FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webová aplikace pro usnadnění vyhledávání událostí
Student:	Radek Čermák
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webovou aplikaci, která bude agregovat kulturní a jiné zájmové události. Tyto události budou moci uživatelé procházet, ukládat si zájem o navštívení atd. Pro implementaci využijte některý z reaktivních JavaScriptových frameworků. Dalším důležitým cílem této práce je zanalyzovat a případně propojit různé externí zdroje dat ze kterých je možno události získávat.

Ideálně postupujte dle následujících kroků:

1) Proveďte rešerši konkurenčních řešení
2) Specifikujte funkční a nefunkční požadavky aplikace
3) Proveďte návrh uživatelského rozhraní a samotné aplikace
4) Implementujte prototyp aplikace
5) Proveďte uživatelské testování aplikace a navrhněte možné úpravy aplikace nebo je rovnou realizujte
6) Implementujte získávání dat z externích zdrojů, pokud to je dle předešlé analýzy možné

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 29. ledna 2024 v Praze.

Bakalářská práce

WEBOVÁ APLIKACE PRO USNADNĚNÍ VYHLEDÁVÁNÍ UDÁLOSTÍ

Radek Čermák

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Filip Glazar
16. května 2024

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat vedoucímu práce Ing. Filipovi Glazarovi za ochotu, cenné rady a přátelský přístup při vedení této práce. Také bych chtěl poděkovat rodině a přítelkyni za podporu po celou dobu studia. V neposlední řadě děkuji účastníkům testování za ochotu a trpělivost.

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.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zabývá návrhem a následnou implementací webové aplikace, jejíž cílem je usnadnit vyhledávání kulturních, společenských či jinak zajímavých událostí v okolí. Aplikace umožňuje vizualizovat události na mapě, ve které lze události prohlížet, filtrovat nebo vyhledávat s pomocí našeptávače. V rámci analýzy byla provedena rešerše konkurenčních řešení a zhodnocení jejich nedostatků. Následně byly podle získaných informací definovány požadavky na aplikaci. V kapitole věnované návrhu byl rozebrán postup při tvorbě uživatelského rozhraní, byl vytvořen databázový model a byly popsány technologie použité k implementaci řešení. Do kapitoly o implementaci byly vloženy nejdůležitější části z procesu implementace. Na závěr byla aplikace uživatelsky otestována. Z výsledků byly vyhodnoceny úpravy, které byly následně implementovány.

Klíčová slova události, agregace událostí, webová aplikace, Maps JavaScript API, React, Next.js, Crawlee

Abstract

This bachelor thesis focuses on the design and implementation of a web application, which helps with finding culture, social or other events nearby. The application provides visualization of events on the map, that allows browsing events, filtering results or searching events with autocomplete option. The analysis was carried out by research on existing solutions and evaluation of their cons. Then, according to the information obtained, functional and nonfunctional requirements were defined. The chapter dedicated to design contains dissasembled process of user interface design, database model creation and description of used technologies for implementation of the solution. The chapter about implementation includes the most important parts of this process. Lastly, the web application was user tested. The results of testing were used for finding improvements, that were later implemented.

Keywords events, event aggregation, web application, Maps JavaScript API, React, Next.js, Crawlee

Seznam zkratek

API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
JSX	JavaScript XML
NoSQL	Not only SQL
ORM	Object Relational Mapping
SQL	Structured Query Language
SSR	Server-Side Rendering
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
XML	Extensible Markup Language

Úvod

Rozhodl jsem se vyjít si ven, navštívit novou expozici nebo se zúčastnit promítání premiéry. K nalezení konkrétní akce nejspíše využiji jednu z dostupných aplikací nebo webových stránek, které se na shromažďování událostí zaměřují. Většina z nich již nabízí širokou škálu filtrování, např. podle typu události nebo času konání. Problém nastává, jakmile chci zobrazit akce konané v mém okolí. Některé aplikace sice nabízí seřazení událostí podle vzdálenosti, ovšem bez vizualizace na mapě a mnohdy i chybně. Tento nedostatek může mít pro uživatele za následek složitější určení přesného místa konání a tím i náročnější vybírání. Osobně je pro mě používání těchto aplikací frustrující, jelikož při vyhledávání akce v pro mně neznámé lokaci nedokáži odhadnout vzdálenost pouze za pomoci adresy a musím proto složitě vyhledávat událost na mapě v jiné aplikaci.

Z výše uvedených důvodů jsem si zvolil za cíl bakalářské práce navrhnout a následně implementovat webovou aplikaci pro vyhledávání kulturních a společenských akcí v okolí tak, aby vyřešila nedostatky existujících řešení a udělala pro uživatele vyhledávání událostí co nejjednodušší. Proto hlavní vlastnost aplikace představuje vizualizace všech událostí na mapě, kterou bude možné prohlížet. Pokud uživatel povolí přístup aplikaci k poloze, mapa se automaticky přemístí na lokaci uživatele a zobrazí nejbližší okolí. Další důležitou vlastností je propojení několika externích zdrojů dat, ze kterých aplikace čerpá informace o událostech. Díky tomu uživatelé nemusí prohledávat více různých webů a budou mít maximální přehled o dění kolem.

Kapitola 1

Analýza

Jako první krok analýzy provedu rešerši aktuálních řešení, což je důležité pro správné stanovení funkčních a nefunkčních požadavků. Aplikace by totiž měla obsahovat funkce, na které jsou uživatelé zvyklí, a zároveň řešení problémů vznikajících u konkurenčních webů a aplikací. Součástí rešerše je zhodnocení jednotlivých aplikací, zda jsou vhodné jako externí zdroj dat, a uvedu potřebné kroky k jejímu propojení. Následně vzniklé požadavky definuji.

1.1 Rešerše aktuálních řešení

Jak jsem již zmiňoval v úvodu, alternativ na vyhledávání událostí je opravdu mnoho. Existují jak weby zaměřující se jen na Českou republiku, tak i aplikace obsahující akce z celého světa. Já se zaměřil na tři konkurenční řešení, které jsou za mě v rámci České republiky nejpopulárnější.

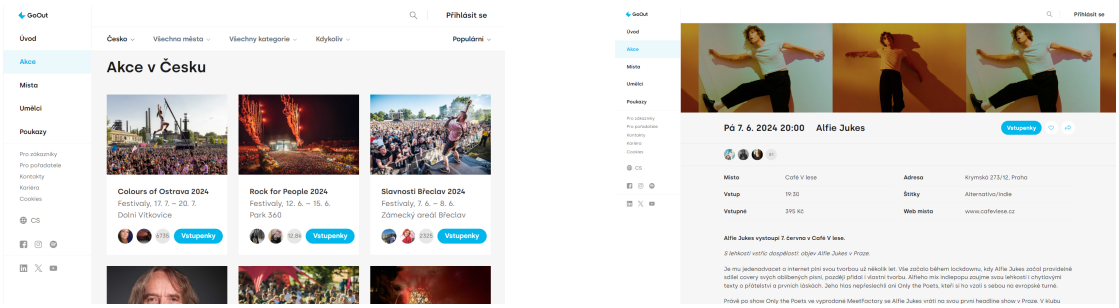
1.1.1 GoOut

Firma GoOut se zaměřuje na online prodávání vstupenek. V rámci této služby provozuje webovou aplikaci[1], kde se nachází všechny události, které zaštiťují. Vyhledávat mezi akcemi lze za pomocí názvu akce, místa nebo umělce. Samotné možnosti filtrace jsou pak podle země, města, času konání a kategorie. V roce 2019 se GoOut s jejich webem umístil v soutěži Křišťálová Lupa na 4. místě v kategorii „Nástroje a služby“, pomocí kterého dnes prodají okolo 300 000 vstupenek každý měsíc. [1, 2]

Na jejich aplikaci se mi líbí jednoduchý a minimalistický design obsahující černobílou paletu barev spolu s jejich charakteristickou světle modrou. Minimalistický vzhled je rozhodně vlastnost aplikace, kterou bych chtěl zahrnout do svého řešení. Pokud se zaměřím na vyhledávání v okolí, jediná možnost je hledat podle města. To považuji za nedostačující, jelikož při velkém městě může být výsledek mnohem širší, než bychom chtěli. V situaci, kdy se ve městě nenachází žádná akce, aplikace zobrazí alternativní místa blízko původnímu. Toto řešení, opět bez předchozí znalosti okolí, rozhodování neulehčuje dostatečně.

Klient aplikace nejdříve načte základní kostru stránky, a až poté se pošle požadavek o zaslání dat. Aby toto bylo možné, URL požadavku musí být veřejná. Díky tomuto chování mohu využít danou adresu jako externí zdroj dat.

[1] Dostupná na: `https://goout.net`



(a) Výpis akcí **(b)** Detail akce

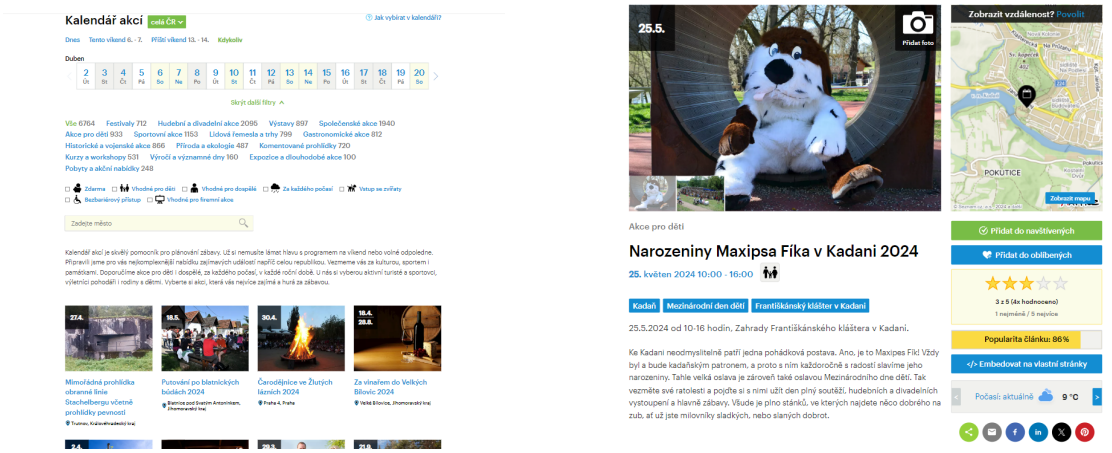
Obrázek 1.1 Náhled webu goout.net [3]

1.1.2 Kudy z nudy

Portál kudyznudy.cz je projektem agentury CzechTourism, která vznikla pro podporu domácího cestovního ruchu. Nabízí komplexní služby v oblasti cestování, ubytování a zážitkové turistiky. V hlavní turistické sezoně stránky zaznamenávají měsíčně přes 4 miliony návštěv. V roce 2023 web zaznamenal 12 milionů uživatelů, 28,6 milionů návštěv a přes 251 milionů událostí. [4]

Stránka s kalendářem akcí je přehledná, design má ovšem konzervativnější než aplikace firmy GoOut. Filtrace je zde podobná jako u předešlého webu – obsahuje možnosti filtrace podle času konání, kategorie a města. Navíc ale umožňuje filtrovat podle vlastností události, jako např. „vhodné pro děti“, „bezbariérový přístup“, „zdarma“, …Vyhledávání událostí v okolí je opět možné jen pomocí zadání města. Web kudyznudy.cz sice po otevření detailu akce zobrazí lokaci na mapě, ale to už vyžaduje kliknutí uživatele na každou zamýšlenou akci, což určitě jeho rozhodování neulehčuje.

Na rozdíl od předchozího příkladu tato aplikace posílá rovnou stránku i se všemi daty. To má za následek, že nelze získat přímý přístup k datům. Způsob, jakým získat data, může být poslat požadavek o HTML soubor stránky a data z něho extrahovat. Díky této akci mohu i tuhle aplikaci využít jako externí zdroj dat.



(a) Výpis akcí **(b)** Detail akce

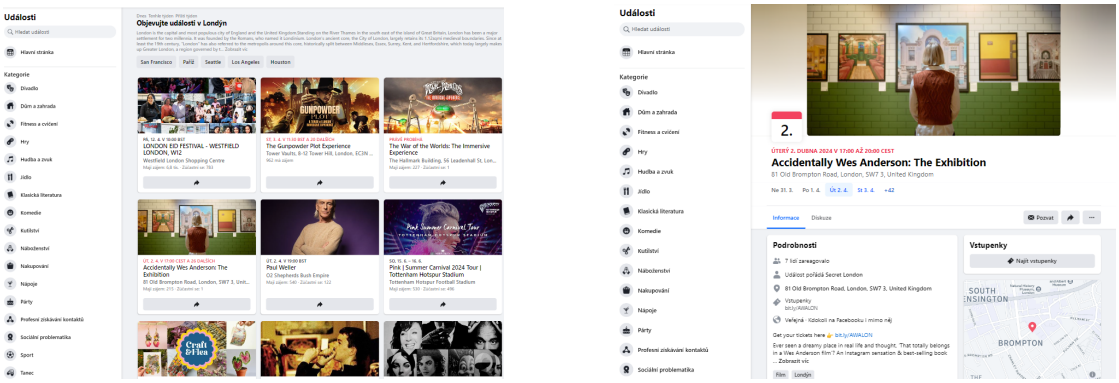
Obrázek 1.2 Náhled webu kudyznudy.cz [5]

1.1.3 Facebook Events

Facebook Events[2] je jedna ze služeb nabízená aplikací Facebook, která umožňuje snadné a rychlé vytvoření jakékoliv události a umístění jej rovnou na jejich sociální síť. Uživatelé mohou události filtrovat podle různých kritérií, jako jsou datum, místo nebo typ události. Díky propojení se sociální sítí Facebook je zde několik funkcí navíc, které předchozí zmiňované aplikace neměly. Jedná se na příklad o zobrazení akcí, ke kterým byli uživatelé pozváni nebo které události navštíví jejich přátelé. Dále s nimi mohou interagovat pomocí projevení zájmu, komentování nebo sdílením. Facebook v říjnu 2023 pokořil 3 miliardy aktivních uživatelů a stal se tak nejaktivnější sociální sítí. [6]

Pokud bych chtěl na stránce Facebook Events vyhledávat akce v okolí, mohu využít nastavení seřazení podle „Moje poloha“. Bohužel výsledky jsou dost diskutabilní. Momentálně se nacházím v Praze a první dvě akce, které by měly být nejblíže, jsou v Nelahozevsi a Roztokách u Prahy. Až poté jsou vypsány akce v Praze. Přijde mi zvláštní, že vyhledání nepožadovalo přístup k mé poloze a ani se mi nikde nepodařilo zjistit, podle jaké polohy vyhledávání probíhá.

Jakmile nejsem přihlášen pod uživatelským účtem, Facebook ukazuje velmi omezený výpis akcí. Abych mohl využít Facebook jako externí zdroj dat, je zapotřebí být přitom přihlášen. Facebook má ovšem velmi striktní pravidla pro zacházení s jejich vlastními daty a bylo by tak časově náročnější tyto pravidla obejít. Proto je pro mě Facebook jako externí zdroj dat nevhodný.



(a) Výpis akcí **(b)** Detail akce

Obrázek 1.3 Náhled webu facebook.com/events [7]

1.1.4 Shrnutí

Z provedené rešerše jsem zjistil, že žádná z existujících aplikací neřeší zobrazení nejbližších událostí v okolí efektivně. Zkoumané aplikace sice obsahují vyhledávání podle města, ale to nepovažuji za dostatečné. Zdánlivě nejlépe na tom byla aplikace Facebook Events, která nabízela seřazení událostí podle vzdálenosti od mé polohy. Bohužel při mém testování toto seřazení vůbec nefungovalo a nabízelo mi první akce, které ode mě byly dál, než akce seřazené až na dalších pozicích. Z tohoto důvodu je i toto řešení dle mého názoru nedostatečné. Proto jsem usoudil, že vytvořit vlastní aplikaci má smysl a lidem by tak mohlo usnadnit výběr při hledání události.

