VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO RYCHLÉ VYHLEDÁVÁNÍ LETOVÝCH SPOJENÍ
TOOL FOR QUICK FLIGHT SEARCH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Matúš Muránsky**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Číka, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Matúš Muránsky ***ID:*** 154813
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Nástroj pro rychlé vyhledávání letových spojení

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vývoj nástroje pro jednoduché vyhledávání letových spojů s následnou možností notifikace v případě vzniku změn. Student analyzuje a porovná dostupné platformy a technologie vyhledávání využívané při vývoji mobilních a webových aplikací vhodné pro vyvíjený nástroj. Klíčové parametry vyvíjeného nástroje jsou rychlost, dostupnost na variantních hardwarových platformách a možnost provozovat aplikaci i při omezeném nebo žádném přístupu k Internetu. Vedle poskytování informací o letovém provozu bude nástroj poskytovat informace o počasí a zajímavostech ve vyhledávaných destinacích a jiné. Student dále analyzuje případy použití nástroje za účelem návrhu vhodného uživatelského prostředí. Podle výsledků analýz a návrhu bude nástroj implementován, otestován a nasazen do ostrého provozu.

DOPORUČENÁ LITERATURA:

[1] NIEDERST ROBBINS, Jennifer. Learning web design: a beginner's guide to HTML, CCS, JavaScript, and web graphics. Fourth edition. Beijing: O'Reilly, [2012]. ISBN 978-1-449-31927-4.

[2] ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Petr Číka, Ph.D.
Konzultant: Vladimír Hudec (pelicantravel.com s.r.o.)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Diplomová práca sa zaoberá návrhom a vývojom nástroja pre jednoduché vyhľadávanie letových spojení pre firmu Pelikantravel.com s.r.o. Cieľom práce je vyvinúť nástroj, ktorý bude umožňovať prevádzku pri obmedzenom internetovom pripojení a bude podporovať spätnú notifikáciu používateľa. V prvej časti práce je popísaná analýza vhodných technológii a platforiem pre výsledný nástroj. Praktická časť sa venuje problematike používateľského zážitku a s ním spojeným návrhom používateľského rozhrania. V rámci tejto časti je popísaná základná funkcionalita nástroja podľa analýzy prípadu použitia. Ďalej práca bližšie popisuje realizáciu celého nástroja a v závere sa venuje testovaniu výsledného nástroja.

Kľúčové slová

PWA, UX, UI, HTML, CSS, JS, React, Rebass, Styled-system, Styled-components

Abstract

The thesis deals with the design and development of a tool for easy search of flight connections for Pelikantravel.com s.r.o. The aim of this work is to develop a tool that will allow operation in a limited Internet connection and will support user feedback. The first part of the thesis describes the analysis of suitable technologies and platforms for the resulting tool. The practical part deals with the issue of user experience and with the related user interface design. In this section, the basic functionality of the tool is described according to the use case analysis. Furthermore, the work describes in detail the implementation of the whole tool and in the end is devoted to testing the resulting tool.

Key words

PWA, UX, UI, HTML, CSS, JS, React, Rebass, Styled-system, Styled-components

MURÁNSKY, Matúš. *Nástroj pro rychlé vyhledávání letových spojení.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/113471. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Petr Číka.

PREHLÁSENIE

Prehlasujem, že som svoju diplomovú prácu na tému „Nástroj pre rýchle vyhľadávanie letových spojení“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia §11 nasledujúcich autorského zákona č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. Diel 4 Trestného zákonníka č.40/2009 Sb.

Brno ....

Podpis autora

POĎAKOVANIE

Veľmi rád by som poďakoval vedúcemu mojej diplomovej práce pánovi doc. Ing. Petrovi Číkovi, Ph.D. za odborné vedenie, konzultácie, veľkú trpezlivosť a podnetné návrhy k práci. Zároveň by som sa rovnako rád poďakoval pánovi Ing. Vladimírovi Hudecovi za konzultácie a poskytnutú šancu pracovať na projekte pre firmu Pelikantravel.com s.r.o.

Brno ..... ……....

podpis autora

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V súčasnej dobe sledujeme pomerne rastúci trend, kedy používatelia pristupujú k aplikáciám a ich službám čoraz viac z mobilných zariadení. To sa týka aj odvetvia cestovného ruchu, kde sa pri vývoji aplikácií kladie čoraz väčší dôraz na dostupnosť aplikácie. Je teda dôležité, aby sa pri vývoji aplikácie dbalo na poskytovanie rovnakého UX (*používateľský zážitok*) bez ohľadu na to, aké zariadenie používateľ používa.

Táto diplomová práca sa bude zaoberať vývojom aplikácie pre firmu Pelikantravel.com s.r.o., ktorá bude umožňovať jednoduché vyhľadávanie letových spojení s integráciou ďalších dát o destinácii (*počasie a ďalšie rôzne fakty a zaujímavosti o destinácii*). Od výslednej aplikácie sa očakáva tiež schopnosť prevádzky pri obmedzenom, alebo žiadnom pripojení na internet a možnosť notifikácie používateľa pri vzniku zmien. Diplomová práca sa bude špeciálne venovať problematike návrhu a realizácie UI (*používateľské rozhranie*) a zabezpečenia kvalitného UX.

Pred samotným vývojom bude potrebná analýza súčasných platforiem a technológií, ktoré sa v súčasnosti využívajú na vývoj klientských aplikácií. Analýza platforiem, ktoré sa používajú pri vývoji klientských aplikácií je popísaná v kapitole 2 a kapitole 3. Nasledujúca kapitola 4 približuje a v závere porovnáva popredné technológie, ktoré sa využívajú pri vývoji klientských aplikácií. V kapitole 5 sú rozobraté požiadavky na vyvíjaný nástroj, a zároveň je tu priblížená analýza prípadu použitia. Návrh a vývoj UI a celej klientskej časti aplikácie je podrobne rozobratý v kapitole 6. Za touto časťou nasleduje posledná kapitola 7, ktorá sa zaoberá testovaním výslednej aplikácie, zhodnotením výsledkov a načrtnutím ďalšieho smerovania práce.

1 Vhodné platformy pre vývoj aplikácie

Ako prvý krok bude potrebné vybrať vhodnú platformu, na ktorej bude výsledná aplikácia postavená. V dobe písania tejto diplomovej práce sú najpoužívanejšími platformami webové, mobilné a pomerne nové tzv. PWA (*Progresívna webová aplikácia*).

Všetky spomínané platformy využívajú v princípe rovnakú komunikačnú technológiu a tou je HTTP (*HyperText Transfer Protocol*). Jedná sa o aplikačný protokol, ktorý umožňuje komunikáciu dát naprieč službou WWW (*World Wide Web*). V tejto kapitole budú bližšie rozobraté a popísané práve vyššie spomínané platformy.

1.1 Webové aplikácie

Webové aplikácie, ako aj mobilné, využívajú službu World Wide Web, kde pomocou protokolu HTTP manipulujú s HTML (*HyperText Markup Language*) dokumentami. V princípe ide o klasickú komunikáciu medzi klientom a serverom, kedy sa klient pomocou webového prehliadača dotazuje na server pomocou URL (*Uniform Resource Locator*) adresy a následne mu server poskytne odpoveď v podobe HTML dokumentu. V porovnaní s mobilnými aplikáciami, webové aplikácie využívajú na svoju prevádzku webový prehliadač. Keďže využívajú webový prehliadač, a teda službu WWW, potrebujú na svoje bezprostredné fungovanie nepretržité pripojenie k sieti Internet, pretože bez pripojenia k Internetu sa klient nedokáže dotazovať na server. [1]

Z hľadiska vývoja je webová aplikácia kombináciou súboru skriptov na strane servera (*php, node.js a pod.*), ktoré obstarávajú ukladanie a prístup k dátam v databáze a skriptov na strane klienta (*JavaScript*) a HTML dokumentov, ktoré prezentujú dáta používateľovi a odosielajú dáta a dotazy na server. Vývoj webových aplikácií je teda možné rozdeliť do dvoch častí. Jednou je vývoj tzv. frontend-u, alebo inak povedané prezentačnej vrstvy na strane klienta. Ten sa realizuje v jazykoch, ktoré rozoznáva webový prehliadač (*JavaScript, HTML*). Druhou časťou je vývoj serverovej časti tzv. backend. Tá sa realizuje v jazykoch ako napríklad PHP, Python, Ruby a iné.

1.2 Mobilné aplikácie

Mobilné aplikácie ako svojim názvom napovedajú, sú aplikácie vyvinuté pre prevádzku na mobilných zariadeniach (*mobilné telefóny a tablety*). Aplikácie majú pri tom k dispozícii všetky kapacity mobilného zariadenia, a teda nie len kapacity webového prehliadača, ako je

to u webových aplikácií. Kategóriu mobilných aplikácií môžeme rozdeliť do troch podkategórií:

1. **Natívne aplikácie**:

   Natívne aplikácie využívajú konkrétnu mobilnú platformu, respektíve jej operačný systém, ako napríklad Android, iOS a pod. Natívne aplikácie sú teda väčšinou vyvíjané v jazykoch, ktoré sú akceptované danou platformou (*Swift prípadne Objective C pre iOS a Java, alebo Kotlin pre Android*). Rovnako tiež sa pri ich vývoji používa špecifické IDE (i*ntegrované vývojové prostredie*) pre daný operačný systém, ako napríklad Android Studio pre Android aplikácie a XCode pre iOS aplikácie. Hlavnou výhodou natívnych aplikácií je, že poskytujú optimálny UX (*používateľský zážitok*). To znamená, že tým, že sú navrhované a vyvíjané pre konkrétnu platformu, poskytujú vysokú kvalitu prevádzky. Príkladom takéhoto typu aplikácií sú napríklad sociálne siete Instagram a Snapchat. [2]

   Výhody:

   - Rýchle a responzívne, nakoľko sú vyvíjané pre konkrétnu platformu.
   - Sú viac interaktívne, intuitívne v rámci funkcionalít ich mobilného používateľského rozhrania.

   Nevýhody:

   - Značne náročnejší a drahší vývoj v porovnaní s webovými aplikáciami.
   - Vývoj zaberie viac času, nakoľko musí byť aplikácia napísaná v rôznych jazykoch vzhľadom na rôzne platformy.

2. **Hybridné aplikácie**:

   Hybridné aplikácie sa vyvíjajú za účelom ich nasadenia na viaceré platformy. Teda môžu byť nasadené na obidvoch, iOS aj Android platformách. Vývoj hybridných mobilných aplikácií je podobný vývoju webových aplikácií. Oba typy aplikácií využívajú kombináciu rovnakých technológií, a síce HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) a JavaScript. Avšak na prevádzku nevyužívajú webový prehliadač, ale tzv. WebView, ktoré je realizované v rámci natívneho kontajnera. To im umožňuje pristupovať k hardware kapacitám mobilného zariadenia. Aktuálne prevažná väčšina hybridných mobilných aplikácií využíva platformu Apache Cordova, ktorá poskytuje set JavaScript API, ktoré umožňujú

prístup k hardware kapacitám cez plug-iny napísané v natívnom kóde. Príkladmi takýchto aplikácií sú napríklad MarketWatch, alebo TripCase. [2]

Výhody:

- Adaptácia na viacerých platformách.
- Zlúčený a menej nákladný vývoj.
- Kratšia doba potrebná na vývoj v porovnaní s natívnymi aplikáciami.

Nevýhody:

- Pomalšie, z dôvodu ich architektúry.

3. **Progresívne Webové Aplikácie**:

PWA (*Progresívna Webová Aplikácia*) je vo svojej podstate webová aplikácia, ktorá využíva kapacity moderných webových prehliadačov no zároveň poskytuje podobný používateľský zážitok ako to je u mobilných aplikácií. Dôležité je podotknúť, že progresívne webové aplikácie pri tom nevyžadujú od používateľa inštaláciu aplikácie zo známych App Store (*iOS*) alebo Google Play (*Android*). Pristupuje sa k nim, podobne ako u webových aplikácií, pomocou URL (*Unifrom Resource Locator*) pri čom používateľom mobilných zariadení je poskytnutá možnosť pripnúť ikonu aplikácie na plochu ich zariadenia, čím sa UX (*používateľský zážitok*) pri opätovnom používaní PWA približuje UX mobilnej aplikácie. Z technického hľadiska sú PWA tvorené technológiami HTML, CSS a JavaScript. Progresívne webové aplikácie využívajú napríklad AliExpress, Financial Times, NASA a iné. [2]

Výhody:

- Jednoduchšie na vývoj ako iné mobilné aplikácie.
- Jednoduchšie na údržbu v porovnaní s inými mobilnými aplikáciami.
- Umožňuje prevádzku pri obmedzenom, alebo žiadnom pripojení na internet.
- Jedna PWA dokáže obstáť rovnako kvalitne na všetkých platformách, nakoľko využíva na svoju prevádzku webový prehliadač.

Nevýhody:

- Na svoju prevádzku využívajú webový prehliadač.
- Nedisponujú až tak širokými kapacitami ako natívne aplikáciami.

2 Progresívne webové aplikácie

PWA (*Progresívna Webová Aplikácia*) kombinuje to najlepšie zo sveta webových a mobilných aplikácií. Na svoj chod nepotrebuje žiadnu inštaláciu, naopak stačí jej iba webový prehliadač. Medzi popredné vlastnosti patrí schopnosť pomerne rýchlej prevádzky pri slabom pripojení, offline režim, push notifikácie, možnosť uložiť odkaz aplikácie na pracovnú plochu zariadenia, ako aj zabezpečenie rovnakého používateľského zážitku naprieč všetkými platformami.

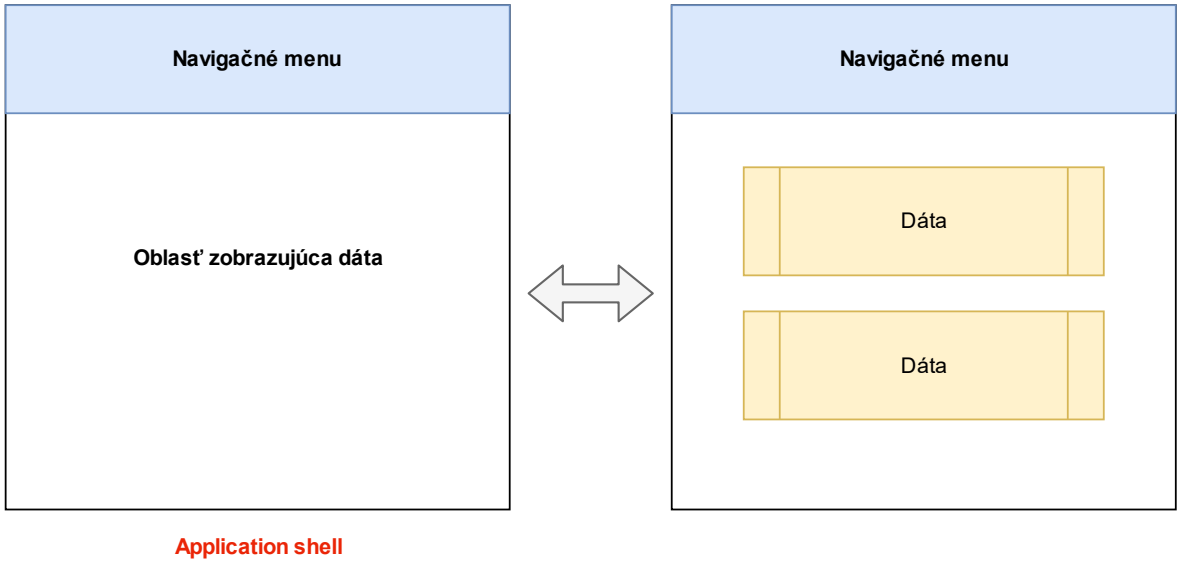
Kľúčové vlastnosti PWA [3]:

- **Progresívna** – Uspokojuje každého používateľa bez ohľadu na typ webového prehliadača, ktorý používateľ používa.
- **Responzívna** – Zabezpečí rovnako kvalitnú prevádzku bez rozdielu veľkosti zobrazovacej plochy zariadenia.
- **Nezávislá na pripojení –** Tzv. service worker umožňuje režim offline, ako aj prevádzku pri obmedzenom internetovom pripojení.
- **Podobná natívnym aplikáciám** – Pôsobí ako natívna aplikácie, kvôli tzv. app shell modelu, ktorý oddeľuje aplikačnú funkcionalitu od obsahu (*dáta*).
- **Aktuálny obsah** – Všetky aktualizácie zabezpečuje service worker nepretržite na pozadí.
- **Bezpečná** – Využíva na svoju prevádzku protokol HTTPS.
- **Rozoznateľná pre iné služby a aplikácie** – Identifikovaná ako aplikácia vďaka jej manifest súboru a registrácií service workera.
- **Push notifikácie** – Podobne ako natívne mobilné aplikácie.
- **Odkaz na ploche** – Namiesto inštalácie ako u natívnych aplikácií umožňuje pridať odkaz na plochu zariadenia.
- **Ľahko šíriteľná** – Jednoduché šírenie aplikácie vďaka jednoduchého zdieľania URL adresy bez požiadavku na inštaláciu.

V nasledujúcich podkapitolách budú rozobraté základné črty tejto pomerne novej platformy, ktorá bola prvýkrát definovaná v roku 2015. Jej vznik bol podmienený pokrokom v oblasti webových technológií (*HTML 5, CSS3 a JavaScript*) a stále výkonnejšími a schopnejšími webovými prehliadačmi [3].

2.1 Architektúra application shell

Pri tomto type architektúry ide o minimálne množstvo HTML, CSS a JavaScript súborov, ktoré sú potrebné na prevádzku používateľského rozhrania progresívnej webovej aplikácie. Táto architektúra prispieva k celkovej rýchlosti aplikácie. Prvé načítanie aplikácie je klasicky sprostredkované webovou službou na internete, pri čom service worker, pracujúci na pozadí zabezpečí okamžité uloženie kompletného obsahu aplikácie do tzv. cache, alebo inak úložiska zariadenia. To znamená, že pri prvom načítaní sa celá Application Shell spolu aj s dátami sprostredkúva klientovi cez sieť, avšak pri každom ďalšom načítaní aplikácie sa Application Shell, teda statické prvky aplikácie, sprostredkúva z cache. Internetové pripojenie sa potom využíva len na sprostredkovanie nových dát. Vďaka tomuto typu architektúry, ktorá umožňuje instantné načítanie aplikácie, sa progresívne webové aplikácie približujú svojou výkonnosťou natívnym aplikáciám.



Obrázok 1: Application shell

Pri návrhu Application Shell sú veľmi nápomocné nasledujúce otázky:

- Čo musí byť na obrazovke okamžite ?
- Ktoré ďalšie UI komponenty sú kľúčové v rámci aplikácie ?
- Ktoré podporné zdroje sú potrebné pre Application Shell ? Napríklad obrázky, JavaScript, CSS a pod.

2.2 Service worker

Service worker je skript, ktorý je spustený prehliadačom na pozadí, separátne od web stránky, čím otvára dvere novým prvkom, ktoré na svoj beh nevyžadujú web stránku,

respektíve interakciu používateľa. Momentálne sa jedná o prvky ako push notifikácie a background synchronizáciu. Kľúčová vlastnosť service workera je schopnosť zasahovať a riadiť sieťovú prevádzku a manažment tzv. cache (*úložisko*). To umožňuje zabezpečiť offline UX (*používateľský zážitok*).

Kľúčové črty service workera [4]:

- Tvorený JavaScript súborom, takže nemôže pristupovať priamo k DOM (*Document Object Model*). Namiesto toho komunikuje so stránkami tak, že odpovedá na správy poslané cez rozhranie postMessage, a následne podľa odpovede stránka upraví svoj DOM.
- Service worker je programovateľný sieťový proxy, ktorý umožňuje kontrolovať sieťovú prevádzku medzi serverom a stránkou, respektíve aplikáciou.
- Ak sa service worker nepoužíva je terminovaný, takže sa nedá spoliehať na globálny stav v rámci service worker onfetch a onmessage obstarávacích metód. Ak teda existujú dáta, ktoré sa musia zachovať a znova použiť po reštartovaní aplikácie, service worker má prístup k IndexedDB API. Jedná sa o JavaScript objektovo orientovaný transakčný databázový systém. Je podobný SQL relačným databázovým systémom.
- Service worker výrazne využíva tzv. promise. Vo svojej podstate sa jedná o prácu s asynchrónnymi udalosťami. To znamená, že script v rámci promise berie do argumentu funkciu, ktorá vyhodnotí, či bol samotný promise naplnený, alebo nie.

Čo sa týka podpory service workerov naprieč poprednými modernými prehliadačmi, vidíme čoraz viac rastúci trend. Service workery sú podporované štandardnými webovými prehliadačmi ako Chrome, Firefox, Opera a Microsoft Edge [5]. Nedávno ohlásil podporu tejto technológie aj prehliadač platformy iOS Safari [6].

