21 obsahuje přímý odkaz na vytvoření chybějícího indexu v konzole Firebase.

```
Error during getting events:  The query requires an index. You can create it databaseUtils.js:212
here: https://console.firebase.google.com/v1/r/project/meet-me-now-9d6b6/firestor…MvZXZlbnRzL2luZG
V4ZXMvXxABGgwKCGNhdGVnb3J5EAEaBgoCaWQQARoMCghfX25hbWVfXxAB
```

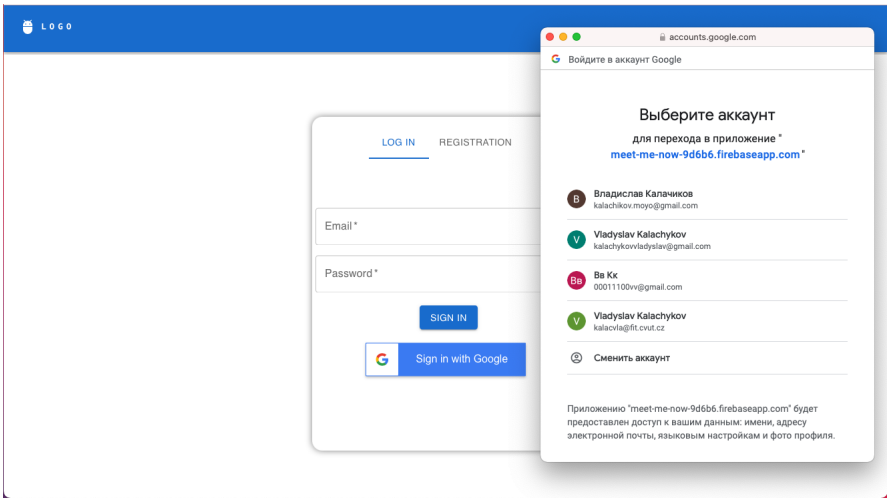
Obrázek 5.21: Ukázka chyby při indexace.

5.2.6.2 Firebase Authentication

Služba **Firebase Authentication** poskytuje serverové služby pro ověřování uživatelů v aplikaci. Podporuje ověřování pomocí hesel, telefonních čísel, populárních poskytovatelů federativních pověření, jako jsou Google, Facebook,

Twitter atd. V dané aplikaci byla uživatelům nabídnuta možnost registrace a přihlášení pomocí uživatelského jména a heslastejně jako přihlášení pomocí služby Google. Aby bylo možné zaregistrovat účet s e-mailem a heslem, nabízí Firebase Authentication funkce `createUserWithEmailAndPassword()`, které stačí předat e-mail a heslo. Funkce rovněž nabízí možnost ověření dat, například, pokud je heslo použité při registraci slabé (jako *"1111"* nebo *"qwerty"*), vygeneruje chybovou zprávu, která na to upozorní a zabrání uživateli vytvořit účet s takovým heslem. Funkce `signInWithEmailAndPassword()` potřebuje k přihlášení stejné parametry.

Aby se uživatel přihlásil pomocí účtu třetí strany, tak se používá funkce `signInWithPopup()`, kde se jako parametr poskytovatele předává objekt vytvořený příkazem new `GoogleAuthProvider()`. Poté se otevře vyskakovací okno, ve kterém si uživatel zvolí svůj preferovaný účet Google tak, jak je znázorněno na obrázku 5.22.



Obrázek 5.22: Google Authentication.

5.2.6.3 Firebase storage

Pro místo ochrany profilových fotografií uživatelů bylo zvoleno úložiště **Firebase Storage**. Na obrázku 5.23 je zobrazena implementace ukládaní fotografií.