[2] Dostupná na: `https://www.facebook.com/events`

1.2 Funkční požadavky

Funkční požadavky jsou součástí procesu analýzy požadavků. Popisují cílové chování aplikace, které musí splňovat. Tyto požadavky jsou většinou odvozeny z potřeb uživatelů, nároků klienta nebo celkového cíle aplikace. [8]

Na základě rešerše konkurenčních řešení a vlastních požadavků byly stanoveny následující funkční požadavky:

F1 – Zobrazení událostí Všechny události budou vizualizovány na mapě. Mapu bude možné posouvat, přibližovat, oddalovat, …Místo konání akce bude označeno značkou, která bude rozlišená unikátní barvou a ikonkou podle kategorie dané akce. Při více událostech na jednom místě bude lokace označena značkou s počtem shluknutých událostí. Značka pro tyto události se také objeví, jakmile bude více míst příliš blízko u sebe. Ta se při přiblížení následně rozdělí.

F2 – Zobrazení informací Aplikace bude umožňovat zobrazení základních informací o události. Při prvním kliknutí na událost se uživateli ukáže stručný popis a nejdůležitější potřebné informace. Při projeveném zájmu bude možné otevřít kompletní detail se všemi informacemi. Mezi tyto informace patří především název, místo konání, čas konání, kategorie události a popis události.

F3 – Lokalizace uživatele Uživatel bude mít možnost použít geolokaci k rychlému nalezení své polohy. Jakmile aplikace dostane přístup k jeho poloze, mapa se přesměruje na polohu uživatele a zobrazí nejbližší události.

F4 – Odkaz na původní stránku události U každé události bude možné dostat se na původní stránku události, ze které byly informace získány.

F5 – Filtrace událostí Aplikace bude umožňovat filtrovat události podle kategorie. Těchto kategorií půjde k vyfiltrování zvolit více. Dále bude možné vybrat možnost zobrazení akcí, které probíhají právě teď.

F6 – Vyhledávání míst V aplikaci bude uživateli umožněno vyhledávat podle názvu místa pořádání události. Aplikace poté toto místo a jeho okolí ukáže na mapě. Během zadávání se bude zobrazovat našeptávání možných názvů, které se automaticky po kliknutí vyhledají. Zadání místa, které v aplikaci není uloženo, bude náležitě ošetřeno.

F7 – Periodická aktualizace událostí Nové události se z externích zdrojů budou přidávat jednou denně. To zahrnuje i přidávání nových míst. Záměrem tohoto požadavku je nepřetěžovat externí zdroje příliš častými dotazy.

1.3 Nefunkční požadavky

Nefunkční požadavky jsou také součástí analýzy požadavků. Na rozdíl od funkčních požadavků nepopisují chování, ale definují vlastnosti a omezení aplikace jako celku. Patří mezi ně na příklad bezpečnostní požadavky, nároky na kvalitu a udržitelnost implementace, výkonnostní požadavky, legislativní omezení, …[8]

N1 – Responzivní design Aplikaci bude možné používat na různě velkých zařízeních. Proto je zapotřebí, aby se uživatelské rozhraní bylo schopné přizpůsobit na různé velikosti obrazovky. Důležité bude zachování všech podstatných ovládacích prvků bez újmy na přehlednosti vzhledu i na malých zařízeních.

N2 – Rozšiřitelnost pro další externí zdroje Aplikace bude umožňovat jednoduché přidání dalších externích zdrojů dat. Při implementaci tohoto požadavku bude kladen velký důraz na udržitelnost a znovupoužitelnost kódu.

N3 – Aplikace pro získávání dat Získávání dat z externích zdrojů se bude implementovat jako samostatná aplikace. Nebude tedy součástí hlavní aplikace.

N4 – Veřejný přístup Jelikož se jedná o webovou aplikaci, uživatel musí být schopný aplikaci používat jen za pomocí webového prohlížeče. Proto aplikace bude nasazena na webový server, ze kterého bude veřejně přístupná.

N5 – Technologie Obě aplikace budou napsané v jazyce TypeScript. Pro implementaci hlavní aplikace bude použitý některý z full stack[3] frameworků. Pro implementaci aplikace na získávání dat bude použita vhodná knihovna.

[3]Umožňuje implementaci jak front end části tak back end části.

Kapitola 2
Návrh

Tato kapitola se věnuje návrhu mého řešení. Jako první rozeberu postup při tvorbě uživatelského rozhraní. Následně přiblížím proces výběru databáze a vytvoření modelu databáze. Na závěr objasňuji použité technologie.

2.1 Návrh uživatelského rozhraní

Návrhu uživatelského rozhraní jsem věnoval nejvíce času, jelikož to považuji za nejdůležitější část tohoto procesu. Uživatelské rozhraní (neboli UI) je hlavním prvkem komunikace mezi uživatelem a mou aplikací, je proto žádoucí, aby v uživateli vyvolala pozitivní zážitek. Návrh aplikace pro co nejlepší uživatelský zážitek (neboli UX) je nedílnou součástí návrhu UI a pouze kombinací těchto dvou praktik dokážu vytvořit uživatelské rozhraní, které se snadno ovládá, je přehledné a v neposlední řadě se uživatelům líbí. Při návrhu UX je dobré se držet osvědčených principů.

Já jsem dodržoval tato pravidla [9]:

Zaměření na uživatele – Hlavní pravidlo UX návrhu. Uživatelské rozhraní musí být navrženo podle uživatelských potřeb a s cílem vyřešit jejich problémy.

Konzistence – Toto pravidlo hovoří o udržení konzistence nejen v rámci aplikace, ale také o konzistenci vůči aplikacím se stejným zaměřením. Konzistence aplikace znamená, že vzhled a funkce budou v rámci celé aplikace zachovány. Konzistence napříč jinými aplikacemi znamená, že uživatelské rozhraní bude splňovat očekávání uživatelů na základě zkušeností s podobnými aplikacemi.

Hierarchie informací – Nejdůležitější informace musí být pro uživatele dostupné jako první a musí být ihned viditelné.

Zaměření na cílovou skupinu – Při návrhu UX na podmínky a možnosti uživatele při používání aplikace. To může být např. jaké zařízení používá, kde se nachází nebo v jaké je zrovna situaci.

Plná kontrola uživatele nad aplikací – Jakmile uživatel provede chybnou akci, je potřeba mu poskytnout jasně označenou cestu zpět.

Přístupnost – Ovládaní aplikace musí být přístupné pro co nejširší spektrum uživatelů. Při návrhu UX musí být bráno v potaz možné postižení uživatele nebo jiné podmínky, které mu znesnadňují užívání aplikace.

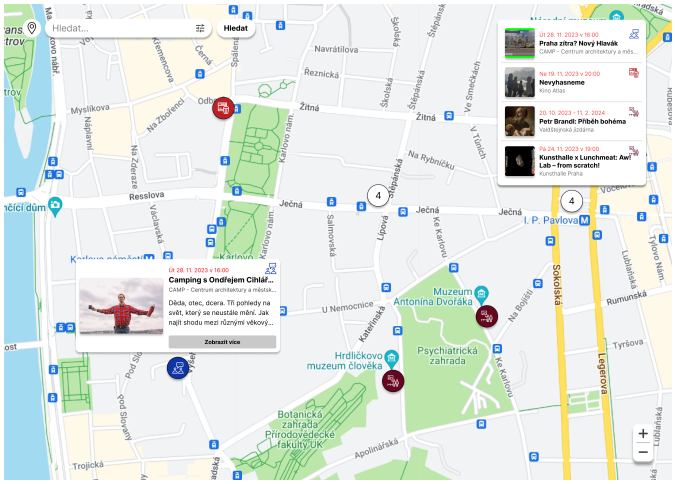
Použitelnost – Uživatelské rozhraní musí být navrhnuto za účelem co nejjednoduššího používání. V této metrice je obsaženo, jak rychle se uživatel ovládání naučí, jak rychle dosáhne svého cíle, jak snadno si ovládání zapamatuje, jak často uživatel udělá chybu a jak moc je uživatel s ovládáním spokojen.

U zobrazení událostí na mapě jsem si bral inspiraci z aplikace `scuk.cz` a jejich výběru výdejny. Pro hierarchii informací o události jsem se inspiroval v aplikaci `facebook.com/events`. Návrh jsem prováděl v nástroji Figma, který nabízí širokou škálu možností a podrobné nastavení vzhledu prvků.

Paletu barev jsem zvolil černobílou. Díky tomu bude mít aplikace moderní vzhled a zároveň nebudou na uživatele barvy působit rušivě. Jediné barevné prvky zde budou sloužit k označení času konání a kategorie události. Slibuji si od toho, že barevný kontrast bude vyvolávat uživatelovu pozornost a dokáže tak rychle zaznamenat čas konání a rozlišit jednotlivé kategorie. Ukázku značek událostí můžete vidět na obrázku 2.1.

Obrázek 2.1 Značky událostí

Při navržení souhrnu události bylo potřeba udržet kompaktní vzhled bez újmy na přehlednosti. Proto souhrn obsahuje pouze ty nejdůležitější informace s možností dozvědět se více v detailu události. Vzhled detailu události jsem chtěl dodržet co nejvíce konzistentní se vzhledem souhrnu události. Proto jsem u většiny informací zachoval stejnou hierarchii a přidal jsem navíc zbytek chybějících informací. Jakmile je na místě více událostí, k jejich zobrazení bude sloužit výpis událostí. Ten nabízí k popisu události nejméně místa, a tak zde budou obsaženy pouze nezbytně nutné informace. Na obrázku 2.2 lze vidět výsledný návrh uživatelského rozhraní.



Obrázek 2.2 Návrh uživatelského rozhraní

2.2 Databáze

Databáze je důležitou částí mého řešení, jelikož obsah aplikace bude silně závislý na zobrazování získaných dat, které je potřeba někde uchovat. V následujících dvou podkapitolách nejdříve popíšu výběr databáze a objasním důvody mého rozhodnutí. Následně uvedu popis navrženého modelu databáze. Na závěr zmíním, kde provedu nasazení databáze.

2.2.1 Výběr databáze

Na výběr mám mezi klasickými relačními nebo NoSQL databázemi. Relační databáze ukládá data ve formě tabulek, ve kterých slouží sloupce jako atributy a řádky jako jednotlivé záznamy. Tabulky lze vzájemně propojit, čímž dokážeme reprezentovat vztahy mezi entitami. Relační databáze má pevně dané schéma a tak se hodí na reprezentaci strukturovaných dat, u kterých dopředu známe vzájemné vazby. Naopak NoSQL databáze dovolují ukládat různě strukturovaná data díky dynamickému schématu, které lze jednoduše modifikovat. Díky vysoké flexibilitě se proto hodí na ukládání dat s předem nedefinovanou strukturou nebo při příliš velkém množství dat pro rozumnou reprezentaci v relačním modelu. Nevýhodou tohoto řešení může být inkonzistence výsledků z důvodu distribuce dat na více serverech. [10]

Jelikož struktura dat, které budu ukládat, je jednoduchá a předem definovaná, rozhodl jsem se pro použití relační databáze. Zároveň si od toho slibuji snazší implementaci a jednodušší tvoření komplexních dotazů.

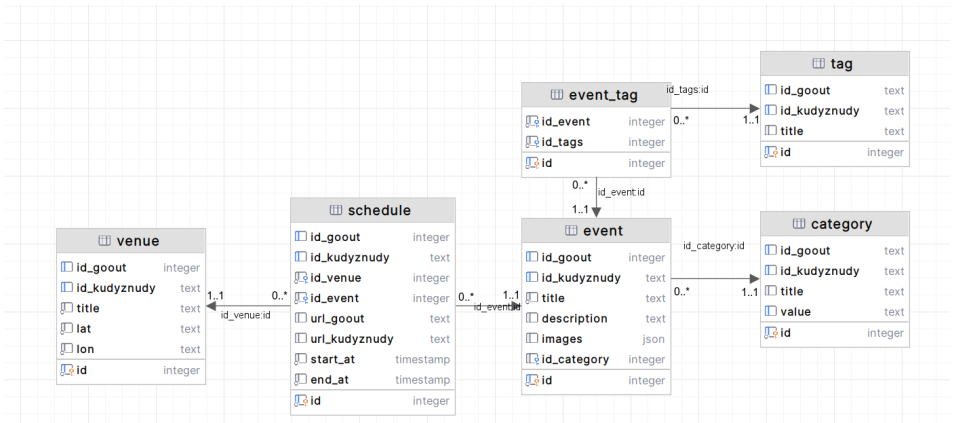
2.2.2 Model databáze

Model databáze obsahuje 6 tabulek, včetně jedné vztahové[1]. Všechny tabulky (kromě vazební) obsahují atributy s převzatými identifikátory z externího zdroje. Tímto se snažím o alespoň základní ošetření duplikace dat, které by jinak mohlo nastat, jakmile bych z externího zdroje získal již uložená data (tyto atributy už nebudu v následném popisu zmiňovat). Nejdůležitějšími tabulkami jsou **event**, **schedule** a **venue**.

Event reprezentuje událost, o které ukládá *title* (název), *description* (popis) a *images* (obrázky). Každá událost patří do právě jedné kategorie, kterou popisuje tabulka **category**.
V **category** se nachází *title* (název) a *value* (unikátní klíčové slovo kategorie). Událost může mít několik štítku pro lepší charakteristiku akce. Tento štítek je v databázi zastoupen tabulkou **tag**. Tabulka obsahuje pouze *title* (název). Vztah mezi štítkem a událostí řeší vazební tabulka **event_tag**.

Jelikož se akce může konat několikrát, není u entity události evidovaný čas konání. Pro tento účel slouží tabulka **schedule**, která reprezentuje jedno konání akce. Ta kromě *start_at* (začátek akce) a *end_at* (konec akce) obsahuje navíc atributy pro odkazy na stránku události z externího zdroje.

Pro úplnost informací je potřeba znát místo konání akce. To je uloženo v tabulce **venue**. O místě konání je evidováno *title* (název) a *lat* a *lon* (poloha). Grafické znázornění modelu můžete vidět na obrázku 2.3.



Obrázek 2.3 Relační model databáze

[1] Používá se pro dekompozici vztahu M:N.

2.2.3 Nasazení

Jelikož jeden z nefunkčních požadavků aplikace je veřejný přístup, musí být i databáze přístupná z internetu – je tedy potřeba ji nasadit na server. Já se rozhodl pro bezserverovou databázi kvůli rychlému nasazení a bez nutnosti správy.

Bezserverová databáze je termín označující službu, která nabízí databázové řešení bez nutnosti správy a údržby, s automatickým škálováním, s nepřetržitým přístupem a s automatickou ochranou a zabezpečením. [11]

2.3 Použité technologie

V této sekci stručně popíšu technologie, které jsem použil k vytvoření aplikace. Nejsou tu obsaženy všechny technologie, ale pouze ty nejdůležitější nebo nejzajímavější. Technologie jsem si vybral podle předchozích zkušeností, funkčních a nefunkčních požadavků aplikace, míry ulehčení práce, možnosti využití i pro budoucí rozšíření aplikace a v neposlední řadě podle popularity.

2.3.1 TypeScript

TypeScript je programovací jazyk vyvíjený firmou Microsoft. Jedná se o nadstavbu jazyka JavaScript, který obohacuje o statické typování[2]. Navíc přidává další koncepty objektově orientovaného programovaní, jako jsou na příklad interface nebo generické datové typy. Velkou výhodou je kompilace TypeScript programu do JavaScript kódu. Díky tomu nemusím pro spuštění programu instalovat nic navíc. [12]

TypeScript jsem si vybral z důvodu menší šance vytvoření chyby, lepší pomoci od vývojového prostředí a větší kontroly nad zdrojovým kódem.

```
interface User {
  firstName: string
  lastName: string
}

function updateUser(id: number, update: Partial<User>) {
  const user = getUser(id)
  const newUser = { ...user, ...update }
  saveUser(id, newUser)
}
```

Výpis kódu 2.1 Ukázka TypeScript kódu

2.3.2 React

React je JavaScript knihovna od společnosti Meta. Slouží k vytvoření uživatelského prostředí za pomoci jednotlivých oddělených částí, které jsou nazývány komponenty. Každá komponenta má svojí vlastní logiku a vzhled, které lze následně použít uvnitř jiné komponenty nebo je použít na vytvoření části stránky. Komponenta se v dnešní době implementuje jako funkce, která vrací JSX element. [13]

JSX je rozšíření JavaScriptu se syntaxí podobnou XML. Používá se pro tvorbu DOM elementů. [14] Tato syntaxe má následující výhody [15, 16]:

[2] Jedná se o třídu programovacích jazyků, u kterých je zapotřebí při deklaraci proměnné určit její datový typ. Do této třídy na příklad patří C++, C#, Java, …

Čitelnost – Syntaxe je podobná HTML syntaxi, kód tak může být pro vývojáře čitelnější, než vytváření elementů za pomocí JavaScript funkcí.