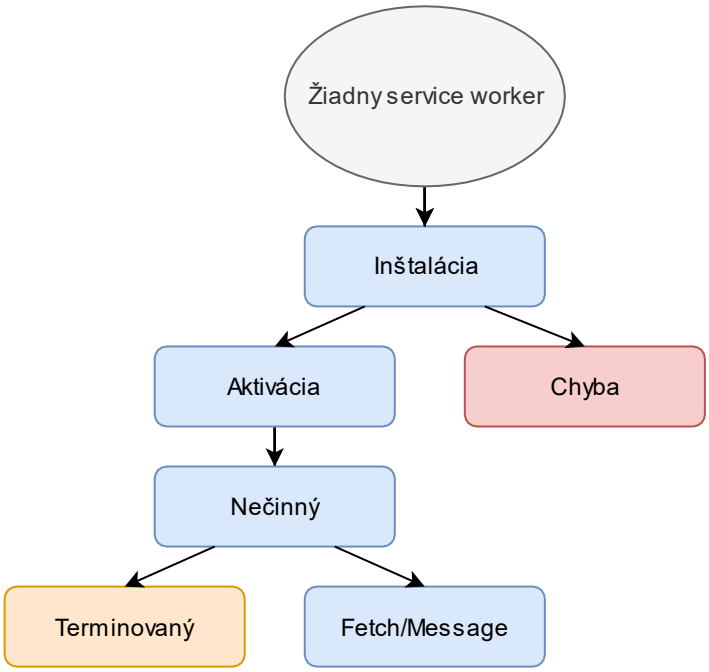
Je veľmi dôležité doplniť, že na úspešné nasadenie service workera do prevádzky je potrebná podpora protokolu HTTPS (*HyperText Transfer Protocol Secure*). Pre zabezpečenie podpory HTTPS je potrebné nastavenie TLS (*Transport Layer Security*) certifikátu pre daný server. Síce v rámci vývoja je možné prevádzkovať service workera v rámci localhost, v ostrej prevádzke, kvôli jeho vymoženostiam, je možná registrácia service workera len na stránkach, ktoré sú zasielané protokolom HTTPS. Tým sa zamedzí potenciálnej manipulácií tohoto skriptu tretími osobami.

2.2.1 Životný cyklus

Na úvod treba poznamenať, že životný cyklus service workera je kompletne oddelený a nezávislý od web stránky. Pre nainštalovanie service workera k príslušnej stránke, je najprv potrebná jeho registrácia, ktorá sa vykoná v rámci JavaScriptu stránky. Registráciou service workera sa spustí jeho inštalačný proces na pozadí.

V rámci inštalačného procesu sa štandardne vytvorí tzv. cache (*úložisko*), kde sa uložia všetky statické prvky aplikácie. Pokiaľ sú všetky súbory úspešne uložené, inštalačný proces service workera je ukončený. Ak zlyhá stiahnutie a uloženie niektorého zo statických prvkov aplikácie, potom je inštalácia service workera neúspešná a service worker sa neaktivuje. Následne sa celý proces opakuje.

Po inštalácii nasleduje aktivácia service workera. V rámci aktivácie sa obstarávajú staré úložiská cache. Po tomto aktivačnom kroku service worker preberá kontrolu nad všetkými stránkami, ktoré pod neho spadajú, avšak stránka, na ktorej prebehla úvodná registrácia service workera ním nebude kontrolovaná až do ďalšieho načítania. V momente keď service worker preberie kontrolu môže nadobudnúť jeden z dvoch stavov. Buď je service worker terminovaný, aby tak šetril pamäť, alebo obstaráva fetch a message eventy, ktoré nastanú pri vzniku sieťovej požiadavky, alebo správy z niektorej stránky príslušnej aplikácie. Celý vyššie opísaný všeobecný životný cyklus môžeme vidieť na obrázku dole.



Obrázok 2: Životný cyklus service workera

2.3 Úložisko cache

Na začiatku kapitoly bol načrtnutý základný koncept správania progresívnej webovej aplikácie, ktorá dokáže vďaka service worker ukladať svoje statické prvky a dáta, čo výrazne zvyšuje rýchlosť aplikácie. Pre ukladanie dát progresívna webová aplikácia využíva dva druhy technológii [7]:

- **Cache API** sa používa pre ukladanie prvkov a prostriedkov nutných pre prevádzku aplikácie bez internetového pripojenia.
- **IndexedDB** sa používa pre ukladanie všetkých ostatných dát, ktoré sú zväčša generované používateľom. Ide o JavaScript verziu databázového systému podobného SQL relačným databázovým systémom.

Obidva varianty sú asynchrónne (*IndexedDB je založený na udalostiach a Cache API je založený na tzv. promise*). Rovnako pracujú s technológiami web worker, window a service worker. Cache API aj IndexedDB sú v súčasnej dobe podporované všetkými poprednými webovými prehliadačmi (*Firefox, Opera, Chrome a Microsoft Edge*) [7].

Čo sa týka množstva dát, ktoré je možné uložiť pomocou týchto technológií, obidve umožňujú ukladanie dát až do dosiahnutia limitu, tzv. kvóta webového prehliadača. Aplikácie dokážu kontrolovať množstvo využitého priestoru kvóty pomocou tzv. Quota Management API. Jednotlivé kvóty sa líšia v závislosti od webového prehliadača. Nižšie je uvedená tabuľka č.1 s limitami pre jednotlivé popredné prehliadače.

Tabuľka 1: Limity ukladacích kvót prehliadačov [7]

Prehliadač	Limit
Chrome	< 6 % voľného miesta
Firefox	< 10 % voľného miesta
Safari	< 50 MB
Internet Explorer 10	< 250 MB
Microsoft Edge	< 20 % voľného miesta

Pôvod využívajúci úložisko (*napr. progresívna webová aplikácia*), dostane alokovaný priestor v pamäti, ktorý je zodpovedný si sám spravovať. Tento voľný priestor v pamäti je zdielaný všetkými možnými formami ukladania (*IndexedDB aj Cache API*). Po prekročení povoleného limitu môže quota manager u niektorých prehliadačov „čistiť“ pamäť pomocou

tzv. LRU (*Least Recently Used*) politiky. Teda dáta z pôvodu, ktorý je časovo najstarší, sú zmazané prvé. Nižšie je uvedená tabuľka politiky uvoľňovania pamäte jednotlivých prehliadačov.

Tabuľka 2: Pravidlá uvoľňovania pamäte [7]

Prehliadač	Pravidlá uvoľňovania pamäte
Chrome	LRU po prekročení pamäte prehliadača
Firefox	LRU po naplnení lokálneho úložiska
Safari	Pamäť neuvoľňuje
Edge	Pamäť neuvoľňuje

2.3.1 Cache API

Cache API je rozhranie, ktoré poskytuje ukladací mechanizmus pre páry objektov typu požiadavka a odpoveď. Napríklad ukladanie rôznych prvkov počas rôznych fáz životného cyklu service workera. Síce bolo pôvodne toto rozhranie vyvinuté a definované pre potreby service workerov, je prístupné aj pre window a web workerov. Vstupný bod rozhrania Cache API sa nazýva caches [8]. Service worker môže využívať viacero objektov typu cache.

Spôsob akým sú obstarávané aktualizácie cache je výlučne definovaný implementáciou príslušného scriptu service workera, ktorý cache využíva. Všetky aktualizácie položiek v rámci cache musia byť explicitne vyžiadané. Pretože Cache API nepozná žiadnu dobu expirácie, musia byť tak isto periodicky mazané. Pokiaľ množstvo dát v cache prekročí explicitný pamäťový limit webového prehliadača (*limit sa líši od prehliadača k prehliadaču*), prehliadač začne vylučovať dáta z cache podľa ich príslušného pôvodu, a to postupne za sebou, až kým sa množstvo dát v cache nedostane pod limit stanovený prehliadačom. To znamená, že prehliadač zmaže všetky dáta jedného pôvodu, skontroluje množstvo zabratej pamäte v závislosti so svojím explicitným limitom a presunie sa na dáta ďalšieho pôvodu, ktoré buď zmaže, alebo ponechá. Zaužívané spôsoby využívania Cache

2.3.2 IndexedDB

Jedná sa o nízkoúrovňové API rozhranie pre ukladanie väčšieho množstva štruktúrovaných dát na strane klienta, vrátane podpory ukladania dát typu blob. Toto rozhranie využíva indexy, čo umožňuje vysoko výkonné vyhľadávanie dát. Zatiaľ čo Cache API rozhranie je

vhodné pre ukladanie menšieho objemu dát (*ako napríklad statické prvky a scripty aplikácií*), rozhranie IndexedDB API je vhodné pre ukladanie väčšieho objemu dát. [9]

Operácie vykonané pri používaní IndexedDB sú vykonávané asynchrónne, aby neblokovali prevádzku aplikácie. Existuje viacero webových technológií, ktoré ukladajú dáta na strane klienta (*lokálny disk*). IndexedDB je však najpoužívanejšou technológiou v tejto oblasti. Proces, ktorý využíva webový prehliadač na určenie množstva alokovanej pamäte pre webové dáta, sa líši od prehliadača k prehliadaču. Rovnako tak závisí na druhu webového prehliadača, ako sa prehliadač vysporiada s prekročením limitu dát, ktorý je pre daný prehliadač stanovený. Toto definujú ukladacie limity a kritériá vylúčenia určitého webového prehliadača. [9]

2.4 Manifest súbor

Manifest súbor je súbor typu JSON (*JavaScript Object Notation*), ktorý popisuje webovému prehliadaču správanie webovej aplikácie po „inštalácii“ na používateľovom mobilnom, prípadne desktop zariadení. Napríklad prehliadač Chrome explicitne vyžaduje manifest súbor v prípade, že chce vývojár umožniť používateľovi mobilného zariadenia pripnúť progresívnu webovú aplikáciu na pracovnú plochu zariadenia. Medzi typickými informáciami, ktoré manifest súbor obsahuje sú napríklad názov aplikácie, ikony v rôznych veľkostiach, štartovacia URL adresa aplikácie a iné. Po vytvorení manifest súboru sa štandardne uvádza jeho odkaz pomocou link tagu na všetkých stránkach, ktoré PWA využíva. [10]

2.5 Prípady použitia skriptu Service worker a úložiska Cache

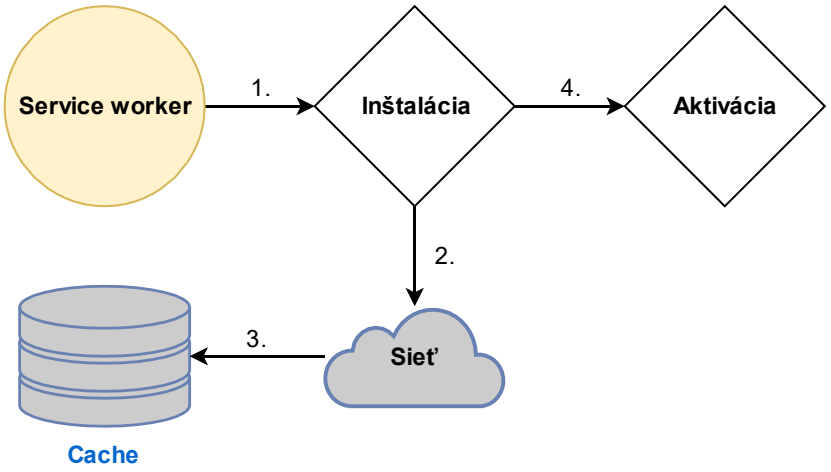
V predchádzajúcich častiach práce boli priblížené všetky platformy, ktoré sa dnes využívajú pri vývoji aplikácii využívajúcich webové služby. Nachádzajú sa medzi nimi staršie no stále využívané platformy (*webová aplikácia, mobilná aplikácia*) a novšie alternatívy, ktoré kombinujú prvky starších platforiem (*hybridné a progresívne aplikácie*) a prinášajú so sebou nové koncepty vývoja aplikácií. Práca sa ďalej zameriava hlbšie na problematiku progresívnych webových aplikácií a vysvetľuje základné črty a vlastnosti tejto platformy.

V tejto časti práce bude priblížená a vysvetlená robustnosť a sila tejto platformy, ktorá leží práve v kombinácií scriptu Service worker a rozhrania Cache, ktoré umožňuje ukladanie dát na strane klienta. Súčinnosť spomínaných technológií a klasického protokolu HTTPS ponúka šikovné riešenie, ktoré poskytuje používateľovi kvalitný a responzívny UX nezávisle od kvality internetového pripojenia. Nasledujúce podkapitoly priblížia niektoré

prípady použitia mechanizmov Cache v súčinnosti so scriptom service worker, a to z hľadiska ukladania prostriedkov a dát, ako aj z hľadiska obstarávania sieťovej prevádzky (*požiadavky typu request*).

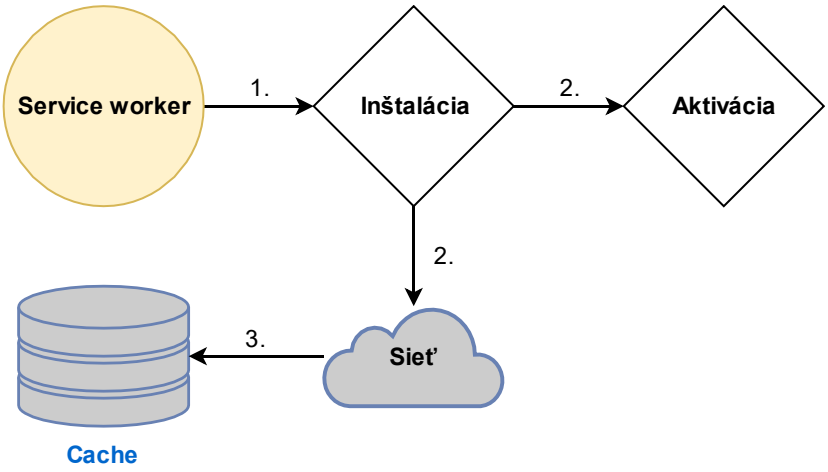
2.5.1 Ukladanie zdrojov aplikácie

Pri ukladaní zdrojov aplikácie využívame rôzne fázy životného cyklu skriptu service worker. Na Obrázku 3 uvedenom nižšie, je znázornená schéma prípadu ukladania základných prvkov aplikácie pri fáze inštalácie skriptu service worker. Jedná sa o ukladanie základných statických prvkov ako obrázky, CSS, JavaScript, HTML a pod. Teda ide o zdroje, bez ktorých by stránka respektíve aplikácia nemohla fungovať (*ekvivalent iniciálneho stiahnutia natívnej aplikácie do zariadenia*).



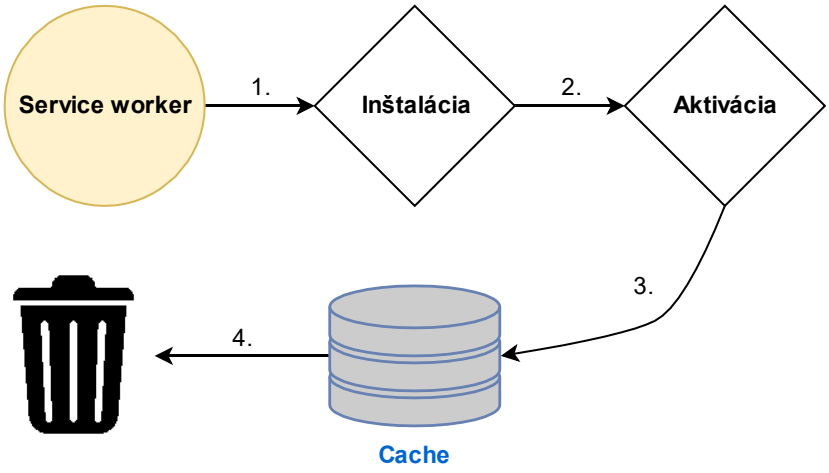
Obrázok 3: Inštalácia je závislá od ukladania

Túto inštalačnú fázu životného cyklu skriptu service worker je možné využiť aj v druhom prípade. Jedná sa o prípad, kedy ukladáme prvky zo siete, ktoré nepovažujeme za statické, a teda nechceme zdržovať inštaláciu skriptu service worker, a teda inštalácia nie je závislá od úspešného uloženia do cache. Prípad je schématicky znázornený na nasledujúcej strane v Obrázku 4.



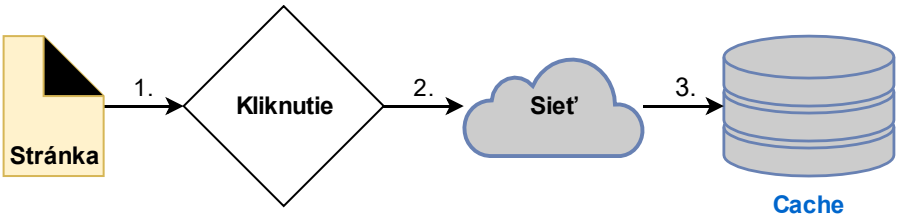
Obrázok 4: Inštalácia nezávislá od ukladania

Bezprostredne po nainštalovaní skriptu service worker, a teda zániku jeho prípadnej predchádzajúcej verzie, prichádza ďalšia fáza jeho životného cyklu, a síce jeho aktivácia. Nakoľko je stará verzia skriptu neaktívna, je dobré v tomto momente riešiť migrácie schém v rámci IndexedDB rozhrania, a rovnako aj vyprázdniť nepoužívané objekty typu cache. Treba brať do úvahy, že akékoľvek ostatné udalosti, ako napríklad fetch požiadavka, čakajú na ukončenie aktivačnej fázy. Schéma využitia aktivačnej fázy je znázornená na Obrázku 5 nižšie.



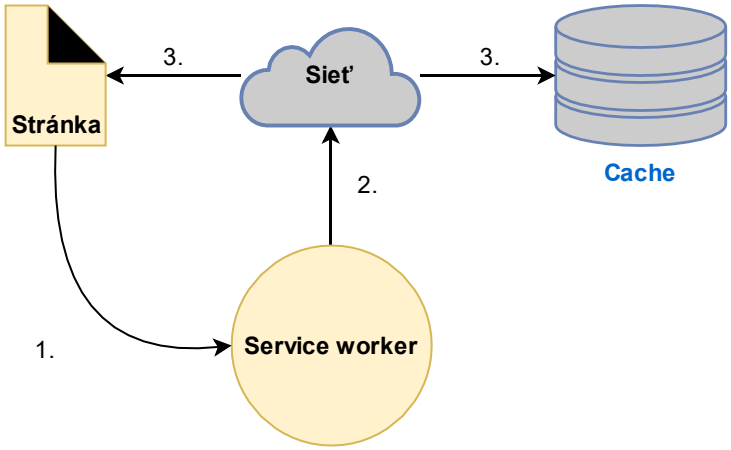
Obrázok 5: Manažment obsahu Cache po aktivácii

V momente keď je skript service worker aktívny, je možné pomocou vhodného prvku (*tlačítko, prípadne check box*) používateľského rozhrania umožniť samotnému používateľovi zvoliť konkrétne dáta (*video, článok a pod.*), ktoré chce uložiť do cache. Tento prípad použitia cache je schématicky znázornený na nasledujúcej strane na Obrázku 6.



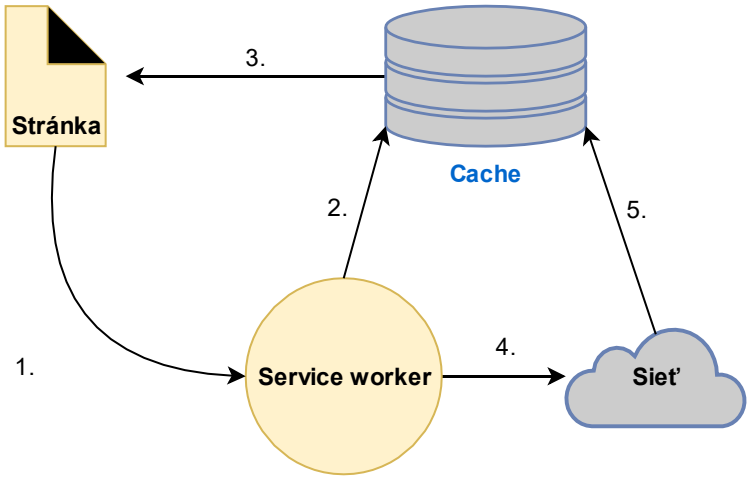
Obrázok 6: Ukladanie po interakcii používateľa

V prípade frekventovane aktualizovaných dát ako napríklad používateľova schránka so správami, prípadne stránka s článkami je možné využiť ukladanie do cache (*respektíve IndexedDB*) podmienené sieťovou odozvou. V tomto prípade treba mať na mysli manažment kapacity úložiska, aby sa predišlo zahlteniu. Keď teda sa odpoveď na sieťový požiadavok nenachádza v cache, service worker potrebné dáta získa zo siete, odošle ich na stránku a zároveň ich uloží do cache. Tento prípad je vyobrazený na Obrázku 7 nižšie.



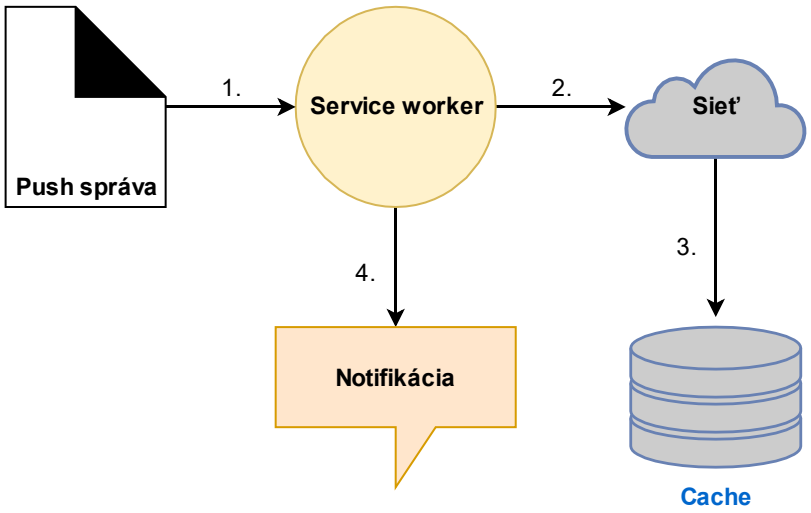
Obrázok 7: Ukladanie podmienené sieťovou odozvou

V prípade často sa aktualizujúcich dát, kde však nie je dôležité mať aktuálnu verziu dát v reálnom čase (*napríklad používateľov avatar*) môžeme využiť skript service worker tak, aby prioritne využíval uloženú verziu dát a aktualizáciu odložil na neskôr (*napríklad pri lepšom pripojení*). Táto varianta zamaskuje problémy pri pomalšej odozve servera a je schematicky znázornená na Obrázku 8 na nasledujúcej strane.