Nejprve je třeba získat odkaz na místo, kam se fotografie uloží (jak je uvedeno na řádku 36). Poté se používá funkce `uploadBytes()`, která jako parametry přijímá odkaz na místo pro uložení, samotný soubor a volitelně také objekt s metadaty nahraného souboru. V daném případě je v metadatech ukázáno, že ukládaný objekt bude obrázek ve formátu jpg. Po nahrání obrázku lze pomocí funkce `getDownloadURL()` získat odkaz na uložený soubor. Tento

```
35  export const addUserAvatar = async (image) => {
36      const imageRef = ref(storage,  url: `avatars/${uuid()}`);
37      try {
38          await uploadBytes(imageRef, image,  metadata: {contentType: 'image/jpg'})
39          const url = await getDownloadURL(imageRef, image);
40          return {successed: true, url: url}
41      } catch (e) {
42          console.log('Error during uploading avatar to storage: ', e.message)
43          return {successed: false};
44      }
45  }
```

Obrázek 5.23: `addUserAvatar()`.

odkaz se přidává do databáze uživatele, aby jej bylo možné v případě potřeby použít.

Kapitola 6

Testování

Třetí fází tvorby webové aplikace je provedení vzdáleného uživatelského testování, jehož účelem je odhalit problémy s použitelností a stanovit další body pro zlepšování a rozvoj navržené aplikace. Optimální doba pro uživatelské testování je 2-3 hodiny. Příprava vzdáleného testování se skládá z následujících položek:

1. Vytvořte scénář uživatelského testování:
   a) Provést registraci:
      - Zadejte údaje.
      - Nahrát fotografii.
      - Odhlášení z účtu
      - Přihlásit se na webovou stránku pomocí vytvořeného účtu.
   b) Vytvořte si vlastní událost:
      - Zítra v 18:00 se bude konat večírek.
      - Zvolte umístění.
      - Zkontrolujte své údaje.
   c) Najděte svůj večírek podle místa:
      - Otevření mapy.
      - Vybrat událost.
      - Zanechat komentář.
   d) Seřadit vyhledávání událostí:
      - Otevřít nabídku.
      - Vyberte kategorie událostí.
      - Obnovení seznamu událostí.
2. Vyhledávání potenciálních uživatelů:

a) Výsledky testování přímo závisí na výběru respondentů. Cílovou skupinou testované aplikace jsou lidé neomezeného věku se základními počítačovými dovednostmi a přístupem k internetu. Podle statistik na obrázku 6.1 se 70 % chyb dá zjistit v důsledku testování již tří respondentů.

Obrázek 6.1: Statistika najezděných chyb podle poctu reduantu [19].

3. Definice zkušebních podmínek.

   a) Na testovací prostředí nejsou kladeny žádné přísné požadavky: uživatelé si mohli vyzkoušet tuto webovou aplikaci na veřejném místě jako je kancelář nebo kavárna, stejně jako na soukromém místě.

4. Instalace softwaru "Discord"

   a) Během testu uživatelé používali osobní počítač s předinstalovaným **Discord**. Respondenti měli zapnuté webkamery a mikrofon - to umožnilo obhájit emoce uživatele a okamžitou reakci na prováděný procesu, případně na problémy, kterým uživatel čelí. Testování bylo nahráváno pro analýzu chování uživatelů během plnění úkolů a stanovení dalšího závěru.

Na základě výsledků výzkumné fáze byly po ukončení plnění úkolu respondentům položeny následující otázky:

- Podařilo se Vám splnit úkol scénáře?

- Co se Vám líbilo nebo nelíbilo?
- Čim byste připadne doplnili nebo naopak by se z aplikace odstranili?

Pokud jste v některém kroku zmateni, podrobně to popište.

"Podařilo-li se Vám splnit ukol?"

Navzdory tomu, že se všem podařilo splnit úkoly popsané ve scénáři, pro 3 ze 6 účastníků byly určité body ve scénáři matoucí. Účastníci se na stránky bez potíží zaregistrovali, vyplnili údaje o vytvářené události, otevřeli mapu s již existujícími událostmi a zanechávali komentáře k vybrané události. Potíže však nastaly při vytváření vlastní události a odstranění filtru vyhledávání událostí. Zvládnutí funkčnosti a pochopení designu při výběru těchto propagačních akcí některým účastníkům zabralo více času, než byl odhadovaný čas, a upadli do rozpaků.

"Co se Vám líbilo nebo nelíbilo?"