Rychlé pochopení – Jak jsem již zmiňoval, JSX je velice podobný jazyku HTML. Vývojáři se tak JSX syntax naučí snadno.

Integrace JavaScript výrazů – Jelikož je JSX rozšířením jazyka JavaScript, dovoluje využití JavaScript výrazů uvnitř elementů. Díky tomu umožňuje dynamické vykreslování stránek v závislosti na vlastní logice.

React jsem si vybral z důvodu předešlých zkušeností, dostupnosti pomocných knihoven pro mé řešení a velké popularitě, díky čemu je zajištěna dlouhá podpora této knihovny.

```
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="World" />;
root.render(element);
```

Výpis kódu 2.2 Ukázka React komponenty

2.3.3 Next.js

Next.js je full stack framework pro vývoj webových aplikací. Pro vykreslení uživatelského prostředí se zde používá knihovna React. Velkou výhodou tohoto frameworku je možnost SSR[3],což má za následek rychlejší načtení stránky a lepší SEO. Framework také nabízí řešení routování v aplikaci, které je definováno pomocí adresářové struktury. Kořenovým adresář se nazývá `app` a zastupuje kořenovou URL adresu. V tomto adresáři lze následně dále tvořit další složky, které vždy zastupují jeden segment URL cesty. Ukázku fungování routingu lze vidět na obrázku 2.4. V těchto složkách jsou následně obsaženy soubory, které vytváří uživatelské prostředí stránky na dané adrese. Chování při vykreslení je ovlivněno názvem souboru. [17]

Nejpoužívanějšími soubory jsou [18]:

page – Slouží k vykreslení hlavního obsahu. Bez přítomnosti tohoto souboru ve složce, není adresa veřejně přístupná.

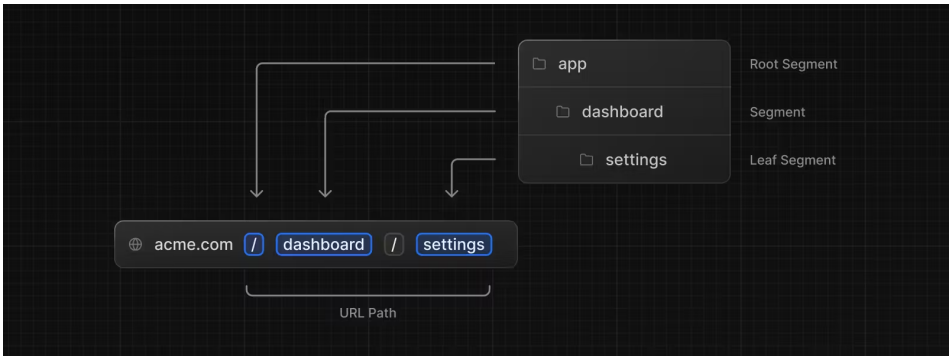
layout – Definuje sdílené uživatelské rozhraní pro všechny zanořené adresy v dané složce.

loading – Tento soubor vykreslí jeho obsah při probíhajícím načítaní dat v souboru `page`.

not-found – Používá se pro vykreslení informace, že daná stránka neexistuje.

Tento framework jsem si vybral, jelikož jedním z nefunkčních požadavků byl použit na vývoj aplikace full stack framework. Také jsem si ho zvolil pro jeho způsob řešení routování. V neposlední řadě jsem si ho vybral, jelikož mezi těmito frameworky je nejpopularnější a má velmi přehlednou dokumentaci.

[3]K vykreslení dojde na serveru a klientovi se pošle již kompletně vykreslená stránka.



Obrázek 2.4 Ukázka routingu v Next.js pomocí adresářové struktury [18]

2.3.4 Drizzle

Drizzle je ORM knihovna psaná v jazyce TypeScript. Nabízí propojení s většinou z populárních bezserverových i klasických databázových řešení. K manipulaci s databází obsahuje rozhraní typické pro ORM nástroje (dále jako Query API) nebo rozhraní imitující tvoření dotazů v SQL (dále jako SQL-like API). Díky Query API nemusím při tvoření dotazu, který obsahuje mnoho provázaných tabulek, řešit komplikované spojování a mapování dat na objekty. Naopak SQL-like API je výhodné, jakmile chci zkonstruovat komplexní dotaz, který by se za pomoci klasického ORM rozhraní tvořil těžko. [19]

Tuto knihovnu jsem si pro manipulaci s databází vybral, jelikož nevytváří téměř žádnou vrstvu abstrakce a dovoluje mi tak definovat schéma a dotazy přesně podle mých představ. Zároveň, díky implementaci knihovny v jazyce TypeScript, mi dovoluje udržovat kód typově bezpečný.

```
db.query.schedule.findFirst({
    where: eq(schedule.id, id),
    with: {
        event: {
            with: {
                category: true,
                tags: {
                    with: {
                        tag: true
                    }
                },
            },
        },
        venue: true
    }
});
```

Obrázek 2.5 Dotaz pomocí Query API

```
db.select({
    id: venue.id,
    title: venue.title,
    lat: venue.lat,
    lon: venue.lon,
}).from(venue).where(or(
    ilike(venue.title, `${query}%`),
    ilike(venue.title, `% ${query}%`))
).limit(limit);
```

Obrázek 2.6 Dotaz pomocí SQL-like API

2.3.5 Crawlee

Crawlee je původem česká knihovna pro automatizaci získávání dat z webu pomocí jazyka JavaScript. Pokud chci extrahovat data z webu, mám na výběr z velkého množství nástrojů, které jsou pro to určeny. Crawlee usnadňuje toto rozhodování a dovoluje mi pracovat s těmi

nejpopulárnějšími z nich jen za pomoci této knihovny. Nástroje jsou v knihovně Crawlee implementované jako samostatné třídy, které spolu sdílí stejné rozhraní. Díky tomu mohu výběr kdykoliv změnit bez nutnosti přepsání velké části kódu. [20]

Knihovna obsahuje tři hlavní třídy:

CheerioCrawler – Implementuje získávání dat pomocí HTTP požadavku. Tělo požadavku je následně zpracováno pomocí knihovny Cheerio[4], která nabízí široké rozhraní pro práci s HTML. Díky přístupu k datům pouze přes jeden HTTP požadavek je tato třída velmi rychlá. Nevýhodou je absence vyhodnocování JavaScript kódu – proto je nevhodnou volbou pro aplikace, které využívají k vykreslení obsahu JavaScript. [21]

PuppeteerCrawler – Pro otevření webové stránky používá headless[5] prohlížeč Chrome. Díky tomu dokáže načíst i stránku, která ke svému vykreslení vyžaduje JavaScript. K ovládání stránky a extrakci dat je zde využita knihovna Puppeteer[6]. Jelikož načtení stránky probíhá v prohlížeči, je o dost pomalejší než třída CheerioCrawler. Proto není vhodnou volbou, jakmile ke stažení stránky není potřeba JavaScript. [22]

PlaywrightCrawler – Tato třída funguje stejně jako třída PuppeteerCrawler, ovšem místo knihovny Puppeteer je zde použita knihovna Playwright[7]. Díky tomu nemusím použít pouze prohlížeč Chrome, ale i Firefox nebo Webkit. [23]

Crawlee navíc všechny tyto nástroje obohacuje o vlastní konfiguraci, díky které cílené weby ve většině případů nepoznají, že se nejedná o uživatele – dokáže tak obejít ochrany proti robotům. Pokud by mi tato konfigurace nestačila, Crawlee dovoluje vytvořit si vlastní s opravdu širokým množství nastavení, např. lokální úložiště, rotace proxy, správa relace, TLS handshake, …[20]

Tato knihovna je pro moji práci zásadní. Budu ji totiž využívat pro získávání dat z externích zdrojů. Vybral jsem si ji, jelikož obsahuje všechny pro mě potřebné nástroje a řeší za mě také jejich správné nastavení. Díky tomu se mohu soustředit na samotné algoritmy získávání dat a nemusím řešit technické detaily každého nástroje.

2.3.6 SWR

SWR je React knihovna pro usnadnění práce s načítáním dat z databáze. Název je zkratkou anglického výrazu „stale-while-revalidate“, který ve volném překladu znamená „neaktuální-během-ověření“. Výraz se vztahuje ke strategii cachování[8] dat, kdy se při požadavku o data vrátí nejdříve data z cache (neaktuální), zatímco se současně posílá požadavek (ověření) o aktuální data z databáze. To má za následek okamžité zobrazení dat uživateli, zatímco se provádí nový požadavek na získání dat aktualizovaných. [24]

Funkce této knihovny budu využívat při načítání událostí, ovšem pouze jako cache dat na klientovi. Šance změny událostí při používání aplikace uživatelem je minimální. Díky tomuto rozhodnutí se však zmenší počet odeslaných požadavků, a tím i množství stažených dat. Co nejmenší vytížení datového připojení je důležité hlavně pro uživatele, kteří budou aplikaci používat na mobilu.

[4] Cheerio, `https://cheerio.js.org`
[5] Typ prohlížeče, který nemá grafické uživatelské rozhraní.
[6] Puppeteer, `https://pptr.dev`
[7] Playwright, `https://playwright.dev`
[8] Mezipaměť pro dočasné ukládání dat, ke kterým má aplikace následně rychlejší přístup bez nutnosti použití síťového připojení.

2.3.7 Supercluster

Supercluster je JavaScript knihovna pro shlukování geoprostorových bodů. Hlavní algoritmus funguje na bázi hladového shlukování. Nejdříve se vezme libovolný bod, najdou se všechny body v požadované vzdálenosti a vytvoří se z nich nový shluknutý bod. Tento krok se dál opakuje pro každý bod, který ještě není součástí žádné množiny bodů. Pro lepší efektivitu výpočtu při počítání nové úrovně přiblížení využívá tento algoritmus již vytvořené shluknuté body z vyšší úrovně přiblížení, na kterých následně provádí stejné operace. [25]

Tato knihovna je pro mě velmi důležitá, jelikož ji budu používat pro shlukování událostí na mapě. Výpočet těchto dat proto musí být přesný a co nejrychlejší, aby byl zaručen nejlepší uživatelský zážitek.

2.3.8 Maps JavaScript API

Maps JavaScript API (dále jen jako API) je služba poskytovaná společností Google, která umožňuje integrovat interaktivní mapy do webových aplikací. Díky tomuto lze libovolně upravovat vzhled mapy, měnit vrstvy, přidávat vlastní značky nebo body či nahradit ovládání vlastní logikou. API také podporuje různé služby, jako jsou geokódování[9] a reverzní geokódování[10], nalezení cesty mezi body, vyhledání místa nebo zobrazení Street View. [26]

Vybral jsem si API od společnosti Google, jelikož patří mezi nejpoužívanější a má nejširší pokrytí. Jako výhodu jsem také bral velké množství propojitelných služeb, které se mohou hodit pro další budoucí vývoj.

2.3.9 React Google Maps

React Google Maps je kolekce React komponent a funkcí pro snadnější vývoj aplikací komunikující s Maps JavaScript API. Způsob, jakým API funguje, se neslučuje s filozofií React aplikace. API si samo spravuje stavy všech informací potřebné k aktualizaci mapy. To je žádoucí pro správné fungování, jelikož obsahuje spoustu funkcí a možností, jak mapu ovládat nebo s ní jinak interagovat. Ovšem pro správné vykreslení všech komponent Reactem je zapotřebí, aby všechny stavy spravoval sám React a dokázal tak komponenty synchronizovat. Tato kolekce řeší uvedené problémy ve svých připravených komponentách, díky kterým můžeme používat API jako klasické React komponenty bez nutnosti řešení propojení. [27]

Pro tyto připravené komponenty jsem se rozhodl z důvodu jednoduchého a rychlého propojení mé aplikace s API. Zároveň nevytváří příliš velkou vrstvu abstrakce a nabízí všechny možnosti úpravy a manipulace jako samotné API.

[9] Převádění adres na geografické souřadnice.
[10] Nalezení adres na základě geografických souřadnic.

Kapitola 3

Implementace

V této kapitole budu věnovat pozornost implementaci mého řešení. Jako první popíši proces vytvoření databáze a implementaci navrženého schéma. Následně se budu věnovat samotné webové aplikaci, která bude implementovat navržené uživatelské rozhraní. Poté objasním postup při vyvíjení aplikace na získávání událostí z externích zdrojů.

3.1 Databáze

Aby mohla má webová aplikace a také aplikace na získávání událostí fungovat, je potřeba nejdříve vytvořit databázi, ve které budu moct ukládat získaná data. V následující sekci popíši, jak probíhal postup vytvoření databáze a databázového schéma.

3.1.1 Vytvoření databáze

Pro vytvoření databáze jsem využil službu Vercel Postgres[1] od firmy Vercel. Vercel Postgres nabízí bezserverové databázové řešení postavené na open-source relační databázi PostgreSQL. [28] Tuhle službu jsem si vybral z důvodu možnosti bezplatné verze, jednoduchého přehledného prostředí a využití této platformy na nasazení i samotné aplikace. K vytvoření databáze stačí vytvořit nový projekt na platformě Vercel[2], ve které pomocí krátké konfigurace vytvořím svoji novou databázi. Následně je důležité uložit informace potřebné k propojení aplikace.

3.1.2 Vytvoření schéma

Schéma databáze jsem vytvořil pomocí knihovny Drizzle. Z této knihovny jsem použil především funkce `pgTable` a `relations`. Funkce `pgTable` se používá pro definici tabulky v databázi PostgreSQL. Přijímá dva hlavní argumenty – název tabulky a objekt složený z definic sloupců. Funkce `relations` pak slouží k definici vazeb mezi tabulkami. Definování tabulek `venue` a `schedules` lze vidět v kódu 3.1.

[1] Vercel Postgres, `https://vercel.com/storage/postgres`
[2] Dostupná na: `https://vercel.com/home`

```
export const venue = pgTable("venue", {
    id: serial("id").primaryKey(),
    idGoout: integer("id_goout").unique(),
    idKudyznudy: text("id_kudyznudy").unique(),
    title: text("title").notNull(),
    lat: text("lat").notNull(),
    lon: text("lon").notNull(),
});
export const schedule = pgTable("schedule", {
    id: serial("id").primaryKey(),
    idGoout: integer("id_goout").unique(),
    idKudyznudy: text("id_kudyznudy").unique(),
    venue: integer("id_venue").references(() => venue.id).notNull(),
    event: integer("id_event").references(() => event.id).notNull(),
    urlGoout: text("url_goout"),
    urlKudyznudy: text("url_kudyznudy"),
    startAt: timestamp("start_at").notNull(),
    endAt: timestamp("end_at").notNull(),
});
export const venueRelations = relations(venue, ({many}) => ({
    schedules: many(schedule),
}));
export const schedulesRelations = relations(schedule, ({one}) => ({
    venue: one(venue, {fields: [schedule.venue], references: [venue.id]}),
}));
```

Výpis kódu 3.1 Definování tabulek `venue` a `schedules`

K vytvoření definovaných tabulek jsem použil nástroj `drizzle-kit`[3], který zajišťuje sestavení a aplikování migrací z definic vytvořených pomocí knihovny Drizzle. Pro správné fungování tohoto příkazu je potřeba konfigurační soubor `drizzle.config.ts` (obsah souboru lze vidět v kódu 3.2). Já jsem použil pouze povinné parametry – `schema` (pro definování umístění souboru s definovanými tabulkami), `driver` (říkající jakou databázi používám) a `dbCredentials` (obsahující potřebné informace pro připojení k databázi). Po nakonfigurování příkazu mohu pomocí `drizzle-kit push:pg` spustit migraci databáze.

```
export default defineConfig({
    schema: "./src/db/schema.ts",
    driver: "pg",
    dbCredentials: {
        connectionString: process.env.POSTGRES_URL!
    }
});
```

Výpis kódu 3.2 Konfigurace příkazu `drizzle-kit`

[3] Drizzle Kit, `https://orm.drizzle.team/kit-docs/overview`

3.2 Webová aplikace

Implementace webové aplikace je jedním z hlavních cílů mé práce. V této sekci popíši nejdůležitější a nejzajímavější části z tohoto procesu. Nejdříve objasním proces vytvoření samotné mapy, na které budu zobrazovat události. Následně popíši, jak jsem vytvářel jednotlivé komponenty pro správné zobrazení uživatelského rozhraní podle návrhu. Poté se pokusím přiblížit postup při řešení shlukování míst, lokalizace uživatele a filtrace událostí. Nakonec se zaměřím na nasazení aplikace.