Obrázok 8: Maskovanie problému s odozvou servera

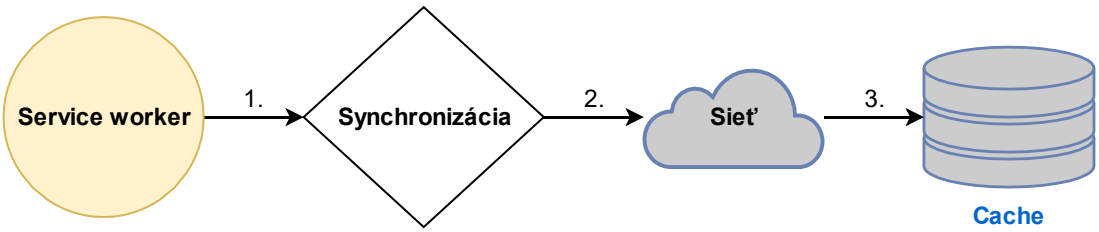
Ďalším prvkom, ktorý využíva technológiu service worker je tzv. rozhranie Push API. Toto rozhranie umožňuje prebudiť skript service worker v reakcii na správu zo služby servera. Prebudiť skript service worker je možné aj v prípade, že je aplikácia (*stránka*) zavretá. V tomto prípade je však veľmi dôležité obsah notifikácie okamžite uložiť do Cache (*IndexedDB*), nakoľko nevieme, či používateľ bude online aj v dobe zistenia, že bol notifikovaný. Prípad použitia cache v súvislosti s push notifikáciou je znázornený na Obrázku 9.



Obrázok 9: Push notifikácia

Nadväzujúc na predchádzajúci prípad, ak dochádza k aktualizáciam príliš často, že prípadné využitie rozhrania Push API by bolo príliš frekventované (*napríklad nové články, prípadne časová os sociálneho média*), je možné využiť tzv. synchronizáciu na pozadí. Jedná sa rovnako o koncept, ktorý je postavený na technológii service worker. Synchronizácia na

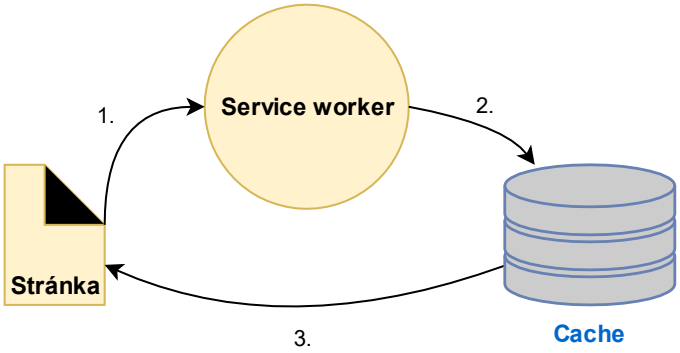
pozadí umožňuje skriptu service worker získať najnovšie dáta aj v prípade, že je samotná aplikácia (*stránka*) vypnutá. Prípad použitia synchronizácie na pozadí je schématicky znázornený na Obrázku 10 uvedenom nižšie.



Obrázok 10: Synhchronizácia na pozadí

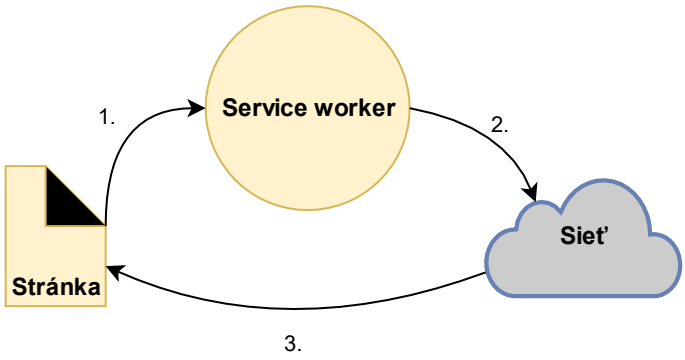
2.5.2 Odbavovanie sieťových dotazov aplikácie

Tak ako je kompletný manažment ukladania do rozhraní Cache, prípadne IndexedDB kompletne v réžii implementácie skriptu service worker, podobne je aj v tejto réžii využívanie dát z týchto rozhraní v reakciách na sieťové dotazy používateľa. V tejto podkapitole budú popísané niektoré prípady poskytovania dát z rozhrania Cache v závislosti na dostupnosť internetového pripojenia.



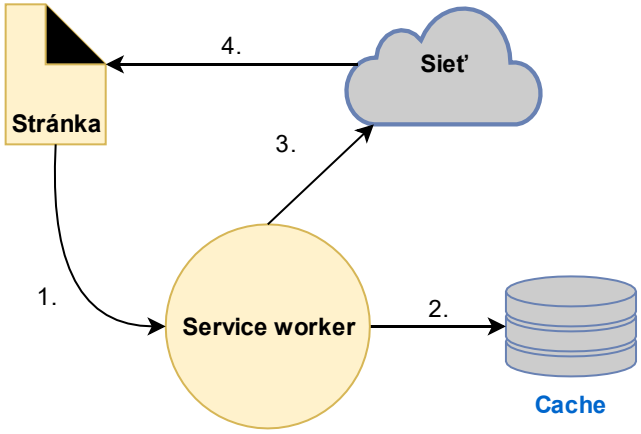
Obrázok 11: Načítanie statických prvkov z Cache

V prípade statických dát (*Application shell*), sú načítavané z úložiska cache všetky statické prvky, ktoré boli uložené počas inštalačnej fázy životného cyklu skriptu service worker. Schéma tohto prípadu poskytovania dát z rozhrania Cache je znázornená na Obrázku 11 vyššie. Pre dáta, ktoré nemajú využitie pre offline UX sa naopak využíva na ich sprostredkovanie sieťové pripojenie. Schéma tohto prípadu je zobrazená na Obrázku 12 na nasledujúcej strane.



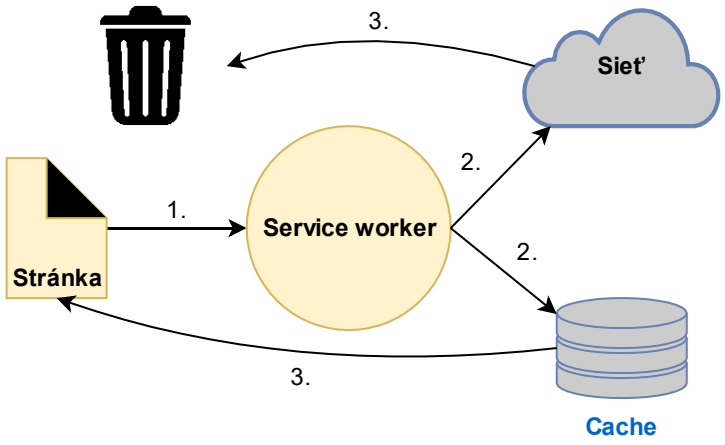
Obrázok 12: Sprostredkovanie dynamických dát

Obidva prípady zmienené vyššie však rieši prístup prioritného využitia rozhrania cache a následného dotazovania sa na sieť v prípade, že rozhranie cache požiadavku nevyhovelo. Tento prístup odbavovania dotazov je ideálny pri vývoji tzv. „offline-first“ aplikácií, ktoré zabezpečujú prevádzku aj v obmedzenom, prípadne žiadnom pripojení na internet. Schéma tohto prístupu je uvedená na Obrázku 13, ktorý je uvedený nižšie.



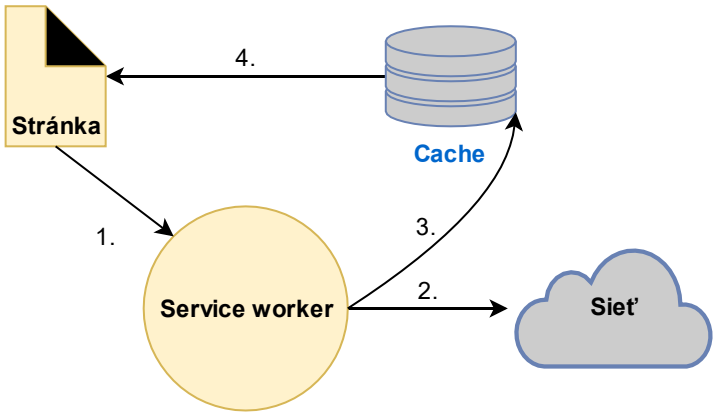
Obrázok 13: Prioritné využitie rozhrania Cache

Pri zariadeniach s obmedzenou veľkosťou úložiska, ako aj pri malých dátových položkách, kde je zámerom hlavne rýchlosť celkovej prevádzky aplikácie je vhodné využiť model použitia, kde sa odozva na dotaz odbavuje štýlom rýchlejší vyhráva. Schéma tohto modelu použitia úložiska a siete je znázornená na Obrázku 14 na nasledujúcej strane.



Obrázok 14: Simultánne využitie siete a Cache pri odozve

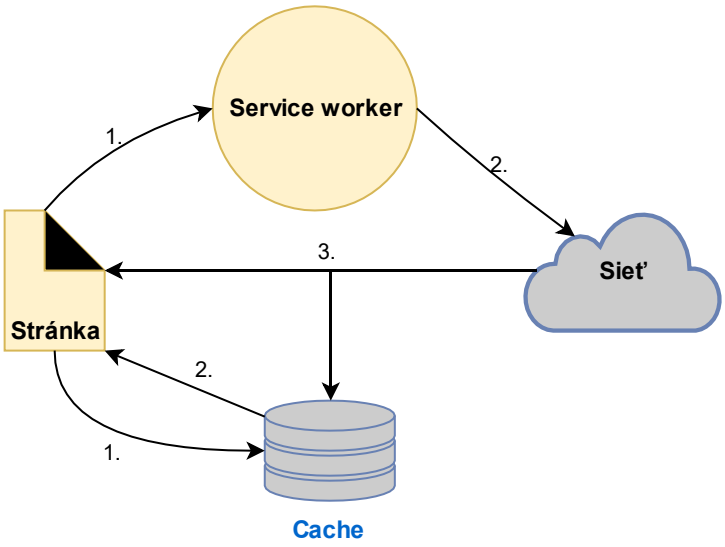
Môže nastať situácia, kedy stránka aplikácie obsahuje prvky, ktoré sú frekventovane aktualizované mimo danú verziu stránky. Napríklad sa jedná o nové články, prípadne časovú os sociálnych médií. To znamená, že aplikácia môže online používateľom poskytovať aktuálne dáta, no offline používateľom je poskytnutá staršia verzia uložená v cache. Samozrejme v prípade odbavenia sieťového požiadavku, sú úložiská aktualizované s najnovšími dátami. Tento model (*sieť prioritná/cache sekundárna*) má však svoje nevýhody v prípade, že používateľ má prerušované, alebo pomalé pripojenie. Vtedy musí používateľ čakať na to, kým mu sieť úplne zlyhá a až následne potom mu budú poskytnuté dáta rozhraním Cache. Tento model je znázornený na Obrázku 15 nižšie.



Obrázok 15: sieť prioritná/Cache sekundárna

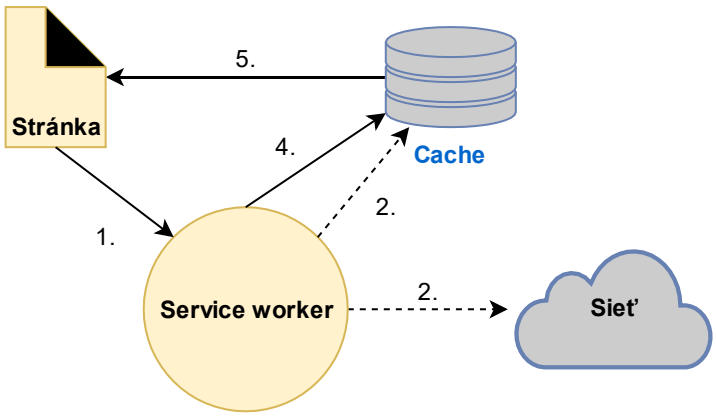
Lepším riešením situácie opísanej modelom sieť prioritná/cache sekundárna je model cache prioritná/sieť sekundárna. Tento model je schématicky znázornený na Obrázku 16 uvedenom na nasledujúcej strane. Tento typ riešenia vyžaduje stránku vykonať dve požiadavky, a síce jednu na sieť a druhú na rozhranie Cache. Myšlienka za tým je tá, že najprv sa používateľovi poskytnú uložené dáta rozhraním Cache a následne sa dáta

aktualizujú, hneď ako dorazia najnovšie dáta zo siete. Túto techniku využíva napríklad sociálna sieť Twitter. Pre lepšie UX je dôležité okamžite nenahrádzať staré dáta novými, aby sa používateľovi nestalo, že mu zrazu zmiznú dáta, ktoré mohol momentálne používať, hoci už neboli aktuálne. Naopak lepším spôsobom je ho informovať o dostupnosti nových dát.



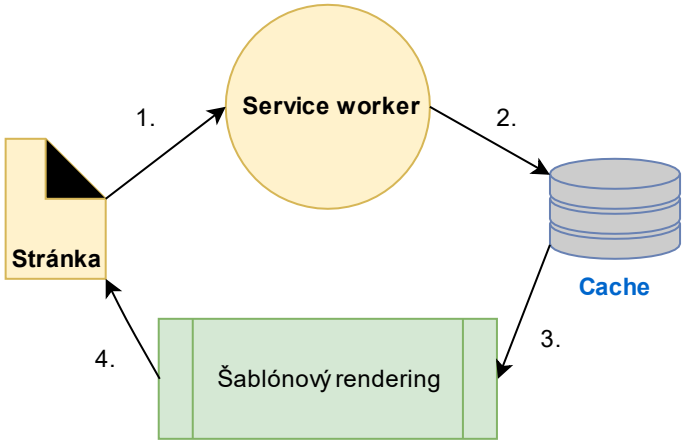
Obrázok 16: Cache priotiná/sieť sekundárna

Samozrejme môže nastať aj prípad, že zlyhajú obidva hlavné kanály pre sprostredkovanie dát (*rozhranie Cache aj sieť*). V tomto prípade je vhodné zabezpečiť nejakú generickú odozvu, informujúcu používateľa o zlyhaní obidvoch kanálov. Túto generickú odozvu bude v tomto prípade tvoriť prvok (*stránka*) získaný pri inštalovaní skriptu service worker. Ďalším využitím tohto modelu je napríklad uloženie rozpísaného emailu service workerom pre neskoršie odoslanie, hneď ako bude dostupné sieťové pripojenie. Schéma prípadu použitia tohto modelu je znázornená na Obrázku 17 uvedenom nižšie.



Obrázok 17: Generická odozva

Posledný prípad použitia skriptu service worker v súčinnosti s rozhraním Cache, ktorý bude v tejto práci uvedený je tzv. šablónovanie za pomoci skriptu service worker. Tento prípad použitia je znázornený nižšie na Obrázku 18. Tento typ modelu v princípe využíva rýchlo poskytnuté dáta (*dáta vo formáte JSON a zároveň HTML šablónu*) serverom, ktoré service worker uloží pomocou rozhrania Cache a následne na strane klienta vyskladá a poskytne používateľovi. Tento štýl odbremení server od vykonávania renderovania dát.



Obrázok 18: Šablónovanie za pomoci skriptu Service worker

V tejto kapitole boli zhrnuté niektoré so spôsobov ukladania dát za pomoci rozhrania Cache API v súčinnosti so skriptom Service worker. Rovnako boli aj uvedené prípady použitia tejto kombinácií technológií pri následnom odbavovaní sieťových dotazov aplikácie, teda využívanie uložených dát z rozhrania Cache API v synergii so sieťovou odozvou. Je potrebné podotknúť, že záleží na konkrétnej aplikácii, ktoré z vyššie spomenutých spôsobov by boli pre výslednú aplikáciu vhodné. Vo všeobecnosti je potrebné sa zamerať na druhy URL dotazov, ktoré aplikácia využíva, a následne každý z nich ošetriť najvhodnejším prípadom použitia kombinácie skriptu Service worker a rozhrania Cache API.

3 Technológie využívané pri vývoji klientských aplikácií

Všetko o aktuálnych platformách, na ktorých sú postavené dnešné moderné aplikácie využívajúce webové služby, bolo priblížené v predchádzajúcich kapitolách. Táto kapitola sa bude zaoberať samotnými najpopulárnejšími technológiami, ktoré sa aktuálne využívajú pri vývoji takýchto aplikácií. Je už známe, že okrem natívnych aplikácií (*aplikácie týkajúce sa iba mobilných zariadení*) máme webové aplikácie, teda aplikácie, ktoré k svojej prevádzke využívajú webový prehliadač. Tieto aplikácie sú zostavené z rady HTML (*HyperText Markup Language*) dokumentov (*stránky*), ďalej ich príslušných CSS (*Cascading Style Sheets*) pravidiel a JavaScript súborov. Všetky tieto tri základné stavebné kamene aplikácie sa v dnešnej dobe vyvíjajú za použitia rôznych nástrojov a knižníc, ktoré zrýchľujú a uľahčujú samotný vývoj aplikácie. Cieľom tejto kapitoly bude predstaviť a porovnať aktuálne najpopulárnejšie front-end JavaScript knižnice, ktoré sa vo svete webových aplikácií využívajú pri vývoji a budú kľúčové pri vývoji klientskej aplikácie pre jednoduché vyhľadávanie letových spojení.

3.1 Knižnica Angular

Angular je značne komplexná a rozsiahla JavaScript knižnica vytvorená pre vývoj klientských aplikácií. Táto knižnica bola v roku 2010 vytvorená spoločnosťou google, ktorá ju aj udržiava. Je postavená na jazyku TypeScript. Jadro knižnice a jej funkcionality sú implementované ako rada TypeScript knižníc, ktoré vývojár importuje do svojej aplikácie [11].

Základným stavebným blokom Angular aplikácie sú tzv. NgModuly, ktoré poskytujú kompilačný kontext pre komponenty. Každy NgModul predstavuje sadu funkcií, ktoré tvoria konkrétny Modul, a teda výsledná Angular aplikácia je tvorená súborom týchto NgModulov a komponentov. Každá Angular aplikácia má teda aspoň jeden NgModul, tzv. root modul a jeden komponent, tzv. root komponent [12]. Komponenty ďalej z pohľadu architektúry celej aplikácie definujú tzv. view (*náhľad*). Náhľad je tvorený sadou zobrazovacích elementov, ktoré knižnica Angular manažuje podľa aplikačnej logiky a dát. Komponenty ďalej využívajú tzv. services (*služby*), ktoré poskytujú špecifické funkcionality, ktoré nemusia priamo súvisieť s prezentačnou vrstvou (*views*). Poskytovatelia rôznych iných služieb môžu byť vložený do komponentov ako tzv. dependencies (*závislosti*), čo robí z knižnice Angular efektívnu modulárnu knižnicu, nakoľko koncept tejto architektúry podporuje pozdejšiu rozšíriteľnosť aplikácie. Komponenty aj služby sú vo svojej podstate

triedy, ktorých typ a základné funkcionality popisujú ich metadáta, ktorými sa Angular riadi pri ich využívaní. Napríklad teda metadáta pre triedu typu komponent slúžia na asociáciu komponentu so šablónou, ktorá definuje náhľad. Šablóna kombinuje štandardné HTML doplnené o Angular príkazy, ktoré umožňujú knižnici modifikovať HTML šablónu pred jej samotným zobrazením. Metadáta pre triedu typu služba poskytujú informácie, ktoré knižnica Angular potrebuje za účelom sprístupnenia týchto služieb pre jednotlivé komponenty aplikácie. Jednotlivé komponenty Angular aplikácie typicky definujú množstvo náhľadov, ktoré sú hierarchicky zoradené. Knižnica Angular preto poskytuje tzv. router service (*smerovacia služba*), ktorá poskytuje v rámci prehliadača sofistikované smerovacie kapacity medzi jednotlivými náhľadmi.

Výhody:

- Veľká komunita vývojárov, knižnica je udržiavaná spoločnosťou Google.
- Podpora TypeScript.
- Využíva MVVM *(Model-View-ViewModel*) architektúru.

Nevýhody:

- Široká komplexnosť knižnice robí Angular časovo náročnou knižnicou vzhľadom na čas potrebný na zvládnutie syntaxu.
- Problémy pri migrácii medzi verziami Angularu.

3.2 Knižnica Vue

Knižnica Vue je pomerne nový projekt, ktorý bol prvýkrát vydaný v roku 2014. Vue bola vytvorená bývalým angular vývojárom menom Evan You, ktorý sa inšpiroval práve knižnicou Angular, avšak jeho cieľom bolo vytvoriť jednoduchšiu a menej komplexnú knižnicu. Do dnešného dňa je knižnica Vue udržiavaná a vyvíjaná Evanom a jeho tímom. Vue je knižnica určená pre vývoj interaktívnych webovýc rozhraní. Hlavným cieľom Vue je poskytovať benefity reaktívneho spájania dát a kompozitných náhľadových komponentov [12]. Reaktívne spájanie dát zabezpečuje okamžitú synchronizáciu dát s DOM (*Document Object Model*). Kompozitné náhľadové komponenty definujú koncept, kde sa konkrétne UI (*používateľské rozhranie*) skladá z rady komponentov. Tieto komponenty fungujú ako nezávislé prvky a celé používateľské rozhranie je rozdelené do elementárnych dielčích komponentov. Tento prístup umožňuje opätovné použitie komponentov vo viacerých používateľských rozhraniach. Samotné Vue sa teda sústredí najmä na problematiku

prezentačnej vrstvy. Z tohto hľadiska nie je až tak značne časovo náročné na osvojenie, ako napríklad Angular. Na druhej strane je veľmi pravdepodobné, že na vývoj konkrétnej aplikácie bude potrebné spojiť túto knižnicu s ďalšími inými technológiami (*knižnicami*).