Výhodami aplikace účastníci zaznamenali rychlost vyhledávání a viditelnost stávajících akcí v blízkosti místa, jednoduchost návrhu při pohybu po aplikaci a výběru potřebných sekcí a také absenci matoucích prvků v návrhu, které předtím neviděli, nebo by v nich vyvolaly zmatek.

Nedostatky vytvořené aplikace podle účastníků byly následující: nepohodlnost použití tlačítka *„+“* při vytváření vlastní události (uživatel očekává, že se ihned po kliknutí na „+“ objeví okno pro vytvoření události), problém nahrání fotografie uživatele pro jednoho z účastníků při registraci, nepochopení/nedorozumění ze strany některých účastníků testu funkčnosti při výběru a mazání filtrů.

"Čim byste připadne doplnili nebo naopak by se z aplikace odstranili?"

Někteří z účastníků poznamenali, že při vytváření události jim nestačilo krátkého úvodu o tom, co se musí dělat při zobrazovaní mapy a jak pomoci ní zvolit místo konání události. Jeden z účastníků se domníval, že by bylo dobré mít osobní účet, kde by uživatel mohl měnit své údaje. 50% účastníků by uvítalo samostatné tlačítko, které by mazalo vybraný filtr, a tlačítko, které by odstranilo všechny filtry najednou. Pro většinu účastníků bylo tlačítko s ikonou tužky pro tvorbu nové události zbytečné a uvítali by, kdyby se ihned objevilo okno s poli pro zadání údajů o události. V neposlední řadě je důležité přidat odpověď na komentář, který v aplikaci zanechal jiný uživatel.

Po analýze záznamů z procesu testování a zpracování zpětné vazby od účastníků testu byla stanovené následující závěry:

1. Mělo by být odstraněno tlačítko *"+"* pro vytvoření vlastní události a přidat *"Uložit událost"*, po jehož kliknutí se okamžitě zobrazí okno pro vyplnění informace o události.
2. Pro zlepšení použitelnosti aplikace je potřeba přidat textový doprovod při vytváření vlastní akce.
3. Počátečním oknem při první návštěvě stránky je nutné, aby byl nejprve registrační formulář a ne přihlašovací okénko.
4. V sekci pro přidávání filtrů je potřeba změnit UX design na uživatelsky známější. Vytvořit například rozevírací seznam v horní části, kde může uživatel zaškrtnout požadované filtry a při mazání filtru zrušit zaškrtnutí, a přidat tlačítko *"Odstranit všechny filtry"*, aby urychlil uživateli práci s aplikací.
5. Při dalším rozvoji aplikace lze přidat osobní účet, kde může uživatel upravovat své informace.
6. Přidat možnost reagovat na komentáře ostatních uživatelů.

Kapitola 7

Závěr

Účelem této práce bylo navrhnout a implementovat funkční prototyp jednostránkové webové aplikace, cílem které by bylo usnadnit uživateli hledaní událostí ve své blízkosti, a to nejen v rozsahu specifického města. V rámci realizace bakalářské práce byl tento cíl splněn pomocí provedení analýzy existujících, stávajících řešení. Na základě jejich nedostatků byl sestaven seznam funkčních a nefunkčních požadavků. Pro realizaci všech zamýšlených funkcí aplikace byly vybrány vhodné technologie, jako jsou knihovna *React* pro psaní jednostránkové webové aplikace, databáze *Firebase Firestore* pro ukládání informací o uživatelích a událostech, *Firebase Storage* pro ukládání fotografie uživatele a *Firebase Authentication*. K implementaci mapy, která uživateli umožňuje snadno a rychle vyhledávat události kolem sebe, slouží *Google Maps API*. S pomocí promyšleného uživatelského rozhraní a výše uvedených technologií poskytuje aplikace uživatelům možnost: registrovat se do aplikace pomocí mailu a hesla a také přidat jedinečnost účtu nahráním fotografie jako uživatelského avatara, přihlášení do aplikace pomocí účtu Google, vytváření vlastních událostí, vyhledávání událostí v okolí, ukládání do seznamu oblíbených událostí, vyjádření svých názorů pomocí funkce zanechaní komentáře a také hodnocení organizátora události.