3.2.1 Vytvoření mapy

Pro použití mapy přes Maps JavaScript API je zapotřebí získat API klíč. Ten získám po vytvoření projektu na platformě Google Cloud[4]. Klíč lze používat pro vícero služeb, ovšem jeho působení jde omezit v rozsáhlém nastavení. Google doporučuje omezení nastavit co nejstriktnější[5], proto bude můj klíč fungovat pouze na toto jedno API a navíc bude omezený pouze pro moji doménu. V projektu je možné také upravit styl mapy. Na ten jsem nijak zásadně nesahal, pouze jsem zakázal zobrazení pár typů míst, jelikož pro aplikaci nemá jejich zobrazení žádný význam. Naopak jsem nechal dle mého názoru důležité značky, jako na příklad zastávky hromadné dopravy, turistické atrakce nebo kulturní místa.

Pro vykreslení mapy jsem využil komponenty `Map` z kolekce React Google Maps (zjednodušenou implementaci lze vidět v kódu 3.3). Aby mapa fungovala, je potřeba do této komponenty vložit vytvořený API klíč jako atribut `mapId`. Navíc jsem nastavil střed mapy na střed České republiky a také výchozí, maximální a minimální přiblížení. Abych měl nad mapou co největší kontrolu, vypnul jsem všechna ovládání a zakázal klikatelné značky výchozích míst na mapě. Pro správné zpracování uživatelského vstupu je potřeba nastavit atribut `gestureHandling` na hodnotu `"greedy"`. Tím se zajistí, že mapa zachytí veškerou interakci od uživatele.

```
<Map
    mapId={process.env.NEXT_PUBLIC_GOOGLE_MAP_ID!}
    defaultCenter={{lat: 49.803, lng: 15.474}}
    defaultZoom={8}
    maxZoom={19}
    minZoom={7}
    streetViewControl={false}
    mapTypeControl={false}
    fullscreenControl={false}
    clickableIcons={false}
    gestureHandling="greedy"
>
    {markers.map((marker) => <Marker/>)}
</Map>
```

Výpis kódu 3.3 Zjednodušená implementace komponety `Map` z kolekce React Google Maps

[4]Dostupná na: `https://cloud.google.com`

[5]Klíč se bude využívat na klientské části aplikace a může ho tak kdokoli zneužít.

3.2.2 Značky událostí a míst

Pro zobrazení událostí na mapě jsou v návrhu uživatelského rozhraní použity dva typy značek:

Značka události – používá se, jakmile se na místě děje pouze jedna akce

Značka místa – slouží k zobrazení více akcí na jednom místě či při shluknutí míst

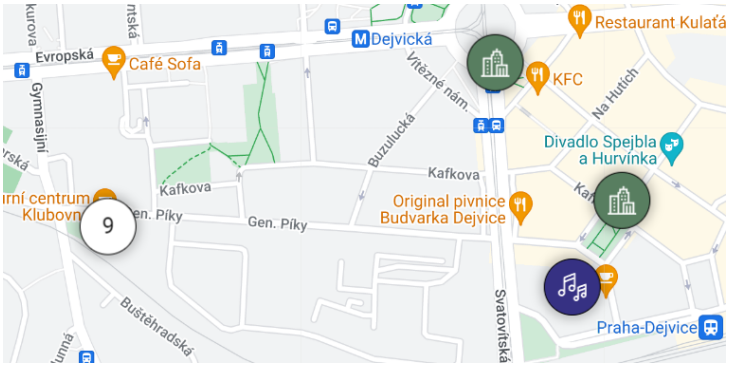
Pro každou značku jsem si vytvořil samostatnou komponentu, ve které používám komponentu `AdvancedMarker` z kolekce React Google Maps pro propojení s Maps JavaScript API (implementaci komponenty pro značku místa lze vidět v kódu 3.4). Obě tyto komponenty přijímají jako své atributy pozici místa a funkci, která se má zavolat po kliknutí na značku. Pozici využívá komponenta `AdvancedMarker` pro vykreslení značky na mapě na správném místě. Značka místa má navíc atribut pro počet konaných událostí a značka události přijímá navíc kategorii události.

Jelikož metoda `getClusters` vrací pole objektů, které může obsahovat dohromady jak clustery tak místa, je pro výběr správné značky potřeba tyto objekty mezi sebou rozlišit. Datový typ objektů používaný třídou `Supercluster` rozlišuje mezi clusterem a místem pomocí atributu `cluster`, který je u clusteru přítomen. Díky tomu lze snadno rozhodnout, zdali se jedná o objekt clusteru podle existence tohoto atributu. U vykreslení samotného místa vyberu správný typ značky podle počtu konaných akcí na daném místě. Implementace vykreslení značek je zobrazena v kódu 3.5

```
export default function Marker(props: Props) {
    const {position, category, onMarkerClick} = props;
    const [markerRef, marker] = useAdvancedMarkerRef();

    return (
        <AdvancedMarker
            className={styles.marker}
            position={position}
            ref={markerRef}
            onClick={(e) => onMarkerClick?.(marker, e)}
        >
            <CategoryIcon category={category} size={24} />
        </AdvancedMarker>
    );
}
```

Výpis kódu 3.4 Implementace komponenty pro značku události



Obrázek 3.1 Náhled vykreslení značek

```
clusters.map((cluster) => {
    const {geometry: {coordinates}, properties} = cluster;
    const position = {lat: coordinates[1], lng: coordinates[0]};
    if (properties.hasOwnProperty("cluster")) {
        const {cluster_id, venues} = properties;
        const count = getSchedulesCount(venues);
        return (
            <ClusterMarker key={cluster_id} position={position} count={count} />
        );
    }
    const {venue: {id: venueId, schedules}} = properties;
    const category = schedules[0].event.category;
    const count = schedules.length;
    return schedules.length > 1 ? (
        <ClusterMarker key={venueId} position={position} count={count} />
    ) : (
        <Marker key={venueId} position={position} category={category} />
    );
})
```

Výpis kódu 3.5 Vykreslení pole clusterů a míst

3.2.3 Souhrn událostí a výpis událostí

Po kliknutí na značku na mapě se zobrazí dialog se souhrnem události nebo s výpisem událostí. Vytvořil jsem si proto funkci `handleMarkerClick`, která bude řešit správné vyhodnocení kliknutí na značku události a značku místa (provedení funkce lze vidět v kódu 3.6). Tuhle funkci vložím do komponent značek jako atribut `onClick`, čímž docílím spuštění funkce pokaždé, co uživatel klikne na nějakou značku. Také jsem si vytvořil stav `selectedMarker`, ve kterém jsou uložené všechny potřebné informace o značce a reprezentovaných datech. Funkce `handleMarkerClick` nejdříve porovná, zdali uživatel neklikl na značku, která již je označena. Pokud tato situace nastala, přepíše se hodnota stavu `selectedMarker` na `null`, čímž se vyvolá překreslení, ale již bez označení značky (dojde k zavření dialogu). V opačném případě se uloží do tohoto stavu informace o místě. K odlišení mezi zobrazením souhrnu události a výpisu událostí slouží hodnota `schedulesCount`, ve které je uložen počet konaných akcí na místě.

Pro oba typy dialogu jsem vytvořil vlastní komponenty, ve kterých k vykreslení na mapu využívám komponentu `InfoWindow` z kolekce React Google Map. Tato komponenta řeší propojení s Maps JavaScript API a spojení dialogu se značkou na mapě, díky kterému se bude vždy zobrazovat u správné značky. Aby toto spojení fungovalo, musím komponentě `InfoWindow` nastavit atribut `anchor`, do kterého náleží reference na element značky. Tu mám uloženou ve stavu `selectedMarker` a do komponenty ji posílám přes atribut komponenty `markerRef`.

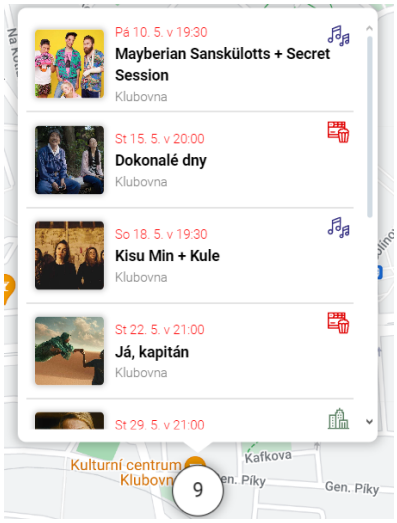
```
const handleMarkerClick = (venue: GetVenuesResponse[0], markerRef) => {
    if (selectedMarker?.markerRef === markerRef) {
        setSelectedMarker(null);
    } else {
        const {schedules} = venue;
        setSelectedMarker({
            markerRef,
            venueIds: [venue.id],
            schedulesCount: schedules.length,
            scheduleId: schedules.length === 1 ? schedules[0].id : undefined,
        });
    }
};
```

Výpis kódu 3.6 Funkce pro řešení kliknutí na značku události

Zobrazovaná data v dialogu z databáze načítám až při jeho otevření. Data o všech událostech bych sice mohl stáhnout hned při otevření aplikace, ale při tomto množství dat by to trvalo spoustu času a většina dat o událostech by se stahovala zbytečně. Z tohoto důvodu k manipulaci se staženými daty používám funkci `useSWRImmutable` z knihovny SWR. Tato funkce ukládá získaná data do cache na klientské straně aplikace. Immutable v názvu funkce říká, že nebude posílat požadavky na server pro aktualizaci již uložených hodnot v cache. Je velice malá šance změny databáze během používání aplikace uživatelem, proto se tato funkce hodí na omezení posílaných požadavků na server. Pro její zprovoznění je potřeba vložit jako první argument funkce klíč, podle kterého bude data ukládat do cache. Druhým argumentem je pak funkce k načtení dat z databáze, do kterého použiji speciální funkci frameworku Next.js nazývanou Server Action. Server Action je funkce, která se vyhodnocuje na serveru. Díky tomu můžu komunikovat s databází bez hrozby úniku citlivých dat. Při volání této funkce na klientovi, data se předají ze serveru pomocí HTTP POST požadavku. Server Action funkci pro získání událostí lze vidět v kódu 3.7. Pro sestavení dotazu jsem využil SQL-like API z knihovny Drizzle.

```
export async getSchedules(filters: GetSchedulesFilters) {
    const queryBuilder = db.select({
            id: schedule.id,
            startAt: schedule.startAt,
            endAt: schedule.endAt,
            event: {title: event.title, category: category.value},
            venue: {title: venue.title},
        }).from(schedule)
        .innerJoin(event, eq(schedule.event, event.id))
        .innerJoin(venue, eq(schedule.venue, venue.id))
        .leftJoin(category, eq(event.category, category.id));
    let queryFilters = [];
    if (filters.venueIds) {
        queryFilters.push(inArray(venue.id, filters.venueIds));
    }
    return queryBuilder.where(and(...queryFilters))
        .orderBy(asc(schedule.startAt));
};
```

Výpis kódu 3.7 Server Action funkce pro získání událostí z databáze



Obrázek 3.2 Náhled vykreslení výpisu událostí

3.2.4 Detail události

Na detail akce se může uživatel dostat přes souhrn události nebo přes kliknutí na jednu z akcí ve výpisu událostí. Manipulace s daty probíhá stejně jako u souhrnu události nebo výpisu událostí. Data se stahují z databáze až po otevření detailu. Pro ukládání dat do cache používám funkci `useSWRImmutable`. K načtení dat z databáze dojde pomocí Server Action funkce. V návrhu uživatelského rozhraní má detail události výšku závislou na výšce obsahu. Jelikož jsem tohoto pravidla nedokázal docílit pouze pomocí CSS stylů, musel jsem k tomu použít javascript funkce. Vytvořil jsem si k tomu funkci `resizeDialog`, která je zobrazena v kódu 3.8. Jako jediný argument je zde reference na element modálního okna. Tuhle funkci totiž používám jako atribut `ref` tohoto elementu, který spustí moji funkci přesně s touto referencí v argumentu. Ve funkci je následně porovnána výška elementu popisu vůči výšce obsahu elementu. Pokud je obsah stejně vysoký jako element, znamená to, že je vidět celý a není potřeba okno zvětšovat. Pokud je ovšem větší, zvětším modální okno o výšku obsahu, který není vidět.

```
const resizeDialog = (dialog: HTMLDivElement | null) => {
    const description = document.getElementById("description");
    const descScrollHeight = description?.scrollHeight || 0;
    const descHeight = description?.clientHeight || 0;
    if (description && dialog && descScrollHeight > descHeight) {
        dialog.style.height =
            dialog.clientHeight + (descScrollHeight - descHeight) + "px";
    }
};
```

Výpis kódu 3.8 Funkce pro dynamické nastavení výšky modálního okna

3.2.5 Shlukování míst

Shlukování míst (dále jako cluster) jsem implementoval pomocí třídy `Supercluster` z knihovny Supercluster (použití lze vidět v kódu 3.9. Tato třída přijímá jako svůj jediný argument objekt s atributy nastavení clusteru. Já jsem nastavil poloměr clusteru, maximální a minimální přiblí-

žení, pro které se bude ještě shlukování počítat, a dvě funkce `map` a `reduce`. Funkce `map` definuje, jak má třída namapovat vlastnosti míst do clusteru. Funkce `reduce` zase definuje, jakým způsobem má spojit vlastnosti dvou clusterů při shluknutí do sebe. Po inicializaci instance (dále jako index) je do ni potřeba místa nahrát. Při nahrání míst je důležité dodržovat správný datový typ požadovaný metodou `load`. Nejdůležitější hodnoty v tomto datovém typu jsou `coordinates`, do které patří poloha místa, a hodnota `properties`, ve které budu mít uložené data místa.

```
loadNewIndex(venues: GetVenuesResponse): Supercluster<P, C> {
    const index = new Supercluster<P, C>({
        radius: 160,
        maxZoom: 20,
        minZoom: 7,
        map: (props) => ({venues: [props.venue]}),
        reduce: (accumulated, props) => {
            accumulated.venues = [...accumulated.venues, ...props.venues];
        },
    });
    index.load(venues.map((venue) => ({
        type: "Feature",
        geometry: {type: "Point", coordinates: [venue.lon, venue.lat]},
        properties: {venue},
    }) as Supercluster.PointFeature<P>));
    return index;
};
```

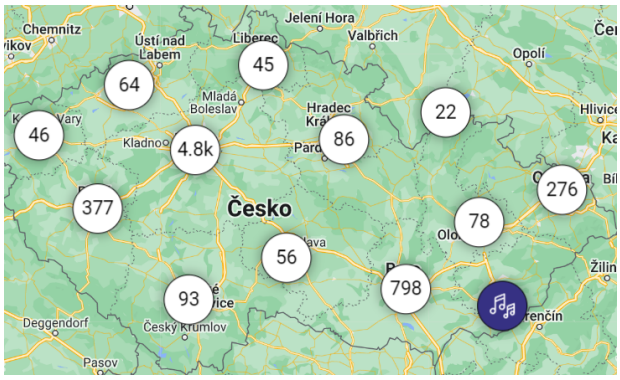
Výpis kódu 3.9 Použití třídy `Supercluster` z knihovny Supercluster

Abych mohl clustery vykreslit, je potřeba použít metodu indexu `getClusters`. Ta v závislosti na argumentech ohraničení a přiblížení mapy vrátí viditelné clustery. Metoda je výpočetně náročná a měl bych ji proto volat co nejméně. Jakmile bych volal tuto metodu pokaždé, co by došlo ke změně přiblížení nebo ohraničení mapy, během dlouhé změny polohy mapy by se metoda zavolala hned několikrát. Ke změně clusterů by však mělo dojít až po dokončení jakékoliv interakce uživatele s mapu. Přesně pro tyto účely slouží atribut `onIdle` komponenty `Map`, který zavolá požadovanou funkci, jakmile dojde k ukončení všech interakcí. Funkce, kterou vložím do atributu `onIdle`, bude volat zmíněnou funkci indexu `getClusters` a získané clustery si uloží jako stav. React vyvolá nové vykreslení pokaždé, co dojde ke změně nějakého stavu. To znamená, že vždy co dojde k ukončení změny polohy mapy, přepočítají se clustery, a kvůli změně stavu React vykreslí nové clustery na mapu. Tento sled událostí zajistí uživateli co nejplynulejší odezvu bez přílišného plýtvání výpočetního výkonu.

