FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Progresivní webová aplikace DrinkApp
Student:	Tomáš Dvořák
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Téma progresivních webových aplikací je velmi rozsáhlé a novátorské. Cílem této práce je implementovat takovou aplikaci, která poskytuje uživatelům katalog nápojů, které je možné vyhledávat a doporučovat pomocí kolaborativního filtrování.

Při realizaci postupujte podle těchto kroků:

1. Analyzujte možnosti progresivních webových aplikací
2. Na základě analýzy proveďte vhodný softwarový návrh.
3. Implementujte za použití vhodných technologií první verzi aplikace.
4. Implementujte libovolný algoritmus pro kolaborativní filtrování.
5. Funkční aplikaci nasaďte a zajistěte možnost použití, alespoň uzavřené skupině uživatelů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Progresivní webová aplikace DrinkApp

Tomáš Dvořák

Katedra softwarového inženýrství
Vedoucí práce: Ing