V závěru psaní aplikace proběhlo uživatelské testování, jehož výsledky určily cíle dalšího rozvoje a zlepšování schopností, kterými aplikace již disponuje. V budoucnu lze k aplikaci přidat funkce:

1. Uživatelský účet, kde bude uživatel moci měnit údaje o sobě uvedené při registraci.

2. Možnost mít jedinečné uživatelské jméno, které se zobrazí, pokud uživatel nechce ukazovat své skutečné jméno a příjmení.

3. Odpověď na konkrétní komentář v události tak, aby uživatel pochopil, že se text týká konkrétně jeho.

Zdroje

[1] Statista. Number of daily active Facebook users worldwide as of 1st quarter 2022 [online]. April 2022 [cit. 08.06.2022]. Dostupné z: https://www.statista.com/statistics/346167/facebook-global-dau/

[2] StackOverflow. % of Stack Overflow questions that month [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cember.js%2Cmeteor%2Cknockout.js

[3] NPM. Angular vs Knockout vs React vs Vue [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://www.npmtrends.com/angular-vs-knockout-vs-react-vs-vue

[4] tianzhou. GitHub star history of open source projects [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://star-history.com/#facebook/react&vuejs/vue&angular/angular&Date

[5] Google. Job trends [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://trends.google.com/trends/explore?cat=31&q=Vue%20jobs,React%20jobs,Angular%20jobs

[6] NPM. Mobx vs Overmind vs Redux [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://www.npmtrends.com/mobx-vs-overmind-vs-redux

[7] Google. Cloud Firebase Authentication [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/auth

[8] Google. Firebase Realtime Database [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/database

[9] Google. Cloud Firestore [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/firestore

[10] Google. Choose a Database: Cloud Firestore or Realtime Database [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/database/rtdb-vs-firestore

[11] Lyakhov, A. Introduction to React Google Maps [online]. January 2021 [cit. 14.06.2022]. Dostupné z: https://react-google-maps-api-docs.netlify.app/

[12] Google. Google Maps Platform Billing [online]. May 2022 [cit. 14.06.2022]. Dostupné z: https://developers.google.com/maps/billing-and-pricing/billing

[13] ReactJS. Lists and keys [online]. July 2019 [cit. 14.06.2022]. Dostupné z: https://ru.reactjs.org/docs/lists-and-keys.html

[14] Gillis, A. S. UUID (Universal Unique Identifier) [online]. August 2021 [cit. 14.06.2022]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier#:~:text=A%20UUID%20(Universal%20Unique%20Identifier,UUID%20generated%20until%20A.D.%203400

[15] Mozilla. Using the Geolocation API [online]. May 2022 [cit. 14.06.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API/Using_the_Geolocation_API

[16] Mozilla. Clipboard API [online]. February 2022 [cit. 14.06.2022]. Dostupné z: https://developer.mozilla.org/ru/docs/Web/API/Clipboard_API

[17] Google. Add data to Cloud Firestore [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/firestore/manage-data/add-data

[18] Google. Get data with Cloud Firestore [online]. June 2022 [cit. 14.06.2022]. Dostupné z: https://firebase.google.com/docs/firestore/query-data/get-data

[19] Nielsen, J. Why You Only Need to Test with 5 Users [online]. March 2000 [cit. 14.06.2022]. Dostupné z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Příloha A

Seznam zkratek

MPA Multiple Page Application

SPA Single Page Application

IČO Identifikační číslo fyzické nebo právnické osoby.

HTML HyperText Markup Language

URL Uniform Resource Locator

AJAX Asynchronous JavaScript and XML

JS JavaScript

API Application Programming Interface

MVP Model-View-Presenter

CLI Command Line Interface

DOM Document Object Model

MVC Model-View-Controller

SQL Structured Query Language

BSON Binary JSON

JSON JavaScript Object Notation

UI User Interface

UX User Experience

Příloha **B**

Obsah přiloženého média

`thesis.pdf` ..... text práce ve formátu PDF
`thesis` ..... zdrojová forma práce ve formátu LaTeX