Po kliknutí na cluster by se podle návrhu uživatelského rozhraní měla buďto přiblížit mapa na jemnější rozdělení míst, nebo by se měl zobrazit výpis všech událostí shluknutých míst, pokud jsou příliš blízko u sebe. K tomuto účelu jsem si vytvořil funkci `handleClusterClick`, která se spustí po každém kliknutí na cluster. Pomocí metody `getClusterExpansionZoom` nejdříve zjistím, na jaké úrovni přiblížení dojde k rozdělení označeného clusteru. Pokud je úroveň nižší než maximální povolená, mapa se přiblíží na tuhle úroveň a vycentruje se na polohu clusteru. Když by přiblížení bylo příliš velké, funkce dál pokračuje podobně jako funkce v kódu 3.6 s tím rozdílem, že aktivní není pouze jedno místo, ale hned několik.

```
const handleClusterClick = (
    clusterProperties: Supercluster.ClusterFeature<C>["properties"],
    markerRef: google.maps.marker.AdvancedMarkerElement,
    ev: google.maps.MapMouseEvent
) => {
    const {venues, cluster_id} = clusterProperties;
    const zoom = index.current.getClusterExpansionZoom(cluster_id);
    if (zoom <= 19) {
        ev.latLng && map?.panTo(ev.latLng);
        map?.setZoom(zoom);
    } else if (selectedMarker?.markerRef === markerRef) {
        setSelectedMarker(null);
    } else {
        const selectedMarkerState: SelectedMarkerState = {
            markerRef,
            venueIds: [],
            schedulesCount: 0
        };
        for (const {id, schedules} of venues) {
            selectedMarkerState.venueIds.push(id);
            selectedMarkerState.schedulesCount += schedules.length;
        }
        setSelectedMarker(selectedMarkerState);
    }
};
```

Výpis kódu 3.10 Funkce pro vyhodnocení kliknutí na cluster



Obrázek 3.3 Shlukování událostí napříč republikou

3.2.6 Lokalizace

Pokud aplikace dostane přístup k poloze uživatele, zobrazí jeho blízké okolí na mapě. Pro tuhle funkcionalitu jsem si vytvořil vlastní komponentu, která bude mít na starosti vykreslení tlačítka podle návrhu uživatelského rozhraní a samotnou lokalizaci. K lokalizaci uživatele jsem si vytvořil funkci `handleClick` (funkci lze vidět v kódu 3.11), která se spustí pokaždé, co uživatel klikne na tlačítko pro lokalizaci. Pro získání polohy využívám funkci `getCurrentPosition` z Geolocation API. Geolocation API je jedno z mnoha API, které jsou v základu ve většině moderních prohlí-

žečů. Není proto potřeba žádných dodatečných balíčků nebo knihoven. Tato funkce přijímá tři atributy – funkce pro úspěšnou lokalizaci, funkce pro neúspěšnou lokalizaci a objekt s parametry lokalizace. Pokud lokalizace proběhla úspěšně, mapa se přesune na pozici uživatele a nastaví se stav lokalizace na aktivní. Pokud se z nějakého důvodu nezdařila, nastaví se stav na chybu. Tento stav následně používám pro správné zobrazení uživatelského rozhraní. Do nastavení lokalizace přidávám povolení pro použití vyšší přesnosti a automatické přerušení lokalizace po jedné vteřině. Přerušení lokalizace zde nastavuji z důvodu občasného zaseknutí na nekonečném lokalizování na mobilu. Touto podmínkou bych tomu tak měl předejít.

```
const handleClick = () => {
    if (!navigator.geolocation) {
        setLocationState(LocationState.error);
    } else {
        navigator.geolocation.getCurrentPosition(
            ({coords: {latitude, longitude}}) => {
                map?.panTo({lat: latitude, lng: longitude});
                map?.setZoom(16);
                setLocationState(LocationState.active);
            },
            () => setLocationState(LocationState.error),
            {timeout: 1000, enableHighAccuracy: true}
        );
    }
};
```

Výpis kódu 3.11 Funkce pro lokalizaci uživatele

3.2.7 Filtrace

Filtrace událostí patří mezi funkční požadavky aplikace. Filtrovat události jde podle její kategorie a lze nastavit, zda se zobrazí pouze momentálně konané akce. Z časových důvodů jsem byl schopný implementovat pouze filtraci událostí podle kategorie. Jednotlivé kategorie ve filtraci jsou reprezentovány pomocí HTML elementu `input`. Tento element je určen pro interaktivní prvky formuláře za účelem získání dat od uživatele. Elementu přiřazuji atribut `type` s hodnotou `checkbox` – tím dám webovému prohlížeči na vědomí, že se jedná o zaškrtávací pole. Element `input` se často páruje s elementem `label`, který slouží pro reprezentaci názvu prvku. Já element `input` umístím do tohoto elementu, díky čemuž mu nemusím přiřazovat atribut `for`. Vytvoření kategorie ve filtraci lze vidět v kódu 3.12.

```
<label>
    <input
        type="checkbox"
        name="category"
        value={category.id}
        onChange={handleCategoryChange}
    />
    {category.title}
</label>
```

Výpis kódu 3.12 Reprezentace kategorie ve filtraci pomocí `input` elementu

Abych dokázal zachytit změnu stavu prvku, vytvořil jsem si funkci `handleCategoryChange`, kterou přiřadím jako atribut `onChange` každému poli. Tato funkce má jako argument objekt informací o vyvolané akci. Pro mě je z tohoto objektu nejdůležitější atribut `target`, který uchovává informace o cíli této akce. Díky tomu můžu zjistit hodnotu `input` elementu a také zda byl zaškrtnutý či nikoliv. Tyto informace následně použiji pro úpravu filtrace. Jestliže uživatel pole zaškrtl, přidám kategorii do filtru, při opačné akci kategorii z filtru vyjmu. Nyní vytvořím nový objekt s filtry, kde přepíšu filtr kategorií novou verzí.

Stav filtrace je potřeba někam ukládat. Já se rozhodl pro uložení stavu jako URL parametr. Tento způsob ukládaní patří v praxi mezi běžnou praktiku, jelikož lze následně sdílet mezi uživateli a ti uvidí stejný výsledek. Objekt s filtry před vložením do parametru zakóduji do base64. Díky tomu můžu celý filtr vložit pouze do jednoho parametru a nemusím řešit rozdělení na více parametrů. Funkci `handleCategoryChange` můžete vidět v kódu 3.13.

```
const handleCategoryChange = (ev: React.ChangeEvent<HTMLInputElement>) => {
    const {value, checked} = ev.target;
    const categoryId = parseInt(value);
    const newCategories = checked
        ? [...activeCategories, categoryId]
        : activeCategories.filter((category) => category !== categoryId);

    const currentActiveFilters = {
        ...activeFilters,
        categoryIds: newCategories.length > 0 ? newCategories : undefined,
    };

    if (currentActiveFilters.categoryIds) {
        const encodedFilters = btoa(JSON.stringify(currentActiveFilters));
        const searchParams = new URLSearchParams();
        searchParams.set("f", encodedFilters);
        router.replace(`${pathname}?${searchParams.toString()}`);
    } else {
        router.replace(pathname);
    }
};
```

Výpis kódu 3.13 Funkce pro změnu filtrace

3.2.8 Nasazení

Jelikož jeden z nefunkčních požadavků je veřejný přístup k aplikaci, je potřeba nasadit aplikaci na server. K nasazení jsem využil opět službu firmy Vercel. K nasazení stačí do již vytvořeného projektu připojit můj repozitář na GitHubu a krátká konfigurace. Služba pak za mě aplikaci sama nasadí pokaždé, co se v repozitáři objeví nová změna. Služba nabízí i docela hezkou doménu, kterou si můžu změnit. Já si vybral doménu `https://nearby.vercel.app`.

3.3 Web scraper

Tato sekce se věnuje implementaci aplikace na získávání dat z externích zdrojů. Nejdříve vyřeším problém s komunikací s databází. Následně popíši proces implementace získávání dat z aplikace goout.net. Nakonec objasním provedené kroky pro co nejjednodušší přidaní dalšího externího zdroje.

3.3.1 Komunikace s databází

Abych zamezil pokusům o vložení již existující entity do databáze a šetřil tak její využití, před vložením zkontroluji, zda se již v databázi nenachází. K tomu je potřeba před začátkem scrapingu získat všechny entity z databáze. Tato aplikace však nemá přímý přístup k mé databázi a musím k tomu proto využít hlavní aplikaci. K zprovoznění komunikace mezi aplikacemi využívám funkci frameworku Next.js zvanou Router Handlers. Router Handler je funkce, která dovoluje posílat odpovědi libovolného formátu na požadavky směřované na URL adresu odpovídajícího Router Handleru. Router Handler se definuje podobně jako webová stránka s rozdílem, že soubor s funkcí se musí nazývat `route`. V tomto souboru může být následně vícero funkcí, kde název funkce definuje HTTP metodu, pro kterou je určena. Já budu využívat hlavně GET metodu pro posílání identifikátorů entit a POST metodu pro uložení entit do databáze. V kódu 3.14 níže lze vidět vytvoření Router Handleru pro entitu konání události.

```
export async function GET() {
    const data = await getScheduleIds();
    return Response.json(data, {status: 200});
}
export async function POST(request: Request) {
    const body = await request.json();
    if (!body.venue || !body.event || !body.startAt || !body.endAt) {
        return Response.json({message: "Missing values"}, {status: 400});
    }
    try {
        const insertedData = await addSchedule(body as InsertSchedule);
        return Response.json(insertedData, {status: 201});
    } catch (error) {
        return Response.json({message: error}, {status: 500});
    }
}
```

Výpis kódu 3.14 Router Handler pro entitu `Schedule`

3.3.2 Získání dat z goout.net

Při rešerši konkurenčních řešení jsem zjistil, že data z aplikace goout.net lze získat pomocí HTTP GET požadavku rovnou z jejich API. K zjištění struktury odpovědi jsem použil aplikaci Postman. Postman je nástroj, který umožňuje vytvářet, testovat a dokumentovat HTTP požadavky a odpovědi. Může být použit pro vývoj a testování různých typů webových služeb, jako jsou třeba REST, SOAP nebo GraphQL.

Po provedené analýze odpovědi jsem si vytvořil třídu `GooutScraper`, která bude získávat data (dále jako scraping) z goout.net pomocí zmíněného HTTP GET požadavku. Pro scraping využívám třídy `CheerioCrawler` z knihovny Crawlee, která umí zpracovat i JSON dokumenty. Tato třída k inicializaci přijímá dva argumenty, objekt s nastavením průběhu scrapingu a objekt s obecnou konfigurací. V nastavení je nejdůležitější atribut `requestHandler`, který definuje spouštěnou funkci pro zpracování každé odpovědi. Aby třída věděla, že bude zpracovávat JSON odpověď, je potřeba do nastavení přidat atribut `additionalMimeTypes`. Navíc ještě přidám atribut `maxConcurrency` s hodnotou jedna, čímž zajistím, že se bude zpracovávat maximálně jeden požadavek najednou a nebudu přetěžovat externí zdroj. V konfiguraci nastavím jediný parametr a to ten, aby se neukládala žádná data na disk. Inicializaci objektu lze vidět v kódu 3.15.

```
const crawler = new CheerioCrawler(
    {
        maxConcurrency: 1,
        maxRequestsPerCrawl: 160,
        maxRequestRetries: 0,
        additionalMimeTypes: ["application/json"],
        requestHandler: this.handler.bind(this),
    },
    new Configuration({persistStorage: false}),
);
```

Výpis kódu 3.15 Inicializace objektu `CheerioCrawler`

Funkce, kterou posílám do tohoto objektu, má jediný argument typu `CheerioCrawlingContext`. Tento objekt slouží k ovládání scraperu. Já využívám hlavně atribut `json`, který obsahuje zpracovaný obsah odpovědi v JSON formátu, a metodu `addRequest`, pomocí které přidávám další URL adresy pro scraping. Jelikož API aplikace goout.net používá pro své odpovědi stránkování, zkontroluji, zda neexistuje další stránka výsledku. Pokud ano, přidám ji jako další URL adresu pro scraping. Následuje procházení dat, které procházím po jednotlivých entitách. Nejdříve zkontroluji, zda již nejsou v databázi pomocí objektů obsahující identifikátory entit. Pokud tomu tak není, přidám je do databáze a jejich identifikátor do příslušného objektu. Zjednodušenou verzi této funkce lze vidět v kódu 3.16.

```
private async handler(context: CheerioCrawlingContext) {
    const responseJSON = context.json as GooutResponseType;
    const scrollId = responseJSON.meta.nextScrollId;

    if (scrollId) {
        await context.addRequests([gooutURL(scrollId)]);
    }

    for (const event of responseJSON.included.events) {
        if (this.events[event.id] === undefined) {
            const {tags} = event.attributes;
            await addEvent(event);

            for (const tag of tags) {
                !this.tags[tag] && await addTag(tag);
                await addEventTag(this.events[event.id], this.tags[tag]);
            }
        }
    }
    for (const venue of responseJSON.included.venues) {
        !this.venues[venue.id] && await addVenue(venue);
    }
    for (const schedule of responseJSON.schedules) {
        !this.schedules.has(schedule.id)) && await addSchedule(schedule);
    }
}
```

Výpis kódu 3.16 Funkce pro zpracování dat

3.3.3 Příprava aplikace na přidání dalších externích zdrojů

Jedním z nefunkčních požadavků je možná rozšiřitelnost. Proto se budu v této podsekci zabývat přípravou kódu a zajištěním co největší udržitelnosti a znovupoužitelnosti.

Jako první jsem se zaměřil na ukládání identifikátorů entit. Místo předešlého vytvoření objektů s identifikátory přímo ve třídě jsem vytvořil novou třídu `Index`, která bude tyto identifikátory spravovat a zajistí také jejich získání z databáze. Díky tomu nemusím ukládat identifikátory do objektů přímo ve třídě scraperu, ale mohu použít tuhle třídu. Část této třídy lze vidět v kódu 3.17.

```
export default class Index {
    public schedules: Set<number> = new Set();
    public events: Record<number, number> = {};
    ...

    public async init(sourceId: string) {
        const schedules = fetch(API_URL + "/schedules");
        const events = fetch(API_URL + "/events");
        ...
        const schedulesJSON = await schedules.then(res => res.json());
        ...
        schedulesJSON.map((schedule: any) => {
            !!schedule[sourceId]) && this.schedules.add(schedule[sourceId]);
        });
        ...
    }
}
```

Výpis kódu 3.17 Části z třídy `Index`

Dalším krokem bylo vytvoření abstraktní třídy `Scraper` pro sdílenou logiku scraperů. V této třídě je především instance zmiňované třídy `Index`. Kontrolu na duplicitu bude provádět každý vytvořený scraper, proto jsem vytvořil instanci této třídy rovnou v abstraktní třídě. Také jsem vytvořil základní konfiguraci, kterou bych jinak musel vytvářet v každé třídě opakovaně. Pokud by nějaký scraper potřeboval jinou konfiguraci, tato instance jde pomocí svých metod modifikovat. Jako poslední je v této třídě abstraktní metoda `run`, kterou bude muset implementovat každý dědící scraper. Implementaci této třídy lze vidět v kódu 3.18.

```
export default abstract class Scraper {
    protected readonly scraperConfig = new Configuration({
    persistStorage: false,
    });
    protected readonly index = new Index();

    public abstract run(): Promise<void>;
}
```

Výpis kódu 3.18 Implementace abstraktní třídy `Scraper`

Tyto podniknuté kroky mi ulehčí rozšíření aplikace o další externí zdroje a udělají kód udržitelnější.