Výhody:

- Jednoduchá architektúra.
- Zorientovanie sa v knižnici v relatívne krátkom čase.
- Stavia na základných webových technológiach (*HTML, CSS, JavaScript*).
- Rýchlosť.

Nevýhody:

- Malá komunita vývojárov.
- Slabá dokumentácia v porovnaní s Angularom a Reactom.
- Udržiavaná relatívne malým tímom v porovnaní s Angularom a Reactom.

3.3 Knižnica React

Podobne ako Vue, React je JavaScript knižnica zameraná na problematiku náhľadov. Knižnica React bola vydaná na open source platforme spoločnosťou Facebook v roku 2013 [13]. React je deklaratívny, to znamená, že sa zameriava na definovanie konkrétnych prvkov používateľského rozhrania (*ich náhľady*), a samotné aktualizovanie a zobrazovanie týchto prvkov už zabezpečuje knižnica React. Z tohto princípu vyplýva, že knižnica React je tiež založená na koncepte komponentov. Pri vývoji sa teda používateľské rozhranie rozdelí do potrebných komponentov, ktoré sú zapúzdrené a spravujú svoj vlastný stav, ktorý priamo súvisí s udalosťami, ktoré sa udiali na konkrétnom používateľskom rozhraní. Tok dát medzi komponentami zabezpečuje tzv. props [13]. Jednotlivé komponenty sú definované ako buď tzv. functional components (*funkčné komponenty, ktoré neudržujú svoj stav*), alebo tzv. class based components (*komponenty implementované ako triedy, ktoré udržujú svoj stav*). Vo svojej podstate oba varianty komponentov sú tvorené JavaScript funkciami, ktoré prijímajú dáta ako argument (*props*) a pomocou render funkcie zobrazujú komponent (*časť používateľského rozhrania*). Pri samotnom zobrazovaní React využíva tzv. virtuálny DOM [13]. V rámci tohto virtuálneho DOM sú jednotlivé React elementy objekty tvoriace strom. Následne React podľa svojho virtuálneho DOM spravuje a aktualizuje faktický DOM webovej stránky. Táto knižnica využíva špeciálny syntax podobný syntaxu HTML prípadne XML. Nazýva sa JSX (*JavaScript eXtension*). JSX je spôsob, akým React rozširuje klasický

syntax JavaScriptu. Toto rozšírenie umožňuje spájať funkcionalitu so samotným statickým náhľadom (*view*), takže komponent je kompletne napísaný v JavaScripte, ktorý zahŕňa funkcionalitu aj vzhľad konkrétneho komponentu. Samozrejme, v knižnici React nie je povinnosťou toto JSX (*JavaScript rozšírenie*) používať, a teda vývojár sa môže rozhodnúť pre písanie v čistom ES5 JavaScript. Je však dôležité spomenúť, že syntax JSX nie je príliš časovo náročný na osvojenie a je viac orientovaný na problematiku používateľského rozhrania, ako klasický JavaScript. Spoločnosť Facebook v knižnici React prichádza s novým fenoménom nazvaným „CSS in JS“, čo by sme mohli voľne preložiť ako integrovanie CSS v rámci príslušného JavaScript súboru. Tento fenomén umožňuje ešte prehľadnejší vývoj CSS aspektu komponentov, a rovnako podporuje celkovú modularitu v rámci prezentačného aspektu komponentov.

Výhody:

- Jednoduchší syntax v porovnaní s Angularom.
- Podobne ako Vue je vysoko flexibilný (*rieši iba problematiku prezentačnej vrstvy*).
- Využíva virtuálny DOM.
- Dáta prúdia vo vzťahu rodič-dieťa, teda dáta dieťaťa neovplyvňujú dáta rodiča.
- Obrovská komunita vývojárov.
- Podpora PWA.

Nevýhody:

- Trend vo vyvíjaní React aplikácií sa spolieha hlavne na syntax JSX, a to aj v rámci problematiky CSS a HTML.
- Priama manipulácia s DOM sa neodporúča, nakoľko React manipuluje s DOM cez svoj virtuálny DOM.

3.4 Vyhodnotenie

V kapitole štyri sme priblížili aktuálne najpopulárnejšie JavaScript knižnice, ktoré sa používajú na vývoj používateľských rozhraní. Pri každej jednej knižnici boli uvedené výhody a nevýhody konkrétnej knižnice. Z troch spomínaných bol Angular jednoznačne najstaršou, najkomplexnejšou a najrozsiahlejšou knižnicou, a naopak Vue je najnovšou (*avšak rýchlo rastúcou*), vysoko flexibilnou JavaScript knižnicou zaoberajúcou sa problematikou používateľských rozhraní. Pri porovnávaní sa bral ohľad aj na vlastnosti spomínaných knižníc, ktoré sú kľúčové pre začínajúceho vývojára. Dôležitými vlastnosťami

z tohto pohľadu sú napríklad komplexnosť knižnice, náročnosť syntaxu, veľkosť komunity vývojárov a ďalšie. Porovnanie týchto troch knižníc je zosumarizované v Tabuľke 3 uvedenej nižšie. Z troch spomínaných knižníc sa práve knižnica React javí byť, tak povediac, zlatou strednou cestou. Najmä kvôli značnej popularite a faktu, že túto knižnicu využívajú pri vývoji používateľských rozhraní aj vo firme Pelikan, rozhodlo sa, že používateľské rozhranie aplikácie pre jednoduché vyhľadávanie letových spojení bude postavené na knižnici React.

Tabuľka 3: Porovnanie knižníc

Charakteristika/Knižnica	Angular	React	Vue
Vlastník	Google	Facebook	Evan You
Komunita/popularita	2.	1.	3.
Plná podpora MVC ?	Áno	Nie	Nie
Podpora konceptu komponentov ?	Áno	Áno	Áno
Jednoduchosť manažmentu stavu	3.	1.	2.
Jednoduchosť syntaxu	3.	2.	1.
Časová náročnosť osvojenia	3.	2.	1.
Škálovateľnosť ?	Áno	Áno	Áno
Časová náročnosť vývoja	3.	2.	1.
Dokumentácia	3.	1.	2.
Šablónovacie jazyky	Typescript, HTML, CSS/Sass/Less	JSX, CSS/Sass/Less	HTML, JavaScript/Typescript/JSX, CSS/Sass/Less
CLI (*Comand Line Interface*) ?	Áno	Áno	Áno

3.5 Knižnice doplňujúce knižnicu React

V súčasnosti sa pri vývoji moderných používateľských rozhraní webových aplikácií využíva široké spektrum rôznych nástrojov, ktoré rozširujú klasické základné technológie (*HTML, CSS, JavaScript*). Je to hlavne kvôli urýchleniu vývoja aplikácie a odbremeneniu vývojových tímov od vývoja redundantných funkcionalít, čo umožňuje sústredenie sa na implementáciu konkrétneho prípadu použitia (*use case*). V prípade vyvíjaného nástroja sa, okrem teda knižnice React, jedná najmä o knižnice postavené na CSS a ďalších technológiách používaných pri vývoji webových aplikácií, ktoré vhodne doplňujú knižnicu React.

Problematika vývoja CSS pre webové aplikácie a web stránky prechádzala, a stále prechádza, prirodzeným vývinom, reflektujúcim aktuálne vymoženosti webových prehliadačov a technológie využívané pri vývoji používateľských rozhraní(*v súčasnosti sa jedná najmä o rôzne JavaScript knižnice*). Pôvodne sa dáta zobrazovali za použitia obyčajných, serverom generovaných HTML dokumentov a ich základnými elementami (*tabuľka a podobne.*). Viac ako 20 rokov dozadu prichádza CSS, ktoré prináša väčšiu komplexnosť v rámci zobrazovania HTML dokumentov tým, že umožňuje priradzovať štýlovacie atribúty jednotlivým elementom stránky. Je to v snahe oddeliť prezentačný aspekt webovej stránky od samotných dát. V praxi to znamenalo doslova vytvorenie separátneho súboru, tvoriaceho zoznam so štýlmi pre konkrétne aspekty HTML dokumentu. Od tohto momentu vnímame signifikantný vývoj v rámci webového dizajnu a technológie CSS *(vznik CSS preprocesorov SASS a LESS a podobne*).V rámci open source publikovania knižnice React spoločnosťou Facebook bolo možné vnímať nový trend v rámci vývoja CSS aspektu používateľských rozhraní. Keďže knižnica React využíva architektúru komponentov, z ktorých následne vyskladá celú klientskú časť aplikácie, bolo výhodné v rámci zachovania modularity a prehľadnosti tejto architektúry umožniť implementáciu CSS v rámci konkrétneho komponentu. Tento štýl implementácia CSS v rámci komponentu nazývame aj tzv. „*CSS in JS*“ (*CSS v javaskripte*). Všetky technológie a doplňujúce knižnice, použité pri vývoji nástroja, budú priblížené v nasledujúcich podkapitolách.

3.5.1 Knižnica Styled-components

Knižnica Styled-components ďalej rozvíja „*CSS in JS*“ trend vo forme vytvárania opätovne použiteľných komponentov s konkrétnymi CSS atribútmi. V tomto aspekte sa veľmi dobre dopĺňa s knižnicou React, nakoľko oddeľuje prezentačný aspekt komponentu od samotného komponentu tým, že vytvára čisto funkčný komponent (*functional component*). Tento

funkčný komponent potom udržuje všetky informácie ohľadne jeho prezentácie (*CSS*), a zároveň, rovnako ako u knižnice React, tento komponent prijíma dáta cez tzv. props.

Knižnica Styled-components využíva aktuálnu verziu technológie JavaScript ES6, konkrétne jej novú šablónovaciu funkciu, tzv. tagged-template literals. Tagged-template literals umožňuje vytvárať špeciálnu notáciu, ktorá ako vstup využíva reťazec znakov (*definované CSS atribúty*) a na výstupe poskytuje štruktúrované pole, vytvorené z týchto znakov. Ďalej umožňuje aj tzv. interpoláciu, čo znamená schopnosť vkladať JavaScript ES6 kód na ľubovoľné miesto v reťazci notácie tagged-template literals, pod podmienkou použitia na to určeného syntaxu. Tento aspekt umožňuje vkladať ďalšiu dynamickú funkcionalitu v rámci interpolácie reťazca vkladaného do šablóny. Veľmi výraznou črtou tejto knižnice je aj schopnosť rozširovať CSS štýly už existujúceho komponentu, a to vytvorením nového styled komponentu, ktorý príjme ako funkčný argument už existujúci komponent. Knižnica Styled-components ďalej poskytuje tzv. Theme provider komponent. Jedná sa o komponent, ktorý sa implementuje nad všetkými ostatnými komponentami a obsahuje props s názvom theme, ktorá akceptuje objekt theme. V tomto objekte theme sa definujú všetky opakujúce sa CSS aspekty používateľského rozhrania (*veľkosť fontu, typ fontu, farby UI, a mnohé ďalšie*). Vďaka Theme provider komponentu je možné prehľadne a na jednom mieste definovať základné dizajnové aspekty UI. [14]

3.5.2 Knižnica Styled-system

Knižnica Styled-system je ďalšou nadstavbou nad knižnicou Styled-components. V rámci jej správneho fungovania je preto nutné mať zároveň nainštalovanú aj knižnicu Styled-components. Styled-system ponúka ďalšiu abstrakciu nad vytváraním dynamických štýlov pre komponenty. Táto knižnica je tvorená sadou pomocných funkcií, ktoré pridávajú react komponentom ďalšie props, definujúce konkrétny atribút štýlovania daného komponentu. Tento koncept veľmi dobre funguje či už v súčinnosti s už spomínaným komponentom Theme provider, alebo pri jednoduchom definovaní štýlu konkrétneho komponentu v rôznych prípadoch použitia. Výhodou tejto knižnice je rozsiahla dokumentácia podporovaných pomocných funkcií, vďaka ktorým je možné cez príslušné props dynamicky nastavovať konkrétne CSS atribúty. [15]

3.5.3 Knižnica Rebass

Knižnica Rebass je ďalšou knižnicou, ktorá stavia na knižnici Styled-system (*konkrétne knižnicou poskytované props, ktoré podporujú responzívnosť vyvíjaného UI*), teda

v konečnom dôsledku stavia aj na knižnici Styled-components. Knižnica Rebass poskytuje 8 základných komponentov, na ktorých je postavené v podstate každé používateľské rozhranie. Medzi tieto komponenty patria komponenty: Box, Flex, Text, Heading, Button, Link, Image, Card. Každý z uvedených komponentov rozširuje základný komponent Box, a teda okrem jeho všeobecných props všetky tieto komponenty prijímajú ďalšie rozličné props, ktoré definujú ich CSS atribúty dokumentované na stránkach knižnice Rebass. [16]

3.5.4 Knižnica Axios

Každá SPA (S*ingle Page Aplikácia*) potrebuje získavať potrebné dáta od servera, tak aby ich mohla zobraziť v príslušnej časti používateľského rozhrania. Pre tento účel bola zvolená populárna knižnica Axios, ktorá slúži na realizáciu tzv. promise based HTTP klienta ako aj node.js servera. Slovné spojenie promise based znamená, že knižnica Axios podporuje asynchrónne udalosti, teda reťazenie jednotlivých dotazov a požiadaviek v závislosti na ich naplnení. Výrazne uľahčuje implementáciu komplikovaných dotazov na rôzne služby. Na dotazovanie využíva tzv. XHR (*XML HTTP request*) objekt, ktorým interaguje so serverom [17]. Umožňuje to získavať dáta z URL adresy bez potreby akéhokoľvek načítania webovej stránky (*podpora SPA*). Pri posielaní a prijímaní dát podporuje formát XML rovnako ako aj formát JSON, a zároveň aj poskytuje automatickú transformáciu dát na formát JSON. V prípade vyvíjaného nástroja Axios slúži najmä na realizáciu GET a POST dotazov na API rozhrania kandidátskych služieb bližšie popísaných v Tabuľka 4: Kandidátske API vhodné pre use case. [18]

3.5.5 Knižnica React-final-form

Knižnica React-final-form je verzia knižnice Final-form prispôsobená knižnici React. Knižnica Final-form poskytuje manažment stavu formulárov založený na vzore tzv. observer pattern. Observer pattern je vývojový vzor v rámci ktorého objekt, nazývaný aj subjekt, udržuje stav a určitý zoznam tzv. observer objektov (*relácia 1:M*). Podľa zoznamu observer objektov subjekt notifikuje tieto observer objekty o zmene stavu a následne sa observer objekty aktualizujú. To znamená, že aktualizácia v rámci zobrazovania jednotlivých komponentov formulára sa deje v závislosti od zmeny stavu formulára. [19]

4 Analýza požiadaviek na vyhľadávací nástroj

Firma Pelicantravel.com s.r.o. v súčasnosti disponuje viacerými komplexnými aplikáciami pre vyhľadávanie letových spojení a poskytovanie s nimi spojenými ďalšími službami (*nákup zájazdov, požičiavanie automobilov a mnohé iné*). Všetky tieto aplikácie sú však zamerané na desktop zariadenia. Vo svete a aj v rámci odvetvia cestovného ruchu vnímame čoraz rapídnejšie narastajúci trend využívania menších mobilných zariadení (*tablety, telefóny*). Firma Pelicantravel.com s.r.o. má preto záujem vyvinúť nástroj, ktorý umožní vyhľadávanie letových spojení na akejkoľvek hardvérovej platforme. Dôležité pri tom je, aby nástroj zabezpečil rovnako kvalitný UX (*používateľský zážitok*) na rôznych typoch zariadení (*desktop zariadenia, tablet, mobilný telefón*). Cieľom a zadaním diplomovej práce je zabezpečiť prezenciu služieb firmy Pelicantravel.com s.r.o. na takej platforme, ktorá by dokázala zastrešiť všetky typy zariadení. Je potrebné dodať, že firma Pelicantravel.com s.r.o. poskytuje pomerne komplexné služby v rámci odvetvia cestovného ruchu. Vyvíjaný nástroj na novej platforme bude preto slúžiť z počiatku na vyhľadávanie spiatočných letových spojení.

4.1 Všeobecné požiadavky

Cieľom práce je vytvoriť aplikáciu, ktorá umožňuje jednoduché vyhľadávanie letových spojení. Dôležitou vlastnosťou výslednej aplikácie je, že bude schopná poskytovať rovnaký UX nezávisle na type zariadenia, na ktorom bude zobrazovaná. Očakáva sa, že aplikácia bude plne responzívna. Ďalšou vlastnosťou, ktorou by mala aplikácia disponovať je, že bude schopná určitej prevádzky aj pri obmedzenom, prípadne žiadnom pripojení na internet. Rovnako sa očakáva, že bude schopná notifikovať používateľa v prípade vzniku zmien. Popri týchto hlavných vlastnostiach bude aplikácia tiež poskytovať dodatočné dáta tretích strán, ako napríklad počasie, fakty o destinácii a iné zaujímavosti. V nasledujúcej podkapitole bude definovaná a bližšie priblížená funkcionalita, ktorá bola konzultovaná s pánom Ing. Vladimírom Hudecom z firmy Pelikantravel.com s.r.o.

4.2 Analýza prípadu použitia

Vývoj ekvivalentne komplexnej aplikácie, akou sú súčasné desktop webové aplikácie firmy Pelikantravel.com s.r.o. by bol časovo veľmi náročný a vyžadoval by si viacčlenný tím vývojárov a testerov. Aktuálne firma disponuje aplikáciami, ktoré využívajú živé dáta, ktoré sú neustále aktualizované. Nad týmito dátami firma Pelikantravel.com s.r.o. vytvára vlastné agregácie rôznych dát. Napríklad sa jedná o akciové kalendáre s termínmi akciových letov

za veľmi výhodné ceny. Ďalej disponuje aj službou, ktorá zobrazuje akciové pobyty, dovolenky a v rámci leteniek poskytuje službu, ktorá ponúka tzv. multicity letenky.

Kvôli vyššie uvedeným faktom bude vyvíjaný nástroj viac špecifický, a bude umožňovať vyhľadávanie spiatočných letových spojení. Hlavný use case (*prípad použitia*) sa zameriava na poskytovanie aktuálne cenovo najvýhodnejších spiatočných letových spojení pre konkrétny dátum odletu a príletu, a pre stanovené množstvo a typ pasažierov. Spomínané dáta, ktoré firma Pelikantravel.com s.r.o. agreguje a drží vo vlastnej cache (*úložisko*), nemusia byť vždy aktuálne. Preto bude výsledný nástroj využívať ako hlavnú službu rozhranie služby Pelikan search API, ktorá prehľadáva ponuky všetkých leteckých spoločností v reálnom čase, a teda zabezpečuje aktuálne dáta v momente vyhľadávania. Za účelom využitia rozhrania Pelikan search API je potrebné najprv získať token zo serveru firmy Pelikantravel.com s.r.o.. Na to slúži API rozhranie služby Getsession, ktoré vracia unikátny session token, na základe ktorého je povolené využívanie služby Pelikan search. Pre zabezpečenie dobrého UX sa u vyvíjaného nástroja predpokladá vhodný inteligentný formulár. Pre tento účel bude nástroj využívať rozhranie Autocomplete2 API, ktoré vracia informácie o letiskách v konkrétnej destinácii, a to podľa zadaného reťazca.

Popri vyhľadávaní by mal podľa zadania výsledný nástroj zabezpečovať aj následnú spätnú notifikáciu používateľa, v prípade vzniku nejakej zmeny. V rámci prípadu použitia bude táto vlastnosť využitá na notifikovanie používateľa v prípade vzniku lacnejšieho letového spojenia, vzhľadom na používateľom vyhľadávané letové spojenie. Predpokladá sa teda použitie vhodnej technológie za účelom implementácie notifikačného aspektu nástroja. Okrem toho sa predpokladá aj implementácia ukladacieho mechanizmu, podľa ktorého si nástroj bude pamätať, aké letové spojenia používateľ vyhľadával.

Dôležité je tiež myslieť na fakt, že najmä v cestovnom odvetví môžu nastať situácie, kedy má používateľ obmedzený, prípadne žiadny prístup na internet. Štandardné správanie webových aplikácií je, že umožňujú prevádzku len pri garancii internetového pripojenia. Bude potrebné zabezpečiť schopnosť prevádzky nástroja aj pri takýchto podmienkach. Tento aspekt súvisí s predošlým bodom (*vytvorenie úložiska pre vyvíjaný nástroj*). Implementácia vhodného úložiska a ukladacieho mechanizmu nástroja umožní vyvíjanému vyhľadávaciemu nástroju prevádzku pri obmedzenom, prípadne žiadnom pripojení na internet a rovnako bude dôležitá aj pri implementácií spätných notifikácií používateľa.