Kapitola 4
Uživatelské testování

Uživatelské testování je nedílnou součástí vývoje aplikace a také klíčovým nástrojem pro zajištění vysoké úrovně použitelnosti a uživatelské spokojenosti. Je to druh experimentu, jehož cílem je nalezení problémů, které mohou při používání nastat. Provádí se za pomocí testovacích uživatelů, které postupně provádíme aplikací nebo testovanou funkcí. [29]

Nejčastěji mezi nalezené problémy aplikace patří [29]:

špatně nazvané prvky: uživatel neví, co se pod pojmem skrývá

špatně dohledatelné informace: informace, kterou uživatel hledá, je jinde, než kde ji očekává

hůře dostupné elementy: uživatel má potíže s kliknutím na daný element

neodpovídající obsah: uživatel očekává jiné informace, než jaké na stránce doopravdy jsou

prvky se chovají nestandardně

4.1 Výběr metody uživatelského testování

Uživatelské testování může být moderované a nemoderované. Při moderovaném testování prochází respondent aplikaci podle předem připraveného scénáře. Tímto procesem ho provádí „moderátor“, kterému jednotlivé kroky scénáře komentuje. Moderátor se zároveň může doptávat na doplňující informace, čímž se může dostat k přesnějšímu výsledku. V zájmu kvality a objektivity testu je důležité, aby nebyly použity otázky, které by mohly uživatele navádět k určité odpovědi. [30]

Naopak u nemoderovaného testování moderátor nefiguruje a uživatel scénáře prochází sám. Výhodou tak může být časová nenáročnost, jelikož testování nevyžaduje moderátorovu přítomnost. Kvůli absenci moderátora není možnost klást uživateli dodatečné otázky, je proto důležité mít scénář dobře připravený a zaměřený přesně na to, co chci zjistit. Tato metoda testování se hodí, jakmile je potřeba rychle otestovat základní funkčnosti aplikace bez hlubšího vzhledu uživatele. [31]

Já si vybral moderované testování, jelikož při něm není potřeba velký testovací vzorek a můžu se při něm doptávat na skutečné pochopení aplikace, které mě zajímá.

4.2 Účastníci testování

Na otestování aplikace jsem si vybral pět účastníku. Pro uživatelské testování je toto ideální počet, jelikož odhalí většinu problémů. U více uživatelů se dozvídáme méně nových informací a více se opakují již objevené problémy. [32] Nejdůležitější na správném výběru adeptů je, aby zapadali do cílové skupiny aplikace. Jinak bych mohl test provádět na lidech, kteří by o používání aplikace vůbec zájem neměli.

Byli vybráni následující uživatelé:

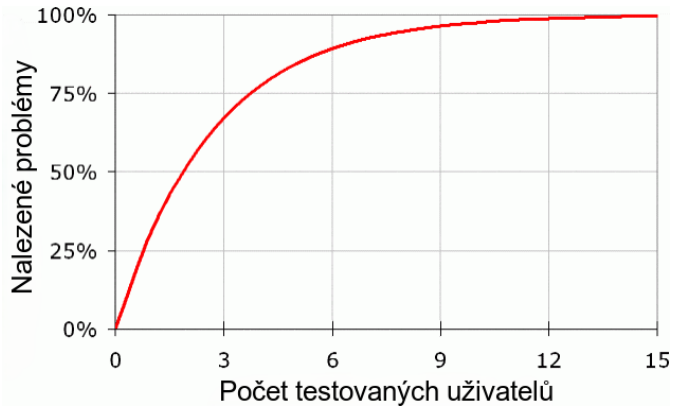
Julie M. 22 let, absolvent 2. LF UK, ráda chodí do divadel a na výstavy

Petr G. 22 let, student vysoké školy managementu, rád navštěvuje koncerty a hudební festivaly

Ondra V. 22 let, student technické vysoké školy, vyhledává sportovní aktivity

Lukáš Z. 22 let, student technické vysoké školy, účastní se koncertů a chodí do galerií

Michaela Č. 29 let, pracuje v oboru UI/UX design, často navštěvuje společenské události



Obrázek 4.1 Nielsenova křivka použitelnosti, která zachycuje vztah mezi počtem testovaných uživatelů a nalezenými problémy. [32]

4.3 Testovací scénáře

Abych docílil co nejpřesnějších výsledků, rozhodl jsem se rozdělit testování do několika krátkých a rychlých scénářů, které dohromady otestují všechny funkce aplikace. Po každém dokončeném testu proběhne krátký dialog, ve kterém se budu testovaných uživatelů ptát, jestli pro ně bylo vše jasné a srozumitelné. Na konci budu účastníkům klást otázky, jak na ně působí vizuální stránka aplikace.

Test lokalizace

Tento test má za cíl vyzkoušet, zda uživatelé chápou, jak funguje funkce lokalizace a jak ji spustit. Mimo jiné testuje schopnost základní orientace na mapě.

Průběh:

1. Spustit lokalizaci.
2. Najít jakoukoliv nejbližší událost.

Test by mě měl ujistit, zda platí:

- Umístění tlačítka je viditelné.
- Uživatel pomocí lokalizace ví, kde se nachází.
- Ovládání mapy funguje dle očekávání.

Test získání informací

Test volně navazuje na předchozí testování. Snažím se otestovat, zda je zřejmé, jak zjistit více informací o události.

Průběh:

1. Otevřít souhrn informací o události.
2. Otevřít detail události.
3. Otevřít jeden ze zdrojových webů.

Výsledkem úspěšného testu by mělo být:

- Uživatel ví, jak otevřít okno se souhrnem.
- Uživatel ví, jak otevřít detail.
- Uživatel dokáže jednoduše najít cestu ke zdrojovému webu.

Test filtrace

Cílem tohoto testu je ověření správného umístění a fungování filtrace akcí. Především je potřeba se ujistit, že uživatelé ví, jak filtr aktivovat/deaktivovat.

Průběh:

1. Otevřít filtraci.
2. Vyhledat pouze koncerty.
3. Vyhledat koncerty a festivaly.
4. Vyhledat filmy.

Tento test by mi měl zajistit následující odpovědi:

- Uživatel ví, jak otevřít nastavení filtrace.
- Ovládaní filtrace je intuitivní.
- Postup nastavení filtrace je očekávané.

Test vyhledávání

Pro tento test bude cílem zjistit správné pochopení vyhledávání míst a používání našeptávání.

Průběh:

1. Vyhledat Kino Lucerna.
2. Otevřít jakoukoli akci na tomto místě.

Po testu bych měl vědět, jestli platí:

- Umístění vyhledávání je viditelné.
- Použití našeptávání funguje dle očekávání.
- Uživatel po vyhledání dokázal najít zamýšlené místo.

4.4 Vyhodnocení

S každým testovaným uživatelem jsem strávil přibližně čtvrt hodiny včetně závěrečného dialogu. Testování proběhlo u všech testovaných v pořádku. Každý věděl, co má v každém kroku dělat a všem se podařilo testování dokončit. Při testování jsem se soustředil na uživatelovu orientaci v aplikaci a jak rychle dokáže plnit dané úkoly. To jsou pro mě totiž hlavní indikátory správně navrženého rozhraní.

Test lokalizace proběhl bez větších překážek. U spuštění lokalizace měl jeden z testovaných problém se správnou interpretací ikonky reprezentující tlačítko lokalizace. Ikonka mu nepřišla povědomá a proto nevěděl, zdali to je opravdu to správné tlačítko. Tento problém je však snadno odstranitelný nahrazením ikonky za jinou, která lokalizaci více vystihuje. Při následné lokalizaci se každý testující dokázal ve svém okolí zorientovat a najít nejbližší událost. Během závěrečného dialogu mi bylo však sděleno, že by uživatelům mohla chybět informace o přesné aktuální poloze. Přesná lokalizace uživatele může být vhodná, jakmile chce vědět, co se děje například v ulici, ve které se nachází. Tento nedostatek by bylo možné odstranit přidáním značky, která by identifikovala uživatelovu polohu. Při posledním úkolu nalezení nejbližší události nedošlo k žádnému problému a všichni zúčastnění dokázali s mapou zacházet.

U testu získání informací byly také všichni aktéři úspěšní. V prvním kroku se nevyskytly žádné potíže. Testovaní uživatelé rychle pochopili, jak se k souhrnu dostat, což pro mě bylo důležité. U otevření detailu akce také nebyl nikdo, kdo by tento úkol nedokázal splnit. Jediným problémem během testování na mobilu bylo to, že testovaný uživatel očekával otevření detailu po kliknutí na jakékoliv místo kartičky souhrnu. Implementace takového chování by mohla podobným uživatelům pomoct. Zároveň si myslím, že by nijak nesnížila použitelnost aplikace pro uživatele, kteří by detail otevírali pomocí tlačítka, jak je tomu nyní. Přemístění na zdrojový web události pro všechny testované nebyl náročný úkol a navíc ocenili ponechání otevřeného detailu po vrácení zpět do aplikace.

Při testování filtrace nedošlo k žádnému nedorozumění a nebyl nikdo, kdo by nějaký úkol nedokázal splnit. Někteří z testovaných ovšem brali ponechání otevřené filtrace po zvolení možnosti jako problémové chování. Prý by dle nich bylo lepší zavření výběru hned, jak zvolím nějaký filtr. Tento názor se vyskytl hlavně na mobilu, kde otevřená filtrace zabírá větší část obrazovky a blokuje uživatele od ovládání mapy. Způsob chování, které testovaní uživatelé nabízeli, mi však vhodný nepřijde. Jakmile by uživatel chtěl zvolit více možností filtru, musel by pokaždé znova otevírat filtraci. Dle mého názoru je momentální způsob řešení s jedním kliknutím navíc pro zavření méně zatěžující, než navržené řešení s vícero kliknutími navíc při volení více filtrů.

Poslední test proběhl bez jakýchkoli potíží. Při zvolení vyhledávání na mobilu byli testující příjemně překvapeni s automatickým zaměřením na vyhledávací lištu a brali to jako velké zjednodušení procesu. Během závěrečného dialogu mi jeden z testovaných uživatelů sdělil, že mu chybí

funkce vyhledávání adresy. Tato funkce by však byla nad rámec funkčních požadavků a vyžadovalo by to propojení další dodatečné služby s aplikací. Proto jsem se rozhodl toto rozšíření neimplementovat.

Uživatelské testování považuji za úspěšné. Povedlo se mi odhalit největší existující problémy a zároveň jsem se ujistil, že navržené rozhraní je použitelné a uživatelům přináší příjemný zážitek z používání aplikace. V následující podsekci popíši, jak jsem postupoval při implementaci úprav uživatelského rozhraní.

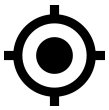
4.4.1 Implementace úprav

V této podsekci se zabývám implementací úprav, které jsem z uživatelského testování vyhodnotil jako vhodné. U každé úpravy uvedu stručný popis, co bylo potřeba pro provedení úpravy změnit.

4.4.1.1 Změnit ikonku lokalizace za více reprezentativní

Abych zaručil co nepravděpodobnější pochopení ikonky uživatelem, udělal jsem rešerši mapových aplikací a zaměřil se na jejich způsob zobrazení lokalizace. Od této inspirace si slibuji, že jakmile bude moje ikonka podobná těm, na které jsou uživatelé zvyklí a dobře je znají, sníží se tím šance na špatnou interpretaci. Inspiraci jsem si vzal z webových stránek mapy.cz a google.com/maps. Tyto dvě aplikace jsou dle mého názoru v České republice nejpoužívanější a měla by je tak většina uživatelů znát. Na obrázku 4.2 lze porovnat původní a novou ikonku pro lokalizaci.

(a) Původní ikonka lokalizace.

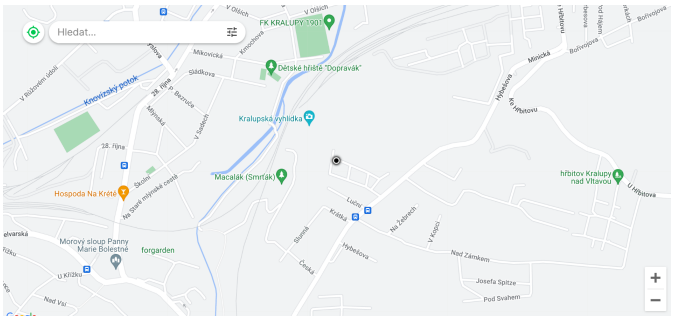


(b) Nová ikonka lokalizace.

Obrázek 4.2 Porovnání původní a nové ikonky lokalizace.

4.4.1.2 Označit aktuální polohu uživatele při aktivní lokalizaci

Pro tuhle funkci bylo potřeba poslat informaci o poloze z komponenty `LocationButton` do komponenty `Map`, ve které podle získané lokace proběhne zobrazení značky. To bylo zajištěno pomocí dodatečného parametru `afterGeolocation` v komponentě `LocationButton`. Tento parametr je funkce, která se zavolá po úspěšné lokalizaci. Jako argument je zde vložena uživatelova lokace. Díky tomu si pozici můžu následně uložit v rodičovské komponentě komponent `LocationButton` a `Map` do stavu. Uložený stav vložím jako parametr `userLocation` do komponenty `Map`, která na tuto lokaci zobrazí značku pro polohu uživatele. Výsledek je ukázán na obrázku 4.3.



Obrázek 4.3 Náhled označení lokace uživatele.

4.4.1.3 Otevření detailu po kliknutí na jakékoliv místo souhrnu

Zajistit tuhle funkcionalitu nebylo nic těžkého. Tlačítku na otevření detailu jsem odebral událost `onclick`. Zmíněnou událost jsem následně přesunul na celou komponentu kartičky souhrnu. Teď se detail akce otevře pokaždé, co uživatel klikne na souhrn.

Závěr

Cílem práce bylo navrhnout a implementovat webovou aplikaci pro vyhledávání kulturních a společenských akcí v okolí tak, aby vyřešila nedostatky dnešních řešení a udělala pro uživatele vyhledávání událostí co nejjednodušší.

Svou práci jsem začal rešerší aktuálních řešení. Pro rešerši jsem si vybral tři nejpoužívanější aplikace s tímto zaměřením. U aplikací jsem zkoumal míru usnadnění vyhledávání událostí v okolí, jejich nedostatky a také, zda by bylo vhodné jejich využití jako externí zdroj dat. Z rešerše jsem vyhodnotil, že žádná z nejpoužívanějších aplikací neřeší vyhledávání událostí v okolí uspokojivě. Získané poznatky jsem následně využil při specifikaci funkčních a nefunkčních požadavků.

Následně jsem vyhotovil návrh řešení. Jako první jsem uskutečnil návrh uživatelského rozhraní. Při tvorbě jsem kladl důraz na vyřešení problémů existujících řešení a zároveň jsem se snažil zahrnout jejich kladné vlastnosti. Poté následoval výběr databáze společně s návrhem databázového modelu a jejím nasazením. Výsledkem bylo použití klasické relační databáze nasazenou jako bezserverová služba. Po návrhu databáze pokračovalo seznámení s použitými technologiemi k implementaci řešení.

Podle návrhu a požadavků jsem provedl implementaci řešení. Nejdříve jsem vytvořil a nasadil databázi. Poté jsem zhotovil implementaci hlavní webové aplikace, kterou jsem následně také nasadil. Dále jsem implementoval aplikaci pro získání dat z externích zdrojů. Díky vhodnému výběru technologií jsem ušetřil spoustu času a dokázal jsem napsat přehledný a udržitelný kód.

Po implementaci aplikace jsem ji podrobil uživatelskému testování. Nejdříve jsem zvážil metody uživatelského testování, ze kterých jsem následně jednu vybral. Poté jsem určil účastníky testování a definoval testovací scénáře. Výsledkem bylo moderované testování na pěti subjektech. Z testování jsem provedl vyhodnocení a navrhl možné úpravy aplikace, které jsem i implementoval.