Okrem samotného vyhľadávania letových spojení a notifikovania používateľa v prípade vzniku lacnejšieho letového spojenia bude výsledný nástroj poskytovať používateľovi ďalšie relevantné a doplňujúce dáta. Pri analýze prípadu použitia bol vykonaný prieskum rôznych ďalších služieb, ktoré by boli vhodné pre integráciu do vyhľadávacieho nástroja. Treba poznamenať, že samotný prieskum sa sústreďoval najmä na voľne dostupné (*open source*) služby, avšak boli identifikované aj komerčné služby, ktoré by boli pre nástroj vhodné. Opäť na základe konzultácií s pánom ing. Vladimírom Hudecom boli vybraté ďalšie API (*Application Programming Interface*) rozhrania vhodných služieb pre výsledný nástroj. Jedná sa najmä o open source API voľne dostupných služieb ako Numbeo, ale aj komerčných služieb ako napríklad World weather online .

World weather online je služba, poskytujúca rozsiahle dáta o počasí. Konkrétne poskytuje predpoveď na najbližších 14 dní, historické dáta o počasí (*od júla 2008*), historické dáta pre námorné posádky (*od januára 2015*), predpoveď počasia v horách, lyžiarskych rezortoch, ako aj predpoveď pre posádky lodí na najbližších 7 dní. Všetky dáta poskytované API rozhraniami je možné získať vo forme XML, alebo JSON. Túto službu využívajú mnohé vládne a súkromné organizácie a je veľmi obľúbená najmä v námornom odvetví. [20]

Služba Numbeo disponuje najväčšou databázou na svete popisujúcou mestá a krajiny. Dáta v databáze neustále dopĺňajú a aktualizujú používatelia z celého sveta. Služba Numbeo poskytuje jednak aktuálne, ale aj historické informácie o podmienkach pre život v jednotlivých krajinách, vrátane výdavkov na život, cien bývania, zdravotnej starostlivosti, dopravy, kriminality a znečistenia. Služba Numbeo bola založená v roku 2009 [21]. Služba Numbeo prehlasuje, že je absolútne nezávislá a nie je ovplyvňovaná žiadnou vládnou organizáciou. Bohužial, služba nie je prístupná bezplatne, a teda bude potrebné zakúpenie licencie pre firmu Pelikantravel.com s.r.o..

Pre vyvíjaný nástroj by sa veľmi hodila služba Sygic places. Firma Sygic je mimoriadne úspešná spoločnosť zo Slovenska zaoberajúca sa vývojom GPS navigačného softvéru. Ponúka širokú škálu služieb v rámci odvetvia navigačných systémov, aplikovateľnú na všetky typy zariadení aktuálne dostupných na trhu (*mobily, tablety, smart hodinky, autá a iné*). Služba Sygic places umožňuje zobrazenie tzv. points of interests (*body záujmu*) na interaktívnej mape. Táto služba umožňuje aj prevádzku bez prístupu na internet. V rámci prípadu použitia vyvíjaného nástroja by táto služba spolu s World weather online vytvárala naozaj značnú hodnotu pre používateľa, nakoľko by bol informovaný o počasí a podstatných

miestach v jeho vybranej destinácii. Rovnako vhodné by bolo aj použitie služby Numbeo, vďaka ktorej by mal používateľ na jednom mieste ďalšie charakteristické dáta bližšie popisujúce danú destináciu. Všetky kandidátske API rozhrania služieb, ktoré by nástroj vedel potenciálne využiť nájdeme zoradené a stručne priblížené v Tabuľke 4 uvedenej nižšie.

Tabuľka 4: Kandidátske API vhodné pre use case

API rozhranie služby	Stručný popis
Pelikan search API	Živé vyhľadávanie letových spojení
Getsession API	Vytvorenie spojenia so serverom firmy
Autocomplete2 API	Informácie o letiskách podľa zadaného reťazca
Cache API	Úložisko typu Cache na strane klienta
IndexedDB API	Úložisko držiace štruktúrované dáta na strane klienta
Historical weather API	Rozhranie pre službu World weather online - počasie
Numbeo API	Rozhranie pre službu Numbeo – fakty o destináciách
Sygic places API	Rozhranie pre službu Sygic places – zaujímavé lokality

4.3 Zhrnutie

Vyvíjaný nástroj bude vo výsledku umožňovať vyhľadávanie spiatočných letových spojení, pre konkrétne dátumy odletu a príletu, a pre konkrétne množstvo a druh pasažierov. Nástroj bude schopný určitej prevádzky aj pri obmedzenom prípadne žiadnom pripojení na internet. Bude využívať viaceré služby (*Tabuľka 4*), za účelom vytvorenia ponuky letového spojenia, a zároveň bude umožňovať spätnú notifikáciu používateľa, v prípade vzniku lacnejšieho letového spojenia. Dôležitým aspektom je zabezpečenie rovnako kvalitného UX naprieč všetkými typmi zariadení, a to z hľadiska veľkosti ich zobrazovacej plochy. Výsledný nástroj musí byť teda plne responzívny. Zo zistených faktov počas analýz dostupných technológií vhodných pre riešenie zadania a analýzy prípadu použitia nástroja, sa javí byť najvhodnejšou platformou pre vyvíjaný nástroj práve PWA (P*rogresívna Webová Aplikácia*). Implementácia nástroja sa bude realizovať pomocou knižnice React a s ňou spojených ďalších podporných knižníc.

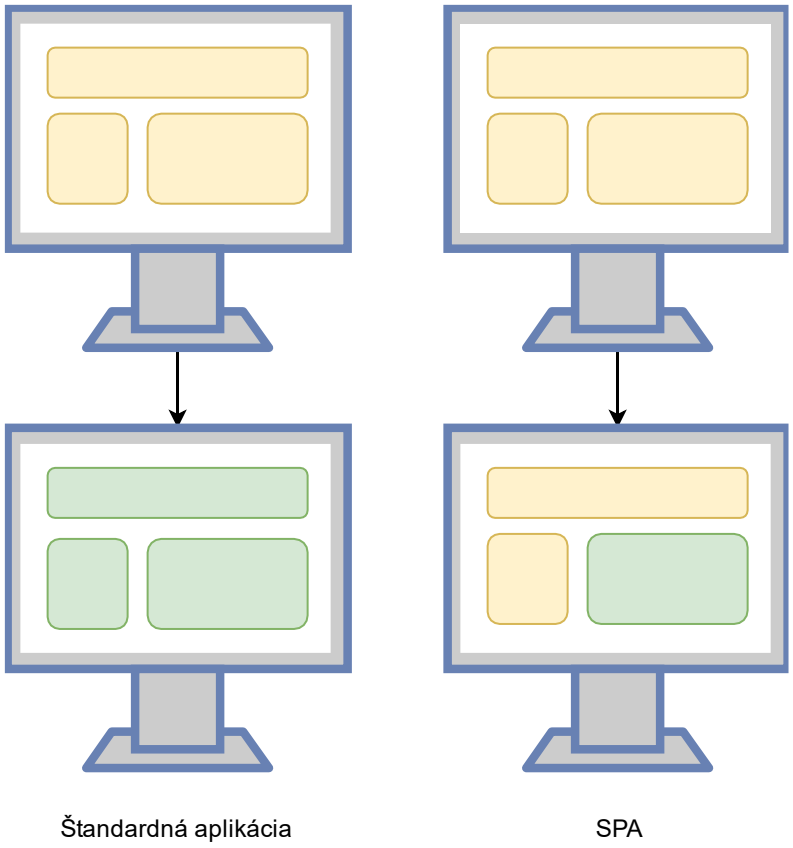
5 Používateľské rozhranie

Z vykonaných analýz vyplýva, že výsledný nástroj na vyhľadávanie letových spojení bude postavený na platforme PWA (*Progresívna Webová Aplikácia*). V rámci rozboru technológií používaných pri vývoji klientskej časti webových aplikácií boli rozobraté tri, v dobe písania práce najpopulárnejšie JavaScript knižnice. Vzhľadom na vykonaný rozbor a fakt, že vo firme Pelikantravel.com s.r.o. využívajú vývojárske tímy čoraz viac knižnicu React, bolo rozhodnuté, že aj navrhovaný nástroj bude vyvíjaný za pomoci JavaScript knižnice React. Okrem tejto knižnice bude používateľské rozhranie a vôbec celá klientská časť aplikácie tvorená technológiou HTML 5 (*HyperText Markup Language 5*) a ďalšími knižnicami, ktoré sú zaoberajú technológiou CSS 3 (*Cascading Style Sheets 3*). Keďže knižnica React je založená na architektúre komponentov, používateľské rozhranie bude vyskladané zo sady vyvinutých komponentov, pri tvorbe ktorých sa bude dbať najmä na ich opätovné používanie. To zabezpečí výslednú modularitu používateľského rozhrania a v konečnom dôsledku aj celej klientskej časti aplikácie. Všetky nástroje a komponenty použité pri vývoji používateľského rozhrania sú priblížené v tejto kapitole.

5.1 Aplikácia typu single page

Medzi stanovenými kľúčovými vlastnosťami aplikácie boli okrem iného uvedené aj rýchlosť a zabezpečenie rovnako kvalitnej user experience naprieč všetkými poprednými hardware platformami. Práve kvôli týmto parametrom bude aplikácia využívať dizajnový vzor tzv. SPA (*Single Page Aplikácie*). Pri tomto vzore aplikácie, sa presúva zobrazovanie dát zo servera, tzv. server-side rendering, na stranu klienta, tzv. client-side rendering. Pri server-side renderingu sa každý nový obsah pre aplikáciu generuje serverom, a to vo forme nových HTML dokumentov, ktoré server zašle klientovi. Pri client-side renderingu sa nový obsah generuje dynamicky na strane klienta, vďaka schopnosti JavaScriptu manipulovať DOM (*Document Oriented Model*) konkrétneho HTML dokumentu. V rámci tradičnej architektúry webových aplikácií využívajú aplikácie index.html stránku, ktorá odkazuje na ďalšie HTML stránky na konkrétnom serveri. SPA dizajnový vzor využíva jeden HTML dokument, na ktorom je nasadená celá aplikácia. Umožňuje to používateľovi plynulú a rýchlejšiu interakciu s aplikáciou, zatiaľ čo všetky potrebné dáta sú získavané postupne a dynamicky na pozadí, v závislosti na používateľových akciách. Znamená to, že celá aplikačná logika a zobrazovanie dát prebieha na strane klienta a server slúži len čisto na poskytovanie a prijímanie dát, v závislosti od požiadaviek aplikácie. Tento základný principiálny rozdiel medzi tradičnými webovými aplikáciami a tzv. single page aplikáciami môžeme vidieť

schématicky znázornený na Obrázku 19. Štandardná aplikácia zobrazuje každú odpoveď servera ako nový, serverom vygenerovaný HTML dokument (*v obrázku odlíšené žltou a zelenou farbou v obrazovkách vľavo*). Naopak u SPA samotná aplikácia generuje nový obsah prijatý od servera, a to stále v rámci rovnakého HTML dokumentu (nový obsah je v obrázku odlíšený zelenou farbou v obrazovkách vpravo).



Obrázok 19: Štandardná aplikácia VS SPA

Výhody:

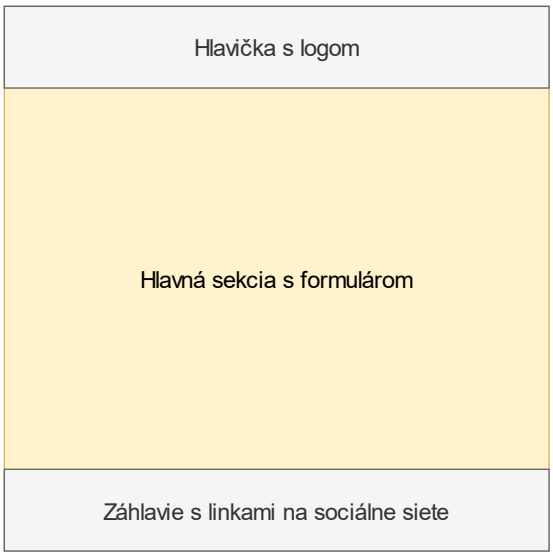
- Oddelenie dátovej od prezentačnej vrstvy. Vďaka API rozhraniam back-end služieb je získavanie dát a ich zobrazovanie (*rendering*), ako aj odosielanie plne v kompetencii SPA.
- Zabezpečene plynulého UX (*používateľský zážitok*) a zníženie nárokov na server, nakoľko server nemusí na každý dotaz aplikácie vytvárať nový HTML dokument, ale poskytne len potrebné dáta.

Nevýhody:

- Vyžaduje podporu technológie JavaScript.

5.2 Návrh používateľského rozhrania

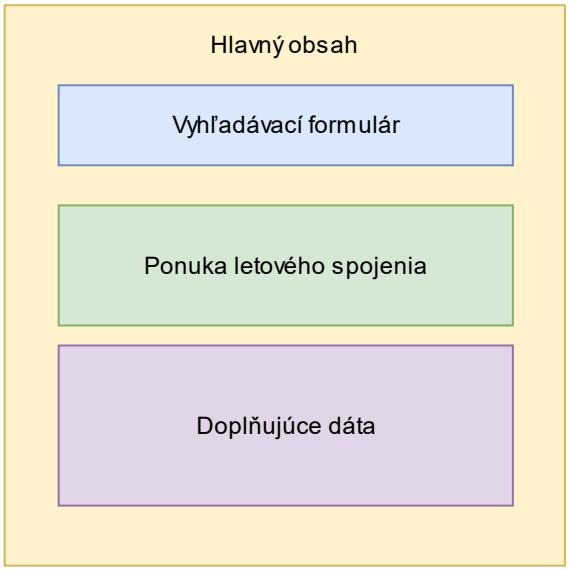
Zo zadania vyplýva, že hlavný účel vyvíjaného nástroja bude jednoduché vyhľadávanie letových spojení. Z doterajších analýz bolo stanovené, že nástroj bude postavený na platforme PWA (P*rogresívna Webová Aplikácia*) a bude sledovať vzor SPA *(Single Page Aplikácia*). Predpokladá sa vytvorenie jednoduchého používateľského rozhrania s vyhľadávacím formulárom a zobrazovacou oblasťou pre vyhľadané letové ponuky a ďalšie doplňujúce informácie. Hlavný koncept používateľského rozhrania je znázornený na Obrázku 20 uvedenom nižšie a pozostáva z troch hlavných častí a to hlavičky, hlavného obsahu a záhlavia.



Obrázok 20: Základný koncept používateľského rozhrania

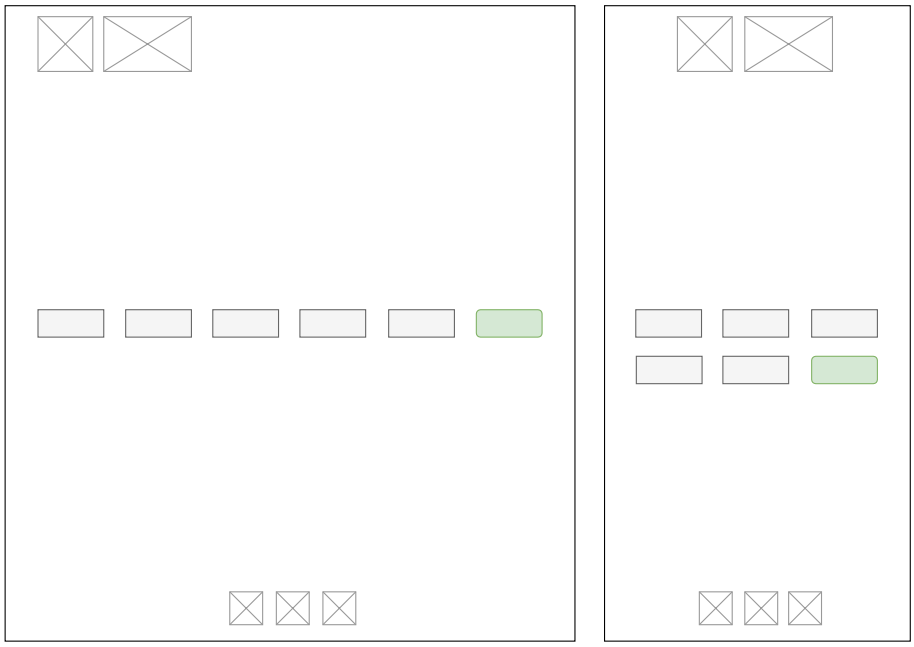
Hlavička bude obsahovať logo firmy Pelikantravel.com s.r.o. a záhlavie bude obsahovať linky na sociálne siete. Hlavný obsah bude obsahovať spomínaný vyhľadávací formulár, pod ktorým sa bude zobrazovať karta s ponúkaným letovým spojením a zobrazenými doplňujúcimi dátami pre danú destináciu. Vyhľadávací formulár bude podporovať funkcionality pre vyhľadávanie spiatočných letov pre konkrétny dátum odletu a konkrétny dátum príletu. Ďalej bude umožňovať voľbu počtu a druhu pasažierov. Schéma konceptu hlavného obsahu je zobrazená na Obrázku 21 uvedenom na nasledujúcej strane. Keďže náš prípad použitia sa zameriava na vyhľadávanie cenovo najvýhodnejších ponúk letových spojení, typ vyhľadávaných letových spojení bude implicitne nastavený na letové spojenie triedy economy. Dodržanie čo maximálnej jednoduchosti pri navrhovaní používateľského rozhrania výrazne zjednoduší jeho implementáciu tak, aby poskytovalo plnú repsonzívnosť pri rôznych zariadeniach, ktoré budú aplikáciu využívať (*mobilný telefón, desktop počítač,*

tablet a pod.). Keďže sa jedná o tzv. single page aplikáciu, technológie využité pri vývoji používateľského rozhrania sa budú zameriavať na riešenie problematiky vývoja tzv. klientskej časti aplikácie. Hlavnými technológiami, ktoré sa využívajú pri vývoji v tejto oblasti sú samozrejme HTML, CSS a JavaScript. V predchádzajúcich častiach práce sme si priblížili JavaScript knižnicu React. Táto kľúčová technológia, vďaka ktorej sa celý nástroj vyvinie ako stromová štruktúra tvorená súborom komponentov, bola však počas vývoja doplnená o ďalšie vhodné knižnice a nástroje, vďaka ktorým sa uľahčil a zrýchlil celkový vývoj používateľského prostredia.

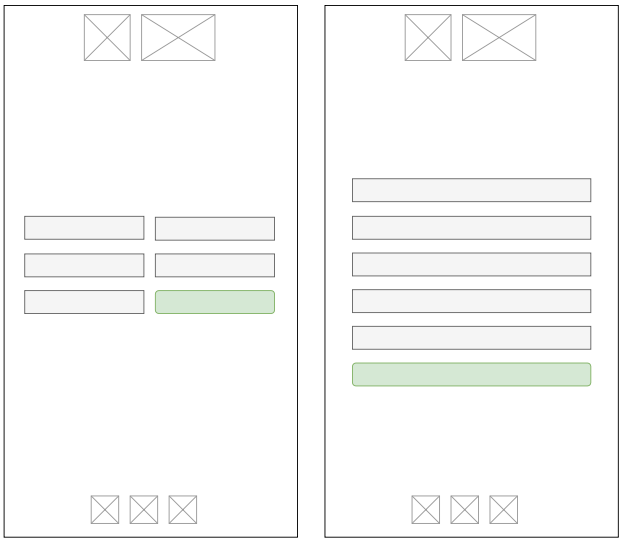


Obrázok 21: Koncept hlavného obsahu

Základný koncept používateľského rozhrania a koncept hlavného obsahu platí štandardne rovnako pre všetky aplikácie typu single page. Návrh jednotlivých častí základného konceptu používateľského rozhrania a jeho hlavného obsahu (*vyhľadávací formulár, ponuka letového spojenia, doplňujúce dáta*) je zobrazený na nasledujúcich stranách pomocou wireframe návrhov. Na Obrázku 22 na nasledujúcej strane sú znázornené wireframe návrhy obrazoviek používateľského rozhrania pre desktop PC a tablet. Jedná sa o zariadenia, ktoré disponujú dvomi najväčšími zobrazovacími plochami. Rozloženie prvkov používateľského rozhrania zostáva viac menej rovnaké, až na formulár. Pri zmenšení zobrazovacej plochy dochádza k centrovaniu obsahu hlavičky na stred (*logo a popis*) a mení sa rozloženie prvkov formulára, pri čom sa postupne prvky organizujú z riadku do stĺpca. Formulár sa skladá z piatich vstupov a jedného tlačítka (*označené vo wireframe návrhu na zeleno*). Správanie rozloženia formuláru pri zobrazení na mobilných zariadeniach je znázornené na Obrázku 23.

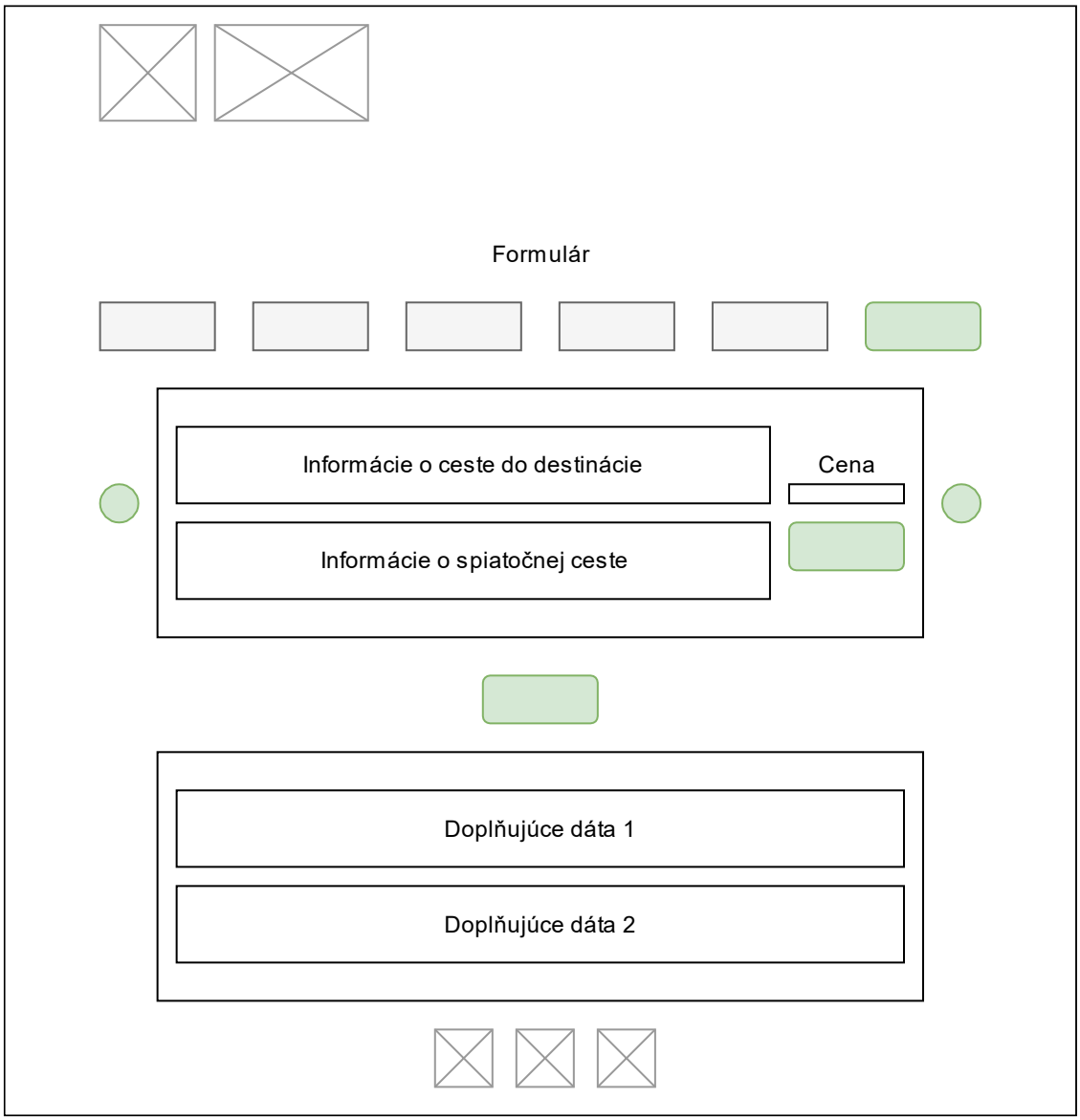


Obrázok 22: Wireframe základného desktop a tablet zobrazenia.



Obrázok 23: Wireframe základného zobrazenia pre mobilné zariadenia.

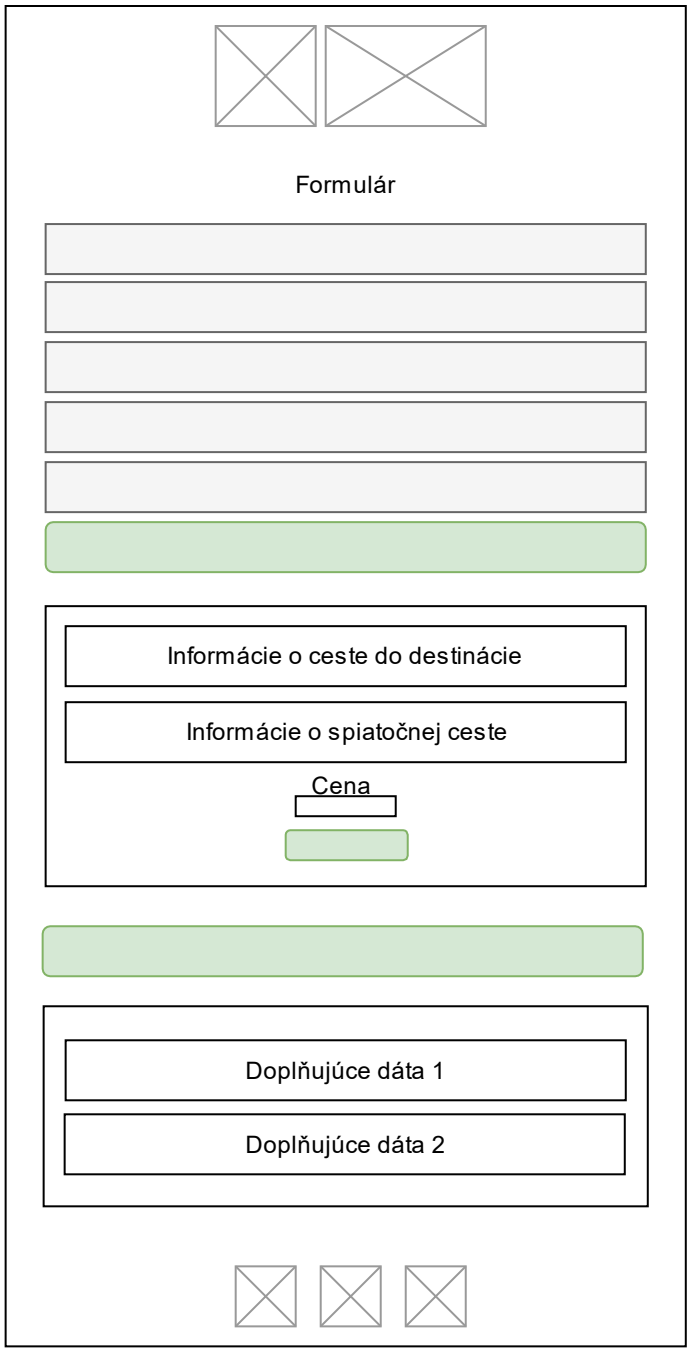
Uvedené wireframe návrhy zobrazujú očakávané responzívne správanie formuláru. Zároveň tvoria návrh základného obsahu vyvíjanej aplikácie typu single page. V momente ako používateľ vyplní formulár (*odletové letisko a letisko v konkrétnej destinácii, príslušné dátumy odchodu a návratu a počet a druh pasažierov*) sa obsah aplikácie doplní o zobrazenie ponuky letových spojení, a rovnako aj doplňujúcich informácii ohľadne vybranej destinácie. Toto zobrazenie je podrobne znázornené na wireframe návrhu v Obrázku 24 uvedenom na nasledujúcej strane.



Obrázok 24: Wireframe desktop a tablet zobrazenia výstupu formuláru

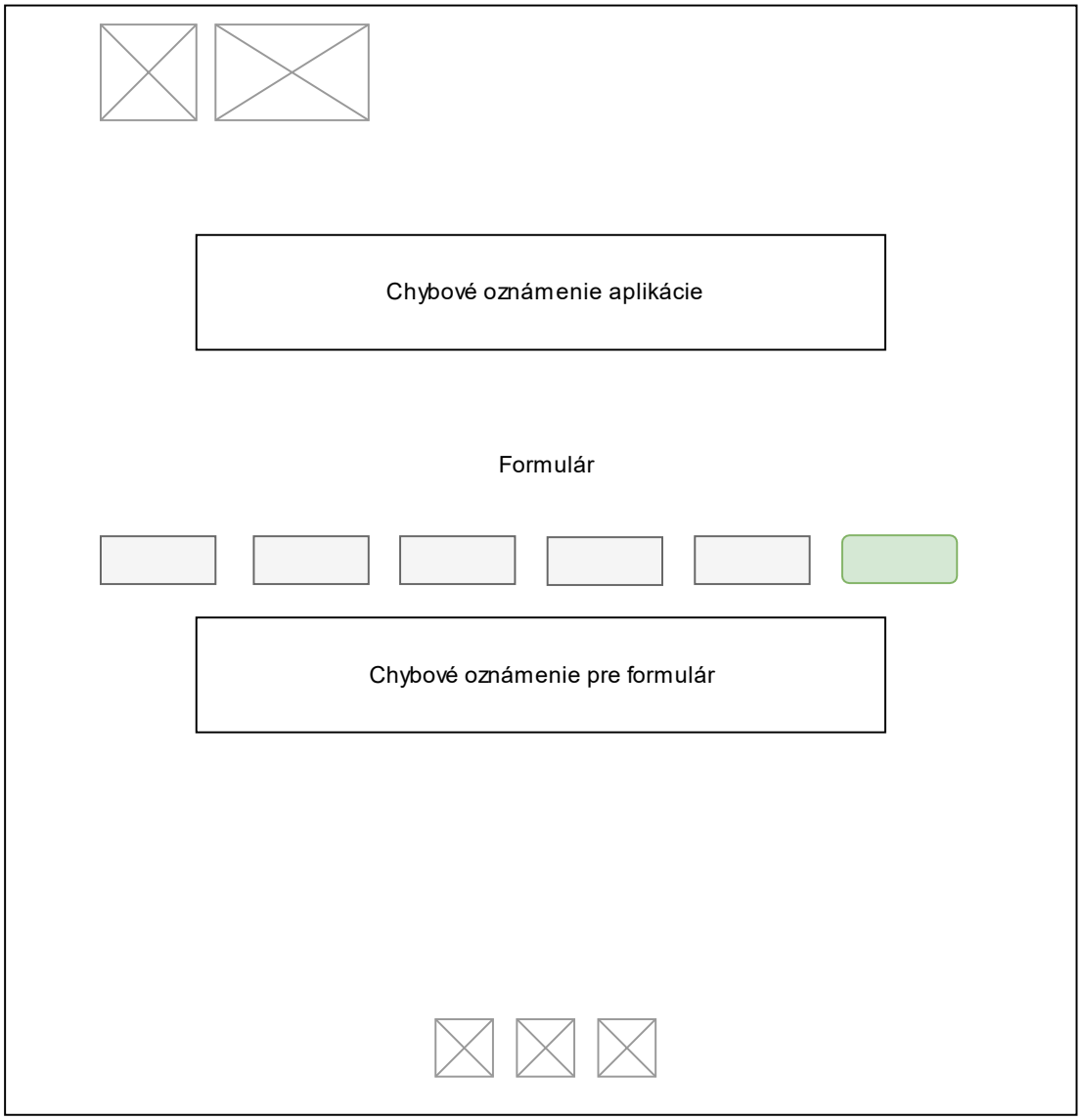
Vo wireframe návrhu zobrazenia výstupu formuláru sú zelenou zvýraznené ovládacie tlačítka. Hlavná karta s letovou ponukou je navrhnutá ako tzv. carausel (*kolotoč*) komponent. Tento komponent umožňuje zobrazovanie elementov v štýle horizontálneho posúvania. Tento návrh bol zvolený práve z dôvodu responzívnosti používateľského rozhrania. Carausel komponent poskytuje lepší UX prezerania letových ponúk na mobilných zariadeniach, ako napríklad zobrazovanie ponúk v kartách pod sebou. Po stranách carausel komponentu sú teda ovládacie tlačítka pre otáčanie obsahu komponentu (*prezeranie letových ponúk*). V rámci zobrazenej letovej ponuky sú informácie o letovom spojení rozdelené do dvoch stĺpcov. Prvý z nich zaberá približne štyri pätiny priestoru (*zobrazovanie na desktop PC*) a obsahuje dva riadky (jeden pre zobrazenie informácii o ceste do destinácie a druhý

pre zobrazenie informácii o spiatočnej ceste). Druhý stĺpec zaberá približne jednu pätinu priestoru (*zobrazovanie na desktop PC*) a obsahuje zobrazenú cenu a ovládacie tlačítko pre rezerváciu letenky. Pod carausel komponentom zobrazujúcim letové ponuky sa nachádza ďalšie ovládacie tlačítko, ktoré slúži na ovládanie zobrazovania doplňujúcich dát o destinácii. Návrh karty s doplňujúcimi dátami predpokladá aspoň dva riadky pre zobrazenie dvoch rôznych služieb poskytujúcich doplňujúce dáta. Wireframe návrh mobilného zobrazenia výstupu aplikácie je uvedený na Obrázku 25 nižšie.



Obrázok 25: Wireframe mobilného zobrazenia výstupu

V rámci návrhu sa predpokladá s aspoň dvomi typmi chybových hlásení. Prvé pre zobrazenie chybového hlásenia celej aplikácie (*offline stav aplikácie*) a druhé pre zobrazenie chybového hlásenia pri neúspešnom výsledku vyhľadávania letových spojení. Rozloženie zobrazení chybových správ aplikácie a formuláru je zobrazené vo wireframe návrhu na Obrázku 25 uvedenom nižšie.



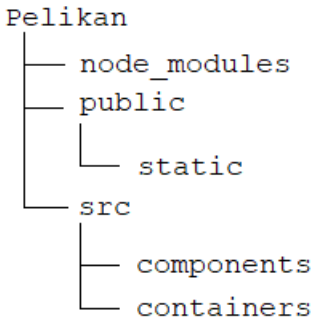
Obrázok 26: Wireframe zobrazenia hlásenia chýb

5.3 Vývoj používateľského rozhrania

Z doterajších vykonaných analýz vyplýva, že používateľské rozhranie bude postavené na súčinnosti vyššie popísaných tzv. „front-end“ technológiách, a pri ich správnej implementácii bude vyhovené všetkým požiadavkám na používateľské rozhranie. Najmä sa jedná o jeho responzívnosť naprieč všetkými štandardnými typmi zariadení. Na začiatok je potrebné poznamenať, že všetky uvedené technológie a knižnice použité pri implementácii používateľského rozhrania sú aktuálne a podporujú ich všetky štandardné webové prehliadače. Jediná možnosť, ktorá by zabránila zobrazeniu používateľského rozhrania, by nastala v prípade, že by mal používateľ na svojom prehliadači zakázané spúšťanie JavaScript súborov, na čo však používateľa upozorní index.html dokument, na ktorom je celá aplikácia nasadená.

5.3.1 Adresárová štruktúra aplikácie

Základná adresárová štruktúra aplikácie bola vytvorená za pomoci nástroja create-react-app. Tento nástroj poskytuje skript, ktorý vytvára komfortné prostredie pri vytváraní React aplikácií. Je veľmi vhodný najmä pre začiatočníkov, nakoľko podporuje rapídnu tvorbu SPA. Pri použití skriptu create-react-app sa vytvorí vývojové prostredie, ktoré umožňuje využívať najnovšie, aktuálne prvky technológie JavaScript a knižnice React. Na prevádzku tohto nástroja je potrebná inštalácia JavaScript knižnice Node verzie 6.x.x a novšej, a rovnako aj CLI (*Command Line Interface*) pre túto knižnicu, napríklad štandardné rozhranie npm verzie 5.2 a novšej [22]. CLI rozhranie npm ďalej umožňuje prevádzku lokálneho servera node.js, a to tak, že vyvíjaný JavaScript (*React*) kód po spustení automaticky kompiluje na pozadí, čím umožňuje editovať vyvíjaný kód za behu aplikácie. Znamená to, že zmeny v kóde vidíme okamžite (*po uložení*) skompilované vo webovom prehliadači. Create-react-app plne vyhovuje účelu vytvoriť základnú kostru aplikácie a zabezpečiť manažment všetkých jej tzv. „*dependencies*“ (*ďalšie knižnice integrované do aplikácie*). Na manažment všetkých ďalších integrovaných knižníc a nástrojov využíva create-react-app knižnice Babel a Webpack, ktoré automaticky konfiguruje.



Obrázok 27: Adresárová štruktúra

Vytvorená adresárová štruktúra je uvedená na Obrázku 22 uvedenom vyššie a obsah jednotlivých súborov je popísaný v Tabuľke 5 uvedenej na nasledujúcej strane. Adresárová štruktúra sa skladá z vrchného adresáru, ktorý obsahuje celú vyvíjanú aplikáciu tvorenú z niekoľkých adresárov a súboru `package.json`, ktorý definuje metadáta ohľadne verzie aplikácie, jej závislostiach na ďalších knižniciach a dostupných skriptoch pre CLI rozhranie. Vygenerovaná adresárová štruktúra bola pre prehľadnosť a podporu modularity systému doplnená o adresáre `components` a `containers`, vytvorené v adresári `src`. Ďalej bol doplnený adresár `public` o adresár `static`, ktorý obsahuje všetky statické prvky vyskytujúce sa v aplikácii (*logo, text k logu, ikony sociálnych sietí*).

Adresár `src` okrem adresárov s komponentmi a kontajnermi obsahuje JavaScript súbor `index.js`, ktorý je nasadený v hlavnej stránke aplikácie `index.html`. Vďaka tomuto súboru je pomocou knižnice React-dom nasadený hlavný komponent `App.js`, zastrešujúci celú vyvíjanú aplikáciu do hlavnej stránky `index.html`. Konkrétne je hlavný komponent `App.js` nasadený do `<div id="root"></div>` elementu hlavnej stránky `index.html`. Nasadenie hlavného komponentu `App.js` je realizované využitím knižnice React-dom. Jedná sa o knižnicu v rámci knižnice React, ktorá ma na starosti tzv. Virtual-DOM (*virtuálny Document Object Model*). Virtual-DOM je virtuálna reprezentácia DOM (*Document Object Model*) aplikácie vďaka ktorej knižnica React dokáže manipulovať so skutočným DOM aplikácie. Deje sa to takým spôsobom, že namiesto neustáleho priameho aktualizovania skutočného DOM aplikácie, knižnica React v závislosti na zmenách state (*stavu*) a props (*vlastností*) konkrétneho komponentu, upraví virtuálny DOM aplikácie a následne vypočíta zmeny v porovnaní s predchádzajúcou verziou virtuálneho DOM. Až v tomto momente knižnica React upraví potrebné časti skutočného DOM aplikácie. Tento

proces je kľúčovou črtou knižnice React, pretože umožňuje rýchlejšie zobrazovanie zmien DOM v rámci používateľského rozhrania.

Tabuľka 5: Popis adresárov aplikácie

Názov adresára	Popis
Pelikan	Hlavný adresár s aplikáciou
node_modules	Adresár obsahujúci integrované knižnice
public	Verejný adresár obsahujúci adresár static, súbor manifest.json a index.html
static	Verejný adresár obsahujúci statické prvky aplikácie
src	Adresár obsahujúci zdrojové súbory a adresáre aplikácie
components	Vytvorený adresár obsahujúci všetky komponenty aplikácie
containers	Vytvorený adresár obsahujúci všetky kontajnery, ktoré obaľujú komponenty aplikácie

Okrem spomínaného súboru `index.js` a hlavného komponentu `App.js` sa v rámci adresáru `src` vyskytujú ešte JavaScript súbory zaoberajúce sa skriptom service worker a v rámci implementácie sa v adresári `src` vytvoril aj súbor `theme.js`, ktorý definuje základné dizajnové aspekty aplikácie.

5.3.2 Téma

V adresári `src` sa nachádza súbor `theme.js`, obsahujúci objekt s nadefinovanými CSS dizajnovými aspektami nástroja. Tento objekt slúži ako šablóna pre knižnicu styled-components, ktorá bola priblížená v tejto kapitole a je využívaná ako hlavný nástroj pri vývoji CSS aspektov aplikácie. Konkrétne je súbor `theme.js` využívaný komponentom `ThemeProvider`. Sú v ňom definované farby použité v rámci používateľského rozhrania (*objekt* `colors`), ďalej štýly tieňovania a responzívne aspekty témy (*slúžiace pre props* `space` *a* `breakpoints` *knižnice Rebass*), ktoré sú definované ako polia. Pole `space` obsahuje čísla, ktoré predstavujú škálu zobrazovania (*odkazujú sa na pomer veľkosti vzhľadom na zobrazovaciu plochu*) a pole `breakpoints` definuje tri tzv. body zlomu, a to v jednotkách `em`, ktoré sú definované ako pomer k veľkosti používaného fontu.

Objekt `theme` je importovaný v hlavnom, vrchnom komponente celej aplikácie, v súbore `App.js`. V rámci tohto hlavného komponentu je v `render` metóde, ktorá slúži na

zobrazovanie, implementovaný spomínaný komponent `ThemeProvider` s definovanou props `theme`. Do props `theme` je priradený objekt `theme` so všetkými CSS dizajnovými aspektami. Komponent `ThemeProvider` je implementovaný nad hlavným `App` komponentom. Vďaka tomu poskytuje dynamicky všetky CSS nastavenia všetkým ostatným komponentom nachádzajúcich sa pod ním, čo uľahčuje a sprehľadňuje prácu s CSS aspektami používateľského rozhrania.

5.3.3 Komponenty a kontajnery

Adresár `src` obsahuje hlavný komponent `App.js`, ktorý reprezentuje vrch stromovej architektúry komponentov akejkoľvek React aplikácie. V bežnej praxi sa celá klientská logika používateľského rozhrania rozdeľuje a implementuje v príslušných komponentoch v rámci používateľského rozhrania. Hlavnou funkciou hlavného, vrchného komponentu `App.js` je import základných komponentov a kontajnerov používateľského rozhrania a prípadne aj implementácia šablóny pre komponent `ThemeProvider`, ktorý sa implementuje použitím knižnice Styled-components. Pre ujasnenie je potrebné dodať, že kontajnery sú vo svojej podstate komponenty knižnice React, ale v rámci praxe vývoja front-end časti aplikácií sa nazývajú kontajnery, kvôli ich funkcií zaobaľovať prvky používateľského rozhrania.