Hlavní cíl, usnadnit vyhledávání akcí v okolí, byl splněn. To jsem si ověřil i díky uživatelskému testování. Aplikace splňuje všechny funkční a nefunkční požadavky kromě jedné části z filtrace z důvodu nedostatku času. Tento nedostatek je však lehko vyřešitelný a zmíním se o něm v sekci o budoucím rozšíření. Aplikaci vnímám jako konkurenceschopnou ostatním existujícím řešením a již ji používá několik lidí z mého okolí. Aplikace je veřejně dostupná na adrese `https://nearby.vercel.app`.

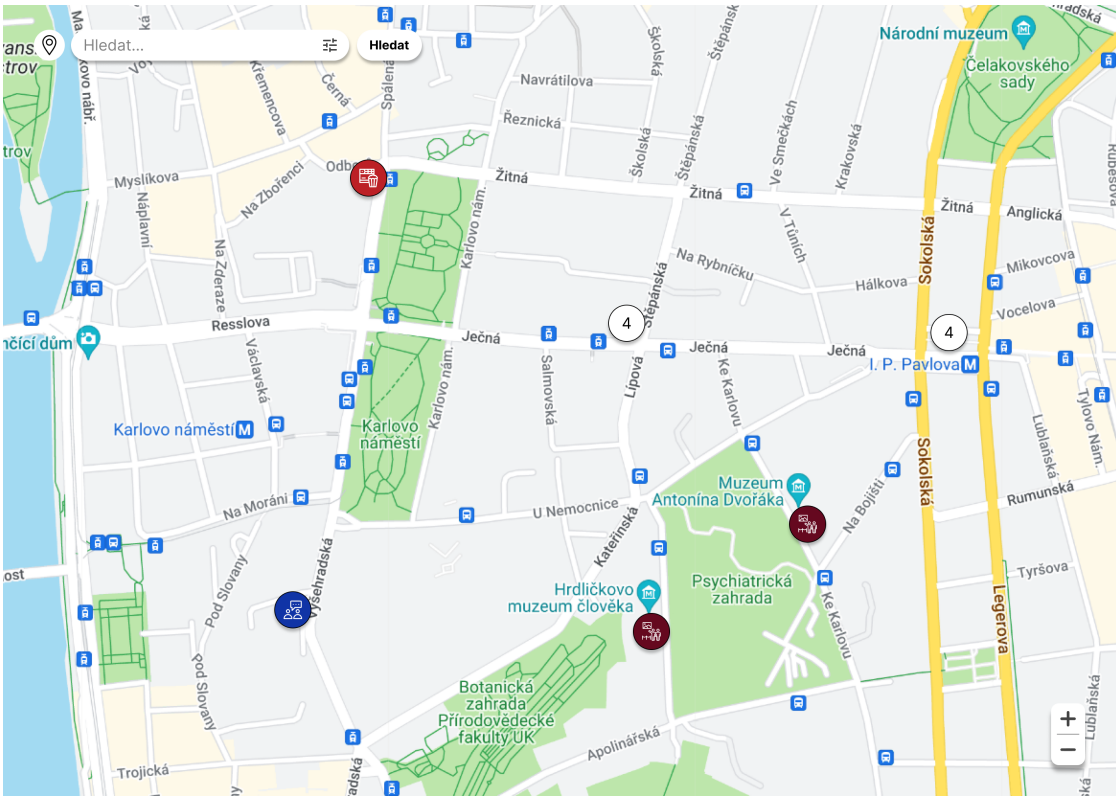
Budoucí rozšíření

Aplikace má velký potenciál na rozšíření. Při budoucím vývoji bych jako první přidal funkci vyfiltrování právě konaných akcí, které jsem měl ve funkčních požadavcích. Přidávání filtrů do aplikace je jednoduché, stačí pouze vytvořit dotaz pomocí knihovny Drizzle a připravit uživatelské rozhraní. Filtrační možnosti tak nemusí zůstat jen u těchto a lze aplikaci rozšířit o další.

Další možné rozšíření je přidání více externích zdrojů pro získávání událostí. Při implementaci jsem kladl velký důraz na jednoduché přidání, aplikace je tak na tuhle možnost připravena.