V rámci implementácie nástroja na jednoduché vyhľadávanie letových spojení sa dbalo najmä na fakt, že sa nástroj bude ďalej rozvíjať a udržovať. Preto sa pri vývoji jednotlivých komponentov dodržovali zásadné konvencie zachovania modularity používateľského rozhrania. Znamená to, že boli implementované jednotlivé prvky používateľského rozhrania tak, aby podporovali svoje opätovné používanie. Tieto základné UI prvky sú vytvorené pomocou knižnice Styled-components, ktorá vďaka podpore funkcie tagged-template literals a jej interpolácii, umožňuje dynamicky nastavovať (*podľa implementovanej témy komponentu* `ThemeProvider`) CSS aspekty UI prvkov (*tlačítka, vstupy, ikony a ďalšie*). Obsah adresárov `components` a `containers` je popísaný v Tabuľka 6: Adresár containersTabuľka 6 a

Tabuľka 7. Adresár `components` obsahuje všetky komponenty, z ktorých je zložené používateľské rozhranie, pri čom adresár `containers` obsahuje komponenty, ktoré vyskladávajú časti používateľského rozhrania z komponentov adresára `components`. Obidva adresáre obsahujú súbor `index.js`, ktorý slúži na exportovanie všetkých komponentov, čo zľahčuje import a následné použitie jednotlivých komponentov.

Tabuľka 6: Adresár containers

Názov súboru	Popis
Containers.js	Hlavné kontajner komponenty
Footer.js	Komponenty zastrešujúce záhlavie UI
Header.js	Komponenty zastrešujúce hlavičku UI
Results.js	Komponenty určené pre zobrazenie vyhľadávania
SearchForm.js	Komponenty implementujúce vyhľadávací formulár
Info.js	Komponenty zastrešujúce doplňujúce dáta o destinácii
Notiffication.js	Komponenty zabezpečujúce notifikácie
index.js	Exportovací súbor pre všetky kontajnery UI

Tabuľka 7: Adresár components

Názov súboru	Popis
Button.js	Ovládacie tlačítko
Icon.js	Ikona sociálnych sietí
Input.js	Textový vstup formuláru
InputHolder.js	Komponent riešiaci umiestnenie komponentu Input
InputWithSuggestion.js	Implementácia inteligentného vstupu s automatickým navrhovaním vyhľadávaného letiska
TravelersInput.js	Komponent slúžiaci pre definíciu počtu a druhu pasažierov
Layout.js	Komponenty implementujúce štandard FlexBox
Spinner.js	Komponent zobrazujíci načítavanie
Typography.js	Komponenty zastrešujúce CSS aspekty použitých fontov
index.js	Exportovací súbor pre všetky komponenty UI

Dôležitou požiadavkou na nástroj bola aj jeho responzivita, teda používateľské rozhranie sa muselo implementovať tak, aby poskytovalo kvalitný UX bez ohľadu na typ zariadenia, na ktorom je zobrazované. Túto problematiku rieši najmä vhodná implementácia kombinácie knižníc Styled-components, Styled-system a Rebass, konkrétne v adresári `components`, v súbore `layout.js`. V tomto súbore bol za pomoci spomenutých knižníc implementovaný nový štandard v rámci CSS technológie, tzv. FlexBox. FlexBox je

novinkou v rámci CSS. Zameriava sa na riešenie problematiky zobrazovania používateľských rozhraní a kvality UX, a to najmä pri zobrazovaní na menších obrazovkách (*mobilné telefóny a tablety*). Inicializuje sa nastavením CSS atribútu zapúzdrujúceho (*kontajner*) elementu `display: flex`. Následne sa všetky elementy vo vnútri kontajnera stávajú tzv. `flex-items`. Po tejto inicializácii štandard FlexBox poskytuje abstrakciu nad prepočítavaním veľkostí a umiestňovaním elementov v rámci kontajneru. V súbore `layout.js` sú implementované základné stavebné bloky používateľského rozhrania. Menovite sa jedná o implementované funkčné (*functional*) komponenty `Column`, `Div`, `Row`, `Cell`, ktoré stavajú na komponentoch `Flex` a `Box` knižnice Rebass. Tieto komponenty využívajú pomocnú funkciu space (*umožňuje mimo iné nastaviť* `height props`) knižnice Styled-system. Použité nástroje (*knižnice*) boli prepojené a ďalej rozšírené práve knižnicou styled-components a využívajú mierky definované v objekte `theme`. V nedávnej praxy sa pri vývoji responzívneho aspektu UI využívali tzv. media queries, pomocou ktorých sa staticky implementovali CSS atribúty, pre konkrétne veľkosti zobrazovacej plochy (*tzv. viewport*). V rámci implementácie nahradila media-queries vynikajúca synergia spomínaných troch knižníc. Veľmi vhodná je práve knižnica Rebass, ktorá umožňuje definovať pomery zobrazovania v závislosti od definovaných bodov zlomu obrazovky v objekte `theme` (*pole* `breakpoints`). To znamená, že namiesto opätovnej implementácie CSS pre každé media-query, sa pomocou `width props` dynamicky implementovala responzívnosť jednotlivých stavebných blokov používateľskeho rozhrania. Toto riešenie výrazne pridalo na prehľadnosti vyvinutého kódu.

Asi najdôležitejšou časťou vyvíjaného nástroja je bez pochýb jeho vyhľadávací formulár. Vďaka spomenutej implementácie knižníc Styled-components, Styled-system a Rebass je vyhľadávací formulár plne responzívny a poskytuje rovnako kvalitné UX či už pre rôzne mobilné telefóny, tablety, ale aj desktopové zariadenia. V rámci vývoja UI je veľmi dôležitá validácia konkrétneho formulára, ktorá zabezpečuje správnu funkcionalitu formulára. Validácia má na starosti garancie správnych vstupov, ktoré očakávajú rôzne API rozhrania back-end služieb. V prípade vyvinutého nástroja sa jedná o rozhranie hlavnej vyhľadávacej služby Pelikan search. Táto služba prijíma pomerne komplikovanú stromovú štruktúru povinných vstupov, ktoré sú získavané z vyhľadávacieho formuláru. Jedná sa o vstupy pre určenie koncových bodov letového spojenia, ďalej o vstupy pre dátum odchodu a dátum návratu a vstupom pre určenie počtu a druhu pasažierov. Celý formulár sa skladá z piatich vstupov plus tlačítka na odoslanie, pri čom je zložený z komponentov zoradených v

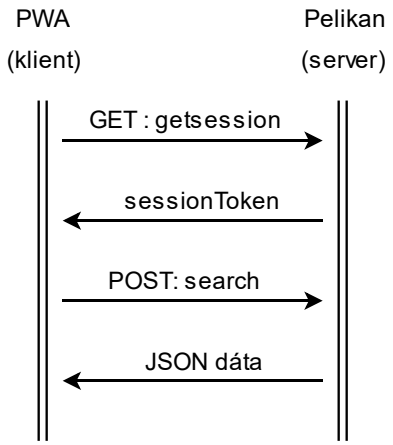
Tabuľka 7. Vstupy pre definíciu koncových bodov letového spojenia boli implementované pomocou API rozhrania Pelikan autocomplete2, čím poskytujú funkcionalitu automatickej nápovedy pri hľadaní letového spojenia. Pre úspešnú implementáciu validácie formuláru bolo použitá knižnica React-final-form. Konkrétne boli importované komponenty `Form` a `Field`, ktoré riešia spomínaný observer vzor. Komponent `Form` drží celkový stav formuláru a komponent `Field` implementuje komponenty, z ktorých je celý formulár zložený, a ktoré sú aktualizované podľa stavu komponentu `Form`. Začiatočný stav komponentu `Form` je definovaný objektom `initialValues`. Validácia formuláru je implementovaná tak, že neumožňuje odoslanie formuláru, pokiaľ nie sú definované koncové body letového spojenia. Ďalej pomocou pomocných funkcií `submitSuceeded` a `dirtySinceLastSubmit` neumožňuje nové odoslanie formuláru pokiaľ síce došlo k úspešnému odoslaniu formuláru, avšak v rámci vstupu formuláru nedošlo k žiadnej zmene.

Zobrazenie letovej ponuky a doplňujúcich informácií o destinácii bolo taktiež implementované s použitím knižníc Rebass, Styled-components a Styled-system. Výsledkom je vytvorený carausel komponent `Results`, ktorý slúži na zobrazenie letových ponúk (*zobrazenie 10 cenovo najvýhodnejších ponúk*). Tento komponent je vytvorený z viacerých elementov (*zobrazenie informácii o ceste do destinácie a o spiatočnej ceste*), ktoré dokážu dynamicky meniť svoju veľkosť a rozloženie, v závislosti na veľkosti zobrazovacej plochy. Tlačítko rezervuj, slúži ako link do rezervačného systému nove.letenky.sk pre konkrétnu letovú ponuku. V rovnakom duchu bol implementovaný aj komponent `Info`. Tento komponent využíva službu Numbeo pre získavanie ďalších informácii ohľadne cieľovej destinácii. Indexy o rôznych kritériách pre konkrétnu destináciu sú vyhodnocované podľa dokumentácie služby Numbeo. Pre prehľadnejšie zobrazenie a zachovanie responzívnosti UI sú indexy kritérií zobrazované vo forme emotikonov (*z dôvodu konštantnej veľkosti*).

5.3.4 Vyhľadávanie

Hlavná funkcia nástroja je vyhľadávanie letových spojení. Táto funkcia je implementovaná v komponente `SearchForm`, ktorý ako hlavnú službu využíva Pelikan search, pomocou rozhrania Pelikan search API. Toto rozhranie prijíma stromovú štruktúru dát tvorenú niekoľkými objektami. Týmto objektom sú priradené hodnoty na základe vstupov z vyhľadávacieho formulára. Objekty `there` a `back` obsahujú informácie o dátume, kedy sa let koná a ďalej obsahujú objekty `to` a `from`, ktoré obsahujú informácie o odletových

a príletových letiskách, prípadne mestách. Okrem toho rozhranie Pelikan search API vyžaduje tzv. unikátny session token. Tento sa získava povinne pri každom dotazovaní na službu Pelikan search, a to zavolaním služby `getsession`. Po zavolaní `getsession` server Pelikan vráti token a následne je sprístupnené využívanie služby Pelikan search cez jej príslušné API rozhranie. Celý priebeh komunikácie so serverom pri vyhľadávaní je zobrazený na Obrázku 28 uvedenom nižšie.



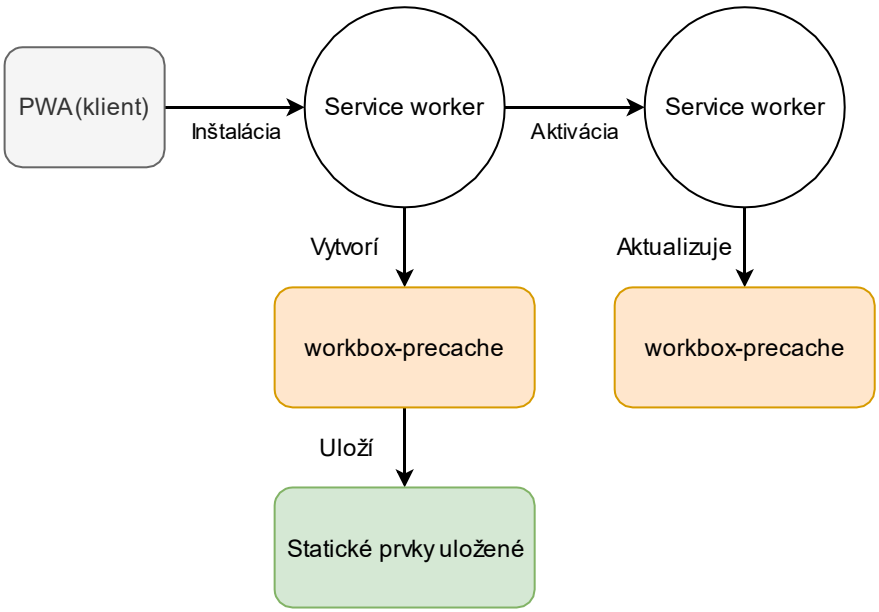
Obrázok 28: Komunikácia so serverom Pelikan

5.3.5 Podpora offline UX

Schopnosť určitej prevádzky aj pri obmedzenom, prípade žiadnom pripojení na internet výrazne zlepšuje UX. Donedávna touto črtou disponovali iba natívne (*mobilné*) aplikácie. S príchodom PWA a s nimi spojenými rozhraniami Cache API a IndexedDB API je možné emulovať natívne správanie aplikácii aj pri aplikáciách webových. Podpora offline teda znamená zabezpečenie takého UX, v rámci ktorého je používateľ okamžite upozornený, že nedisponuje internetovým pripojením, a teda funkcionalita aplikácie je obmedzená.

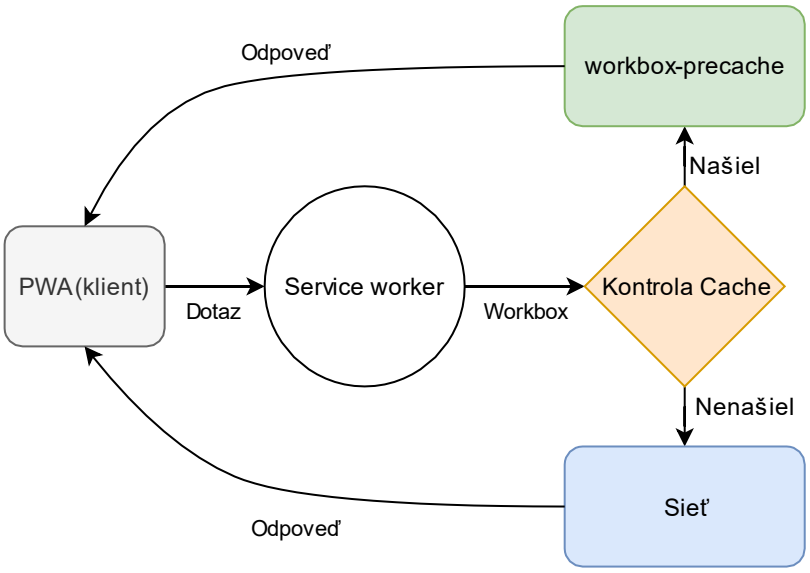
V praxi sa podpora offline UX realizuje vhodnou implementáciou skriptu service worker a rozhrania Cache API, respektíve IndexedDB API. V rámci tzv. best practices (*najlepšie zaužívané postupy pri vývoji*) platí, že všetky statické prvky a skripty aplikácie sa ukladajú pomocou rozhrania Cache API a všetky dáta vytvorené používateľom sa ukladajú a organizujú pomocou rozhrania IndexedDB API. Podpora offline UX bola konkrétne realizovaná implementáciou tzv. offline/cache first prípadu použitia skriptu service worker a rozhrania Cache API. V tomto prípade sú všetky prvky aplikácie (*HTML, CSS, JS, obrázky a podobne*) ukladané do lokálneho úložiska (*rozhranie Cache API*) v momente inštalácie skriptu service worker. K aktivácii skriptu service worker dochádza až po úspešnom uložení

všetkých zdrojov aplikácie.). Pri implementácii stratégie cache first bola použitá knižnica Workbox, ktorá je udržiavaná spoločnosťou Google a slúži na správnu implementáciu využívania úložiska Cache skriptom service worker. Táto stratégia vytvára úložisko workobox-precache typu cache na strane klienta, ktoré slúži na uloženie všetkých statických prvkov aplikácie. Zároveň knižnica Workbox zabezpečuje správu úložiska (aktualizácia jeho obsahu). Pri úvodnej inštalácii verzie skriptu service worker sa vytvorí úložisko workbox-precache so statickými prvkami a následne slúži ako hlavný zdroj pre statické prvky aplikácie. Vytvorenie a aktualizácia úložiska je znázornená na Obrázku 28.



Obrázok 29: Vytvorenie a aktualizácia úložiska cache

Tento typ implementácie je vlastne obdobou prvotnej inštalácie natívnej aplikácie na mobilných zariadeniach, nakoľko bez uloženia všetkých základných prvkov by aplikácia nemohla fungovať. Ďalšou výhodou implementácia Cache API v štýle tzv. „cache first“ prístupu (*využitie Cache API na ukladanie statických prvkov pri inštalácii skriptu service worker*) je, že skracuje čas potrebný na načítanie aplikácie pri každom ďalšom otvorení (*zo siete sa získavajú už len potrebné dáta*. Akonáhle je service worker aktivovaný, všetky prichádzajúce dotazy aplikácie sa najprv porovnávajú s úložiskom workbox-precache. V prípade, že sa požadované dáta nachádzajú v úložisku sú poskytované okamžite aplikácii. Ak Workbox nenájde odpoveď na dotaz v úložisku Cache, dotaz je poslaný ďalej sieťou na server a odpoveď prichádza zo siete. Tento mechanizmus je zobrazený na Obrázku 29.



Obrázok 30: Odbavovanie dotazov pomocou cache

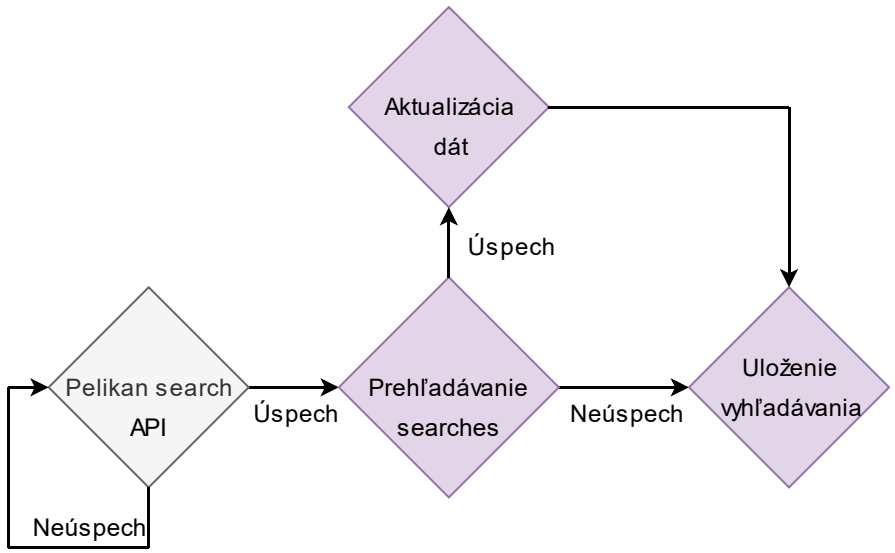
5.3.6 Notifikácie

Aspekt notifikácií bol realizovaný pomocou rozhrania IndexedDB API a knižnice React. Konkrétne sa realizovalo notifikovanie používateľa pri vzniku lacnejšieho letového spojenia. Notifikácie sú realizované komponentom `Notiffication`. Definícia databázy `FlightSearchDB` je realizovaná v komponente `db`. Databáza obsahuje jednu tabuľku `searches` s atribútmi, ktoré popisujú výsledky vyhľadávania letových spojení. Schéma databázy sa nachádza na Obrázku 30. Nakoľko sa práca zameriavala najmä na klientskú časť aplikácie, aj notifikovanie o vzniku lacnejšieho letového spojenia bolo realizované na strane klienta. Na ukladanie informácií o používateľových vyhľadávaniach sa využíva rozhranie IndexedDB API. K rozhraniu sa pristupuje pomocou knižnice Dexie. Táto knižnica uľahčuje vytváranie dotazov v rámci rozhrania IndexedDB API. Vďaka tomuto rozhraniu aplikácia ukladá a uchováva používateľove vyhľadávania na strane klienta.

searches
id Integer NN (PK)
price Float
takeOff Text
land Text
startDate Datetime
endDate Datetime
values Blob

Obrázok 31:Tabuľka udržujúca dáta o vyhľadávaní

Pri získaní dát z vyhľadávania cez rozhranie Pelikan search API v komponente `SearchForm` sa zároveň prehľadávajú záznamy v tabuľke `searches`. Prehľadávanie vyhľadá záznamy o vyhľadávanom letovom spojení a zmaže všetky záznamy s rovnakou alebo vyššou cenou, čím aktualizuje dáta o vyhľadávaných spojeniach v tabuľke `searches`. Následne dôjde k uloženiu záznamu z vyhľadávania (*Pelikan search API*) do tabuľky `searches`. Mechanizmus ukladania výsledkov vyhľadávania letových spojení je bližšie popísaný na Obrázku 31 uvedenom nižšie.