Příloha A

Návrh uživatelského rozhraní



Obrázek A.1 Hlavní obrazovka s mapou

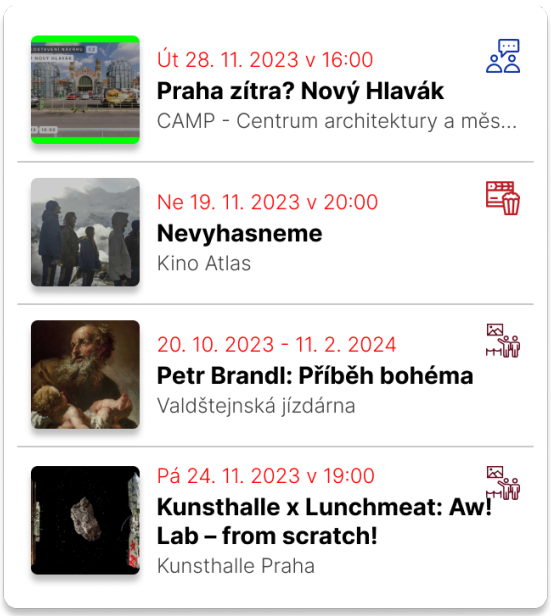
Obrázek A.2 Detail události

Obrázek A.3 Detail události pro menší obrazovky

(a) Větší zobrazení

(b) Menší zobrazení

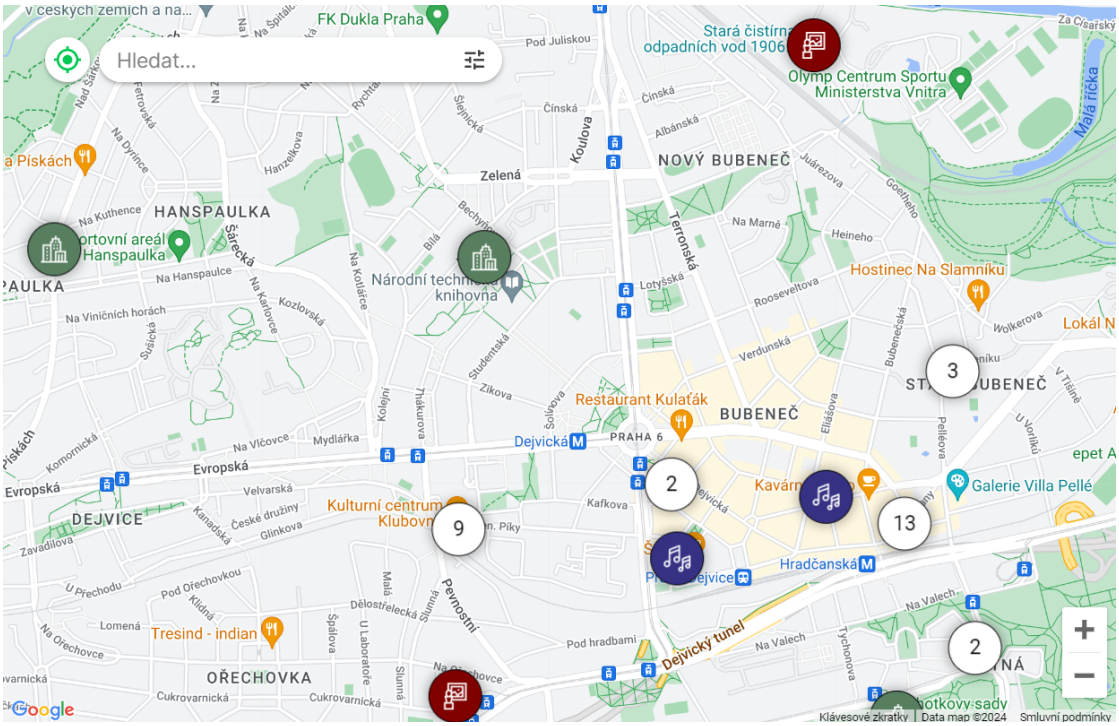
Obrázek A.4 Souhrn události



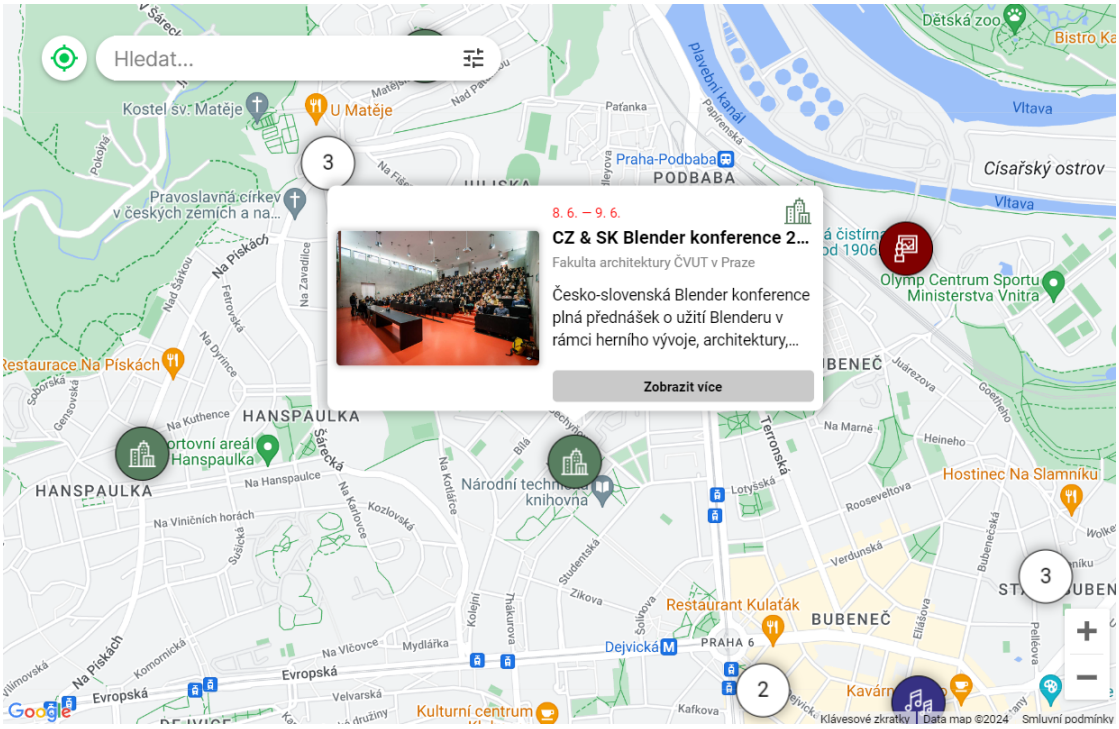
Obrázek A.5 Výpis událostí na místě

Příloha B

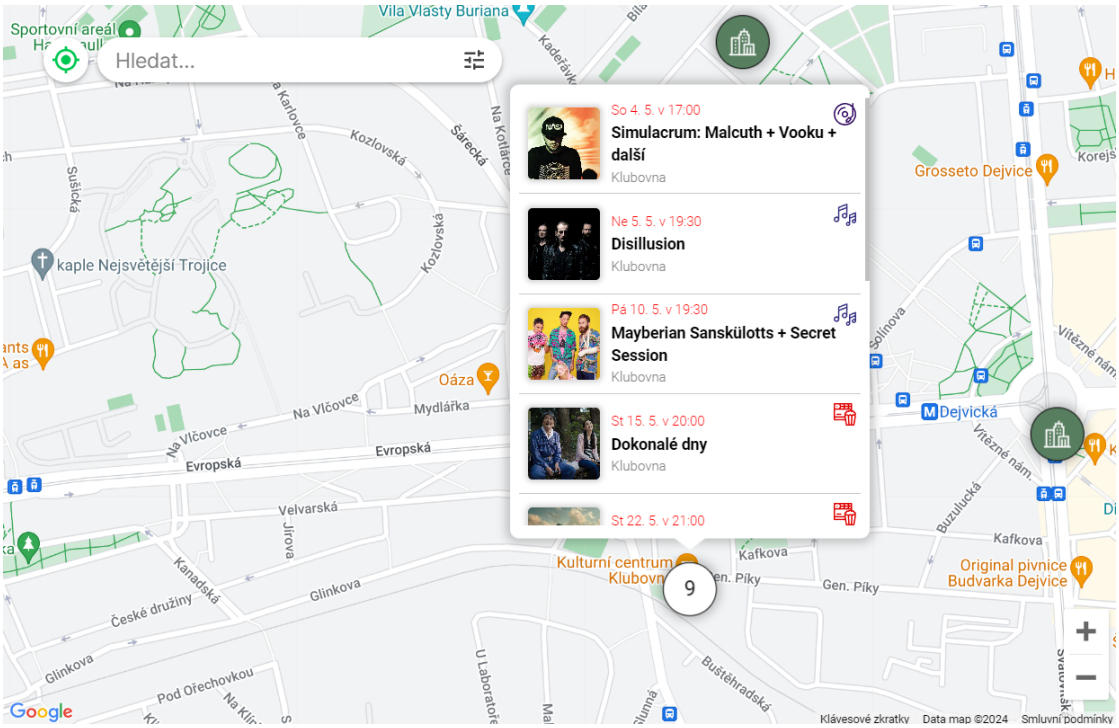
Výsledný vzhled aplikace



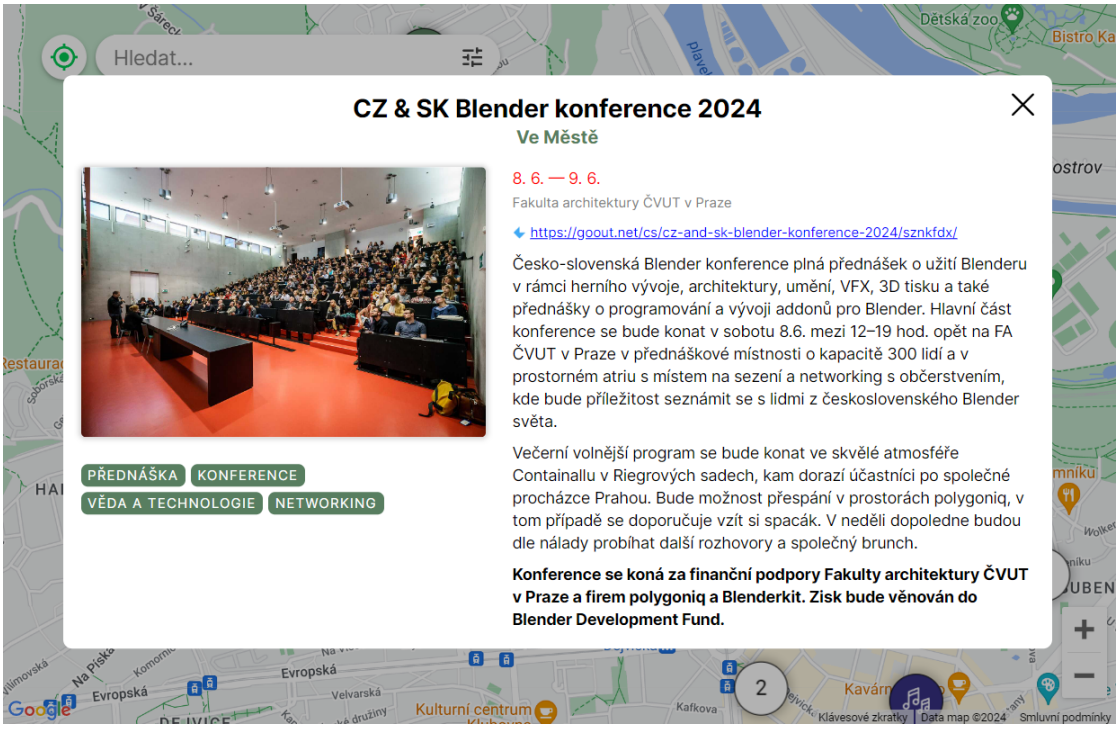
Obrázek B.1 Hlavní obrazovka s mapou



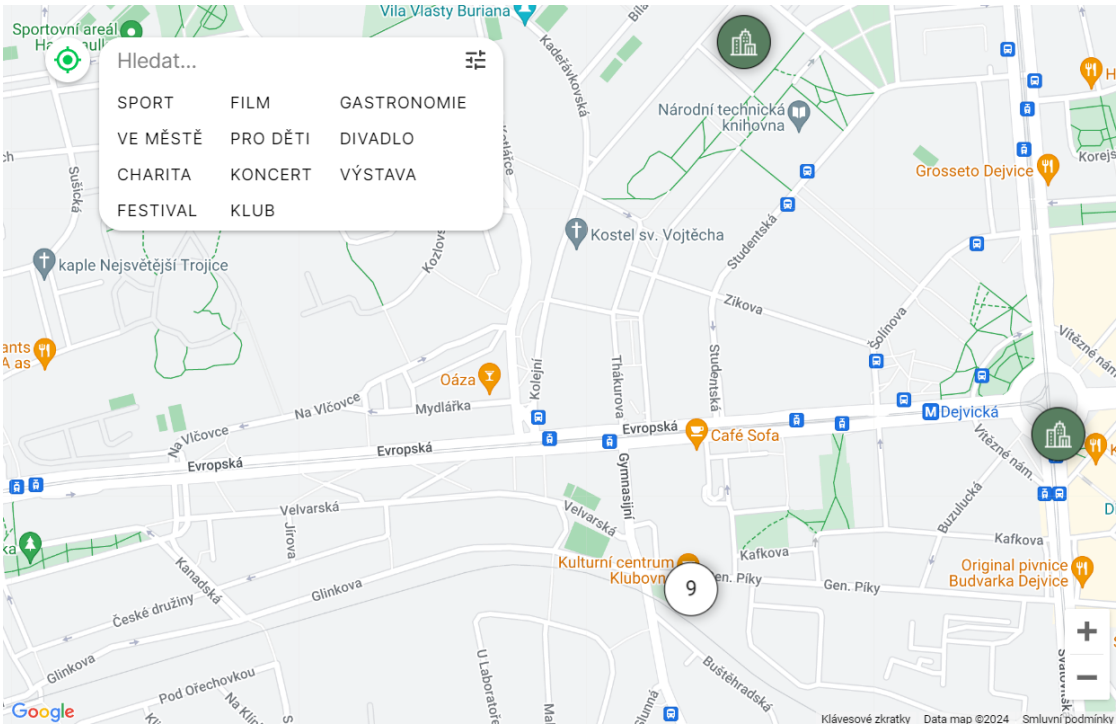
Obrázek B.2 Souhrn události



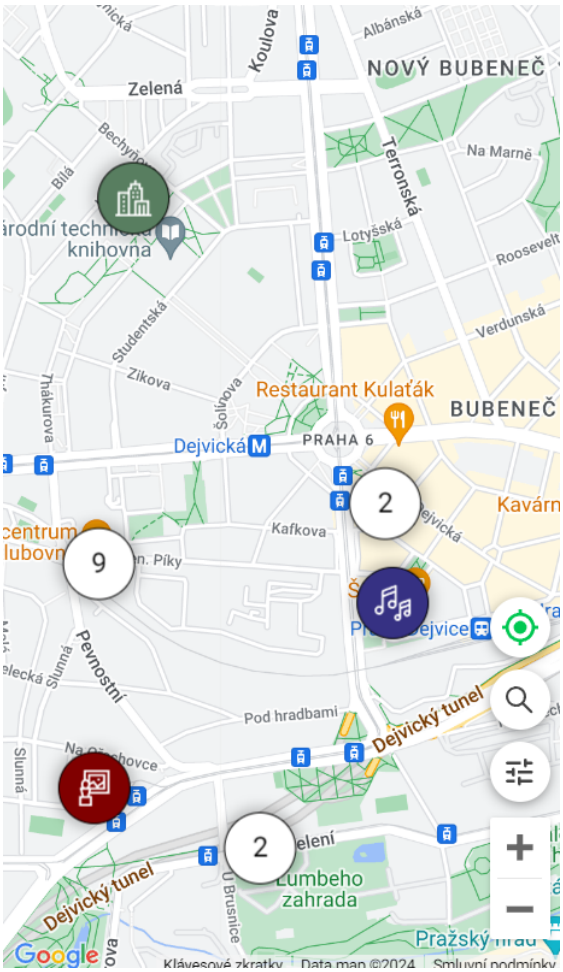
Obrázek B.3 Výpis událostí na místě



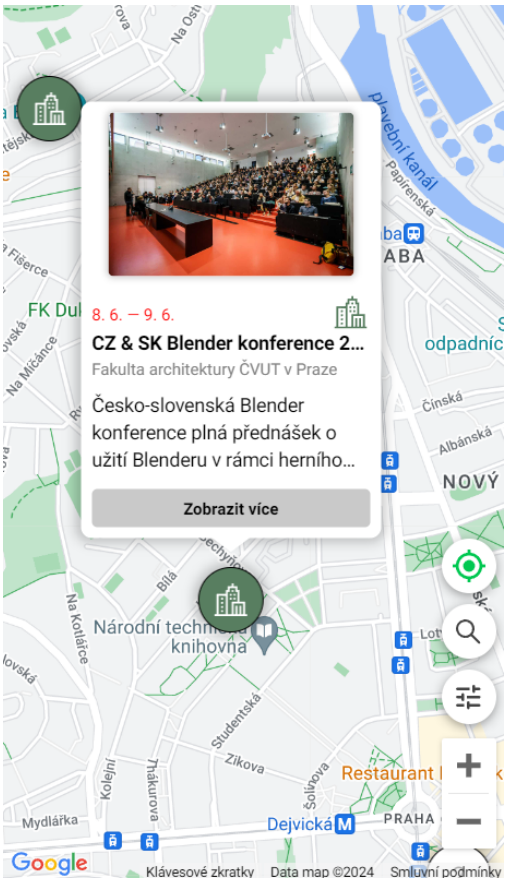
Obrázek B.4 Detail události



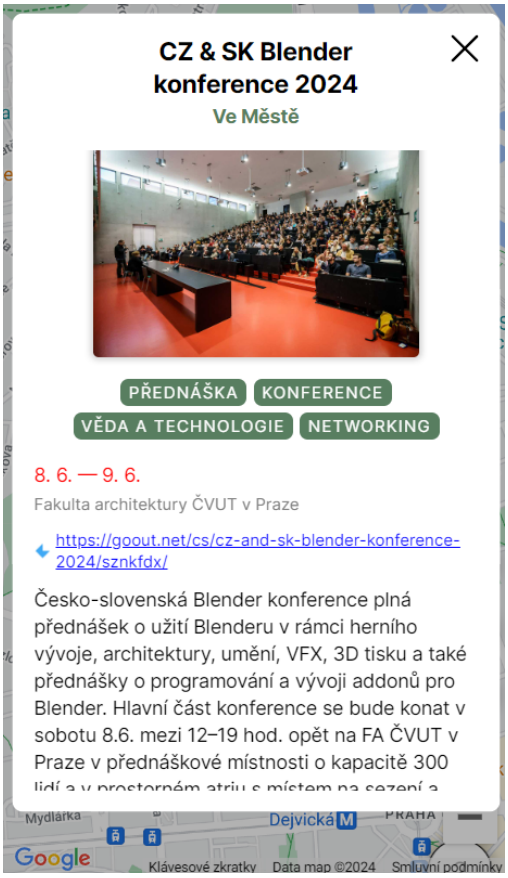
Obrázek B.5 Filtrace událostí



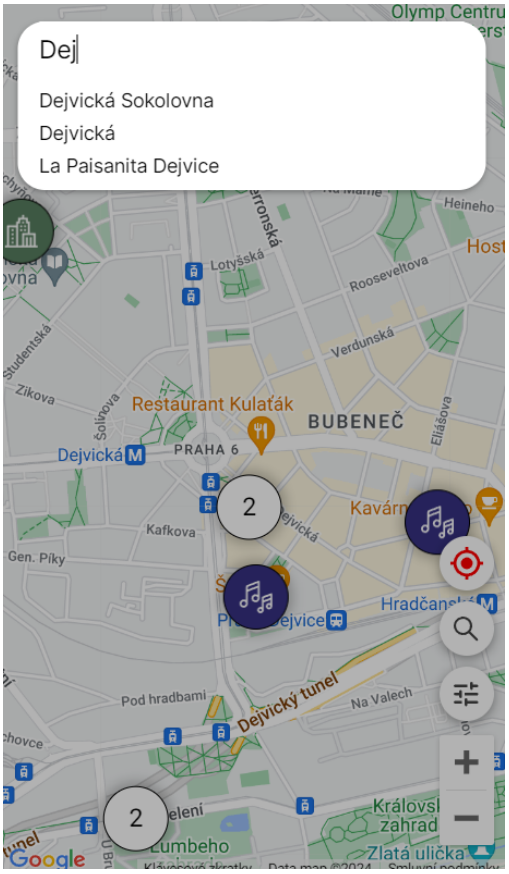
Obrázek B.6 Hlavní obrazovka s mapou



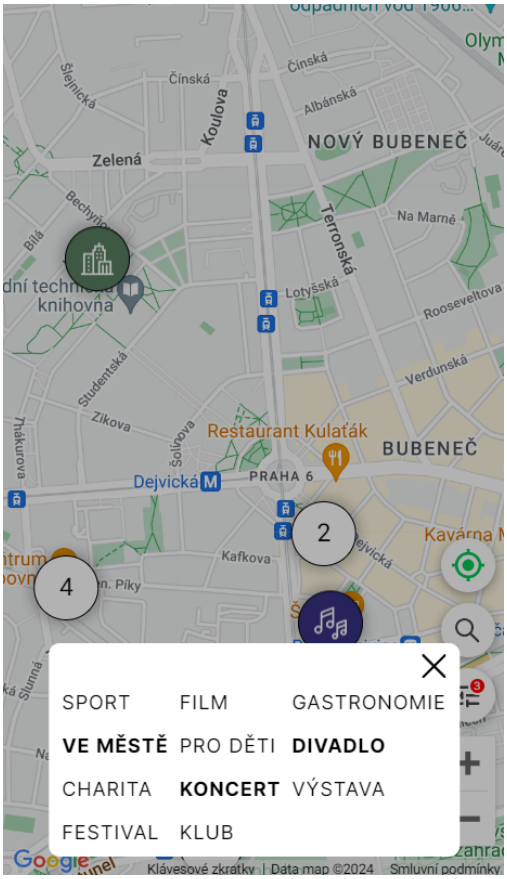
Obrázek B.7 Souhrn události



Obrázek B.8 Detail události



Obrázek B.9 Vyhledávání událostí



Obrázek B.10 Filtrace událostí

Bibliografie

1. *Výsledky – Křišťálová Lupa 2019* [online]. Křišťálová Lupa, 2019 [cit. 2024-04-04]. Dostupné z: https://kristalova.lupa.cz/2019/vysledky.
2. *GoOut* [online]. StartupJobs, 2024 [cit. 2024-04-04]. Dostupné z: https://www.startupjobs.cz/startup/goout-s-r-o.
3. *GoOut* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://goout.net/cs/cesko/akce/lezfymfti/.
4. *O Kudy z nudy* [online]. CzechTourism, 2024 [cit. 2024-04-04]. Dostupné z: https://www.kudyznudy.cz/o-kudy-z-nudy.
5. *Kudy z nudy* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://www.kudyznudy.cz/kalendar-akci.
6. KEMP, Simon. *Digital 2023 October Global Statshot Report* [online]. DataReportal, 2023 [cit. 2024-04-06]. Dostupné z: https://datareportal.com/reports/digital-2023-october-global-statshot.
7. *Facebook Events* [online]. 2024. [cit. 2024-04-04]. Dostupné z: https://www.facebook.com/events.
8. *What are Functional Requirements: Examples, Definition, Complete Guide* [online]. Visure Solutions, 2024 [cit. 2024-04-18]. Dostupné z: https://visuresolutions.com/blog/functional-requirements/.
9. STEVENS, Emily. *7 fundamental UX design principles all designers should know* [online]. UX Design Institute, 2022 [cit. 2024-05-12]. Dostupné z: https://www.uxdesigninstitute.com/blog/ux-design-principles/.
10. ANDERSON, Benjamin. *SQL vs. NoSQL Databases: What's the Difference?* [online]. IBM, 2022 [cit. 2024-04-21]. Dostupné z: https://www.ibm.com/blog/sql-vs-nosql/.
11. GIENOW, Michelle. *What is a serverless SQL database?* [online]. 2021. [cit. 2024-04-24]. Dostupné z: https://www.cockroachlabs.com/blog/what-is-serverless-sql/.
12. *TypeScript for JavaScript Programmers* [online]. Microsoft, 2024 [cit. 2024-04-19]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.
13. *Quick Start – React* [online]. Meta, 2024 [cit. 2024-04-19]. Dostupné z: https://react.dev/learn.
14. *JSX* [online]. Meta, 2022 [cit. 2024-04-19]. Dostupné z: https://facebook.github.io/jsx/.

15. *JavaScript in JSX with Curly Braces* [online]. Meta, 2024 [cit. 2024-04-24]. Dostupné z: https://react.dev/learn/javascript-in-jsx-with-curly-braces.
16. *Writing Markup with JSX* [online]. Meta, 2024 [cit. 2024-04-24]. Dostupné z: https://react.dev/learn/writing-markup-with-jsx.
17. *Docs | Next.js* [online]. Vercel, 2024 [cit. 2024-05-09]. Dostupné z: https://nextjs.org/docs.
18. *Building Your Application: Routing | Next.js* [online]. Vercel, 2024 [cit. 2024-05-09]. Dostupné z: https://nextjs.org/docs/app/building-your-application/routing.
19. *Drizzle ORM - Overview* [online]. Drizzle, 2024 [cit. 2024-05-09]. Dostupné z: https://orm.drizzle.team/docs/overview.
20. *Introduction | Crawlee* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://crawlee.dev/docs/introduction.
21. *CheerioCrawler | API | Crawlee* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://crawlee.dev/api/cheerio-crawler/class/CheerioCrawler.
22. *PuppeteerCrawler | API | Crawlee* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://crawlee.dev/api/puppeteer-crawler/class/PuppeteerCrawler.
23. *PlaywrightCrawler | API | Crawlee* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://crawlee.dev/api/playwright-crawler/class/PlaywrightCrawler.
24. *React Hooks for Data Fetching – SWR* [online]. 2023. [cit. 2024-04-20]. Dostupné z: https://swr.vercel.app.
25. AGAFONKIN, Vladimir. *Clustering millions of points on a map with Supercluster* [online]. Medium, 2016 [cit. 2024-04-19]. Dostupné z: https://blog.mapbox.com/clustering-millions-of-points-on-a-map-with-supercluster-272046ec5c97.
26. *Overview | Maps JavaScript API | Google for Developers* [online]. Google, 2024 [cit. 2024-04-22]. Dostupné z: https://developers.google.com/maps/documentation/javascript/overview.
27. *Introduction | React Google Maps* [online]. GitHub, 2024 [cit. 2024-04-22]. Dostupné z: https://visgl.github.io/react-google-maps/docs.
28. *Vercel Postgres: Scalable SQL for the web* [online]. Vercel, 2024 [cit. 2024-04-25]. Dostupné z: https://vercel.com/storage/postgres.
29. VOJÁK, Michal. *Jak dělat uživatelské testování* [online]. Designdev, 2020 [cit. 2024-04-08]. Dostupné z: https://designdev.cz/jak-delat-uzivatelske-testovani.
30. JANÁSEK, Robin. *Jak na uživatelské testování: metody výzkumu a postupy* [online]. Proof & Reason, 2022 [cit. 2024-04-26]. Dostupné z: https://www.proofreason.com/blog/metody-uzivatelskeho-testovani.
31. JANÁSEK, Robin. *Co je to nemoderované uživatelské testování a jak jej u návrhů webů či eshopů využijete* [online]. Proof & Reason, 2023 [cit. 2024-04-27]. Dostupné z: https://www.proofreason.com/blog/jak-vyuzit-nemoderovane-testovani.
32. NIELSEN, Jakob. *Why You Only Need to Test with 5 Users* [online]. NN Group, 2000 [cit. 2024-04-08]. Dostupné z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

Obsah příloh