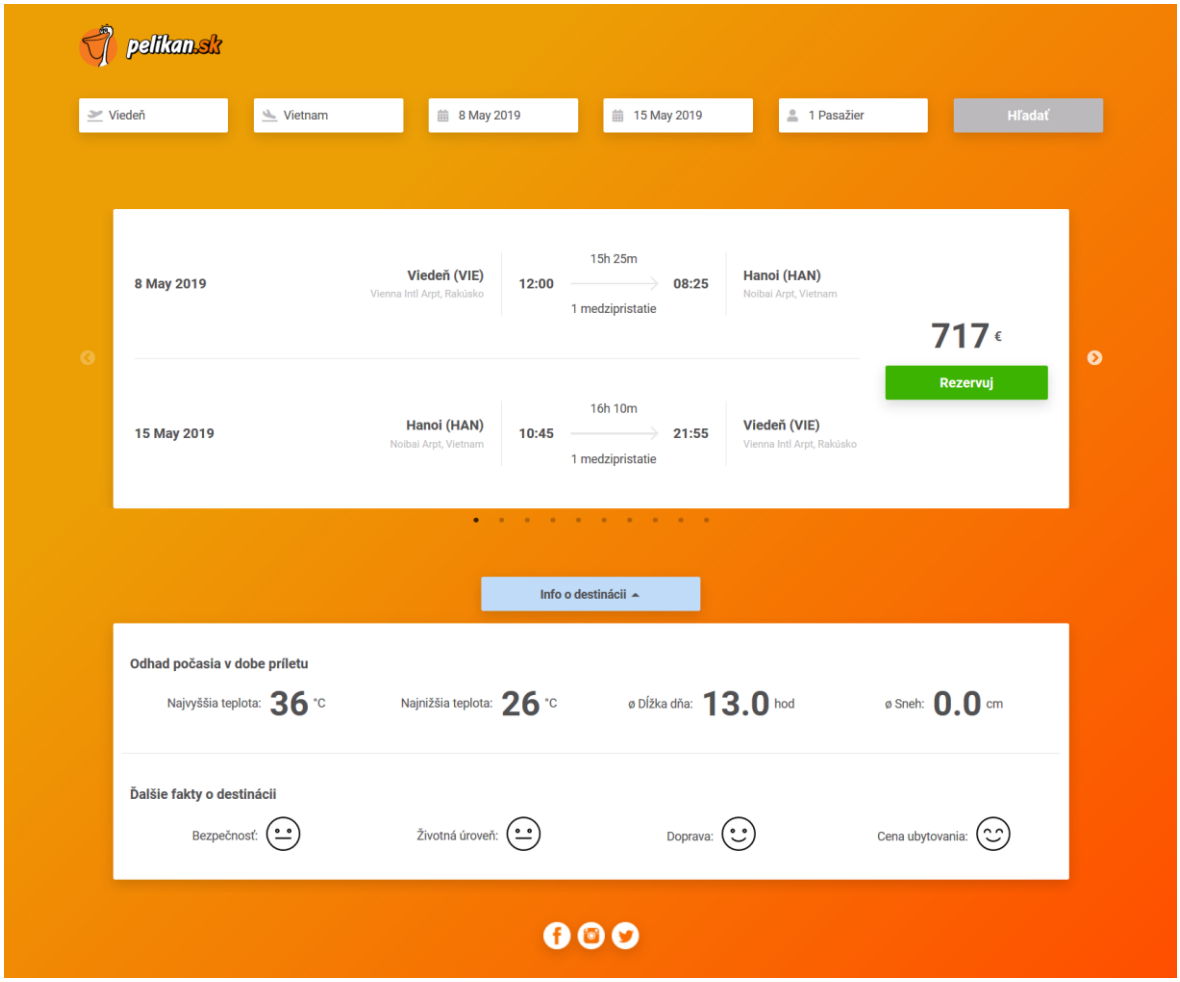
Obrázok 32: Mechanizmus ukladania výsledkov vyhľadávania letových spojení

Následne pri ďalšom otvorení aplikácie dôjde na pozadí aplikácie k opätovnému vyhľadávaniu uložených ponúk. To sa už deje v komponente `Notiffication`. V rámci tohto komponentu sú priebežne vykonávané nové vyhľadávania. V prípade, že medzičasom došlo k vzniku lacnejšieho letového spojenia je používateľ o tomto novom letovom spojení notifikovaný, a po kliknutí na notifikáciu sa používateľovi zobrazí nová ponuka.

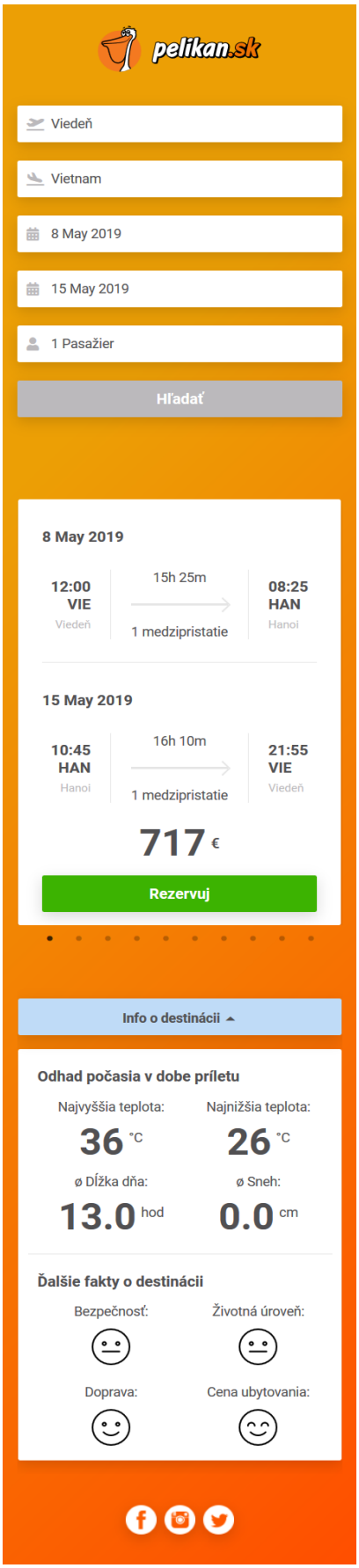
5.3.7 Vybrané obrazovky aplikácie

Aplikácia bola implementovaná za použitia moderných technológií v rámci vývoja klientských aplikácii. Pri vývoji sa kládol dôraz najmä na dodržanie štandardu progresívnej webovej aplikácie a s ním spojenej responzívnosti výsledného nástroja. Obidve tieto kľúčové aspekty vyhľadávacieho nástroja boli úspešne implementované. Na Obrázku 28 uvedenom nižšie je fotka (*kompletná obrazovka*) desktop zobrazenia najlacnejšej letovej ponuky z Viedne do Vietnamu. Pre ilustráciu dosiahnutej responzívnosti UI výsledného nástroja je na ďalšej strane uvedená fotka mobilného zobrazenia tej istej letovej ponuky. Ako

už bolo spomenuté, UI bolo implementované tak, aby dynamicky reagovalo na interval bodov zlomu (tzv. breakpoints), podľa ktorého sa mení veľkosť a rozloženie elementov UI. Interval bodov zlomu je definovaný ako pole `breakpoints` v rámci objektu `theme`, ktorý využíva komponent `ThemeProvider`. Pole breakpoints obsahuje nasledujúce hodnoty bodov zlomu: `1200, 800, 600`. Hodnoty sú uvedené v pixeloch a explicitne udávajú hodnotu šírky zobrazovacej plochy, pri ktorej dôjde k zmene kompozície UI.



Obrázok 33: Fotka obrazovky desktop PC zobrazenia letovej ponuky



Obrázok 34: Fotka obrazovky mobilného zobrazenia letovej ponuky

6 Testovanie

Existuje množstvo spôsobov ako je možné kvalitne otestovať aplikáciu. Základná funkcionalita (*formulár a zobrazovanie jednotlivých prvkov aplikácie*) bola testovaná priebežne počas vývoja funkčnými, manuálnymi testami. Pri týchto testoch bolo obzvlášť nápomocné práve vývojové prostredie vytvorené technológiou CRA (*Create React App*), ktoré poskytovalo automatickú kompiláciu kódu, a teda bolo možné aplikáciu ladiť okamžite za behu. Pri riešení testovania aplikácie bolo potrebné nájsť aj taký spôsob testovania, ktorý by jednoznačne a vecne zhodnotil, či vytvorená aplikácia spĺňa požiadavky PWA (*Progresívna Webová Aplikácia*). Táto časť aplikácie bola otestovaná pomocou nástroja Google lighthouse, ktorý vyvinula spoločnosť Google.

6.1 Nástroj Google Lighthouse

Google Lighthouse je open-source nástroj poskytujúci automatizované testovanie, vďaka ktorému je možné zlepšovať kvalitu webových stránok (*resp. aplikácií*). Tento nástroj možno efektívne použiť pri akejkoľvek stránke alebo aplikácii. Prístupný je z prehliadača Google Chrome, konkrétne z jeho nástroja tzv. developer tools (*vývojárske prostredie*). Po spustení nástroj poskytuje viaceré druhy auditov, v rámci ktorých poskytuje aj návrhy na zlepšenie konkrétneho aspektu aplikácie. Pre účely testovania vyvinutého vyhľadávacieho nástroja je najdôležitejší práve tzv. Progressive Web App audit, ktorého úlohou je kontrola správnej implementácie štandardu PWA. [23]

6.2 Výsledky testovania

Vykonaný audit vytvorenej aplikácie na vyhľadávanie letových spojení sa zameriava na validáciu PWA aspektov aplikácie. Tento typ auditu sa rozdeľuje do troch kategórií. Medzi tieto kategórie patrí rýchlosť a spoľahlivosť PWA, schopnosť inštalácie PWA a na záver kategória optimalizácie PWA. Výsledok prvej kategórie auditu validácie PWA aspektov je zobrazený na Tabuľke 8 uvedenej nižšie.

Tabuľka 8: Kategória rýchlosti a spoľahlivosti PWA

Popis	Výsledok
Dostatočne rýchle načítanie aplikácie cez mobilnú sieť	Splnené
Aplikácia odpovedá statusom 200 v stave offline	Splnené
Štartovacia URL odpovedá statusom 200	Splnené

Schopnosť aplikácie poskytovať dostatočne rýchle načítanie aj cez mobilné siete zabezpečí dobrý UX pri využívaní aplikácie mobilnými zariadeniami. Audit posudzuje rýchlosť načítania podľa doby uplynutej od zadania URL adresy, až po moment kedy je aplikácia schopná interakcie. Pri tejto časti testovania audit simuluje pomalé 4G sieťové pripojenie. Audit je splnený, ak pri týchto podmienkach je aplikácia schopná interakcie do 10 sekúnd. V prípade vyvinutej vyhľadávacej aplikácie je aplikácia schopná interakcie (*za rovnakých podmienok*) od 5,3 sekúnd. Treba pripomenúť, že sa jedná o dobu pri prvom otvorení aplikácie (*obdoba inštalácie natívnej aplikácie*), pri čom nasledujúce otvorenia sú oveľa rýchlejšie vďaka ukladaniu statických prvkov aplikácie na strane klienta (*rozhranie Cache API*). S týmto súvisí aj ďalšia sledovaná črta tejto kategórie auditu, ktorá bola nástrojom Lighthouse vyhodnotená ako úspešná. Jedná sa o schopnosť prevádzky aplikácie offline, ktorá je zabezpečená práve vďaka súčinnosti skriptu service worker a úložiska Cache.

Tabuľka 9: Kategória aspektu inštalácie PWA

Popis	Výsledok
Využívanie HTTPS	Splnené
Aplikácia registruje skript service worker	Splnené
Aplikácia disponuje súborom web app manifest	Splnené

V Tabuľke 9 uvedenej vyššie je znázornené vyhodnotenie kategórie PWA auditu, ktorá sa zameriava na vyhodnotenie aspektu inštalácie aplikácie do prehliadača (*napodobenie inštalácie natívnej aplikácie do mobilného zariadenia*). Táto kategória auditu sa vyhodnocuje v troch bodoch. Prvým je využívanie HTTPS v rámci prevádzky aplikácie, čo zaručí bezpečnú prevádzku. Ďalším bodom je, že vytvorená aplikácia registruje svoje príslušný skript service worker. Tento skript slúži ako tzv. sieťová proxy, čo znamená že skript obstaráva kompletnú sieťovú komunikáciu medzi aplikáciou a serverom. Posledným bodom tejto kategórie je súbor web app manifest. Tento súbor umožňuje inštaláciu PWA aplikácie do prehliadača, a pri mobilných zariadeniach umožňuje pridať odkaz na plochu zariadenia. Táto črta emuluje UX natívnych aplikácií.

Poslednou kategóriou auditu zameraného na validáciu PWA aspektov vyvinutej aplikácie je kategória PWA optimalizácie. V rámci tejto kategórie audit sleduje v konkrétne šiestich bodoch, celkovú optimalizáciu PWA aspektov aplikácie. Výsledok tejto kategórie auditu je uvedený v Tabuľke 10 uvedenej na nasledujúcej strane.

Tabuľka 10: Kategória optimalizácie PWA

Popis	Výsledok
Presmerovanie HTTP prevádzky na HTTPS	Nesplnené
Aplikácia disponuje vlastnou tzv. splash obrazovkou	Splnené
Aplikácia ovláda farbu témy prehliadača	Splnené
Zabezpečenie responzívnosti aplikácie	Splnené
Disponuje meta tagom definujúcim viewport	Splnené
Disponuje obsahom aj pri nepovolení jazyka JavaScript	Splnené
Poskytuje tzv. apple-touch-icon	Splnené

Prvý bod tejto kategórie nebol splnený. Je to z dôvodu toho, že audity boli vykonané za behu na lokálnom hostingu. Tejto požiadavke je možné vyhovieť vhodnou HTTPS konfiguráciou adresy domény, na ktorú sa celá vyhľadávacia aplikácia nasadí. Ďalším bodom je, že aplikácia disponuje vlastnou splash obrazovkou (obrazovka zobrazujúca sa pri spustení aplikácie z pracovnej plochy zariadenia). Tento prvok rovnako emuluje UX natívnych aplikácií (*namiesto bielej plochy konfrontuje používateľa okno naznačujúce načítavanie*). S tým súvisí aj nasledujúci bod, ktorý vyhodnocuje, či aplikácia upravuje vzhľad (*farby*) rozhrania prehliadača, podľa svojich príslušných dizajnových aspektov. Za splnený bol vyhodnotený aj bod zabezpečenia responzívnosti aplikácie. Znamená to, že obsah aplikácie je nepretržite vhodne rozmiestnený v závislosti na viewport (*zobrazovacia plochy*) zariadenia. Pokiaľ sa šírka obsahu aplikácie nezhoduje so šírkou poskytovanou zobrazovacou plochou zariadenia, znamená to, že aplikácia nie je optimalizovaná na zobrazenie v mobilných zariadeniach. Teda nie je responzívna. Ďalším bodom je, že aplikácia disponuje meta tagom definujúcim šírku viewportu aplikácie na šírku viewportu príslušného zariadenia. V rámci tohto meta tagu sa definuje aj škála priblíženia pri zobrazení aplikácie. Opäť aj tento bod v rámci auditu bol splnený. Predposledným bodom auditu je bod, ktorý kontroluje či aplikácia disponuje určitým obsahom, ktorý je zobrazený používateľovi v prípade zablokovania technológie JavaScript v jeho prehliadači. Tento bod auditu bol splnený, nakoľko je používateľ informovaný o nutnosti povolenia technológie JavaScript za účelom prevádzky aplikácie. Posledný bod auditu sa týka ikon pre vytváranie odkazov aplikácie na pracovných plochách zariadení značky Apple. Tento bod auditu vlastne zisťuje, či je aplikácia dostupná na všetkých typoch HW platforiem. Rovnako ako u ostatných aj tento bod bol nástrojom Google Lighthouse vyhodnotený za splnený.

Záver

Cieľom tejto práce bolo analyzovať a porovnať aktuálne dostupné platformy a technológie využívane pri vývoji mobilných a webových aplikácií, ktoré by boli vhodné pre vývoj nástroja na jednoduché vyhľadávanie letových spojení. Výsledný nástroj je schopný notifikovať používateľa o zmenách a rovnako je schopný offline prevádzky. Medzi jeho kľúčové parametre patrí dostupnosť na rôznych hardwarových platformách, rýchlosť a schopnosť prevádzky pri obmedzenom, prípadne žiadnom pripojení na internet.

Prvá kapitola sa venuje rozboru vhodných platforiem pre vyvíjaný nástroj. Kapitola pojednáva o výhodách, a rovnako aj o nevýhodách jednotlivých platforiem.

Druhá kapitola sa bližšie zameriava na rozbor platformy progresívnej webovej aplikácie. Ďalej rozoberá výhody a nevýhody voči ostatným typom platforiem. Nasleduje popis architektúry a priblíženie základných aspektov nevyhnutných pre realizáciu progresívnej webovej aplikácie. Dôležitou časťou tejto kapitoly je vysvetlenie mechanizmov ukladania na strane klienta v súčinnosti so skriptom service worker, ktorý slúži ako sieťová proxy.

Kapitola tri ďalej rozoberá analýzu popredných technológií, ktoré sa využívajú pri vývoji klientských aplikácií. Kapitola poskytuje prehľadné porovnanie jednotlivých technológií a v závere zdôvodňuje voľbu technológie a približuje ďalšie podporné knižnice, ktoré boli použité pri vývoji.

Nasleduje praktická časť, ktorá je rozdelená na tri časti. Prvú časť tvorí kapitola štyri, ktorá sa venuje analýze všeobecných požiadaviek na výsledný vyhľadávací nástroj. V tejto časti je vo forme analýzy prípadu použitia definovaná očakávaná funkcionalita vyhľadávacieho nástroja. Druhá časť je zahrnutá v kapitole päť. Táto kapitola sa venuje kompletnému návrhu používateľského rozhrania vo forme wireframe návrhov, kde sú vyobrazené všetky požiadavky na používateľské rozhranie aplikácie, v rátane jeho repsonzívnosti. Ďalej kapitola päť bližšie rozoberá základné koncepty v rámci implementácie celej klientskej časti aplikácie a jej kľúčových prvkov (*podpora offline, notifikácie*) a v závere uvádza demonštračné fotky obrazoviek vyvinutého rozhrania. Posledná, šiesta kapitola, sa venuje testovaniu vyvinutého nástroja a zhodnoteniu jeho progresívnych webových aspektov. V rámci zadania sa podarilo realizovať všetky požadované črty na vyhľadávací nástroj.

Literatúra

[1] Amos Ndegwa. *What is a Web Application?* [online]. Maxcdn , 2016 [cit. 2018-12-15]. Dostupné z URL: <https://www.maxcdn.com/one/visual-glossary/web-application/>

[2] Nihinlola Fajemilehin. *Introduction To Mobile Applications* [online]. Medium, 2018 [cit. 2018-12-15]. Dostupné z URL: <https://codeburst.io/introduction-to-mobile-applications-ddf9732b690a>

[3] Pete LePage. *Your First Progressive Web App* [online]. Google, 2019 [cit. 2018-12-17]. Dostupné z URL: <https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/>

[4] Matt Gaunt. *Service Workers: an Introduction* [online]. Google, 2019 [cit. 2018-12-20]. Dostupné z URL: <https://developers.google.com/web/fundamentals/primers/service-workers/>

[5] Jake Archibald. *Is service worker ready ?* [online]. Github, 2019 [cit. 2019-1-1]. Dostupné z URL: <https://jakearchibald.github.io/isserviceworkerready/>

[6] Youenn Fablet. *Workers at your service* [online]. Webkti, 2018 [cit. 2019-1-1]. Dostupné z URL: <https://webkit.org/blog/8090/workers-at-your-service/>

[7] Addy Osmani, Marc Cohen. *Offline Storage for Progressive Web Apps* [online]. Google, 2019 [cit. 2019-1-1]. Dostupné z URL: <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa>

[8] MDN web docs. *Cache* [online]. Mozzila, 2019 [cit. 2019-1-1]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/Cache>

[9] MDN web docs. *Cache* [online]. Mozzila, 2019 [cit. 2019-1-1]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API>

[10] Matt Gaunt, Paul Kinlan. *The Web App Manifest* [online]. Google, 2019 [cit. 2019-1-1]. Dostupné z URL: <https://developers.google.com/web/fundamentals/web-app-manifest/>

[11] Angular dokumentácia. *Architecture overview* [online]. Google, 2019 [cit. 2019-1-5]. Dostupné z URL: <https://angular.io/guide/architecture>

[12] Vue dokumentácia. O*verview* [online]. Vue, 2018 [cit. 2019-1-5]. Dostupné z URL: <https://v1.vuejs.org/guide/overview.html>

[13] React dokumentácia. *Architecture overview* [online]. Facebook, 2019 [cit. 2019-1-5]. Dostupné z URL: <https://reactjs.org/>

[14] Styled-components dokumentácia. *Documentation* [online]. Styled-components, 2019 [cit. 2019-3-5]. Dostupné z URL: <https://www.styled-components.com/docs>

[15] Styled-system dokumentácia. *Getting started* [online]. Styled-system, 2019 [cit. 2019-3-5]<https://styled-system.com/getting-started>

[16] Rebass dokumentácia. *Getting started* [online]. Rebass, 2019 [cit. 2019-3-5]. Dostupné z URL: <https://rebassjs.org/getting-started>

[17] MDN web docs. *XMLHttpRequest* [online]. Mozzila, 2019 [cit. 2019-4-3]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

[18] Axios github page. *Axios* [online]. Github, 2019 [cit. 2019-4-3]. Dostupné z URL: <https://github.com/axios/axios>

[19] React-final-form github page. *React Final Form* [online]. Github, 2019 [cit. 2019-4-3]. Dostupné z URL: <https://github.com/final-form/react-final-form#readme>

[20] Developer Portal. *Introduction* [online]. World weather online, 2015 [cit. 2019-2-5]. Dostupné z URL: < https://www.worldweatheronline.com/developer/api/docs/>

[21] Numbeo administrator. *About Numbeo.com* [online]. Numbeo, 2019 [cit. 2019-1-5]. Dostupné z URL: < https://www.numbeo.com/common/about.jsp>

[22] React dokumentácia. *Create a New React App* [online]. Facebook, 2019 [cit. 2019-4-3]. Dostupné z URL: <https://reactjs.org/docs/create-a-new-react-app.html>

[23] Tools for Web Developers. *Lighthouse* [online]. Google, 2019 [cit. 2019-4-10]. Dostupné z URL: <https://developers.google.com/web/tools/lighthouse/>

Zoznam použitých skratiek

WWW	World Wide Web
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
CSS	Cascading Style Sheet
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
XML	Extensible Markup Language
JSON	JavaScript Object Notation
PWA	Progressive Web App
JS	JavaScript
SPA	Single Page Application
API	Application Program Interface
XHR	XML HTTP Request
UI	User Interface
UX	User Experience
Props	Properties
LRU	Least Recently Used

Zoznam príloh

A Obsah priloženého CD

- PDF súbor s textom diplomovej práce
- adresár PelikanPWA obsahujúci zdrojové súbory aplikácie a adresár build so skompilovanou verziu aplikácie vhodnou na nasadenie na server. Pri vývoji bola použitá knižnica React verzie 16.8.6 s ďalšími podpornými knižnicami (vypísané v súbore `package.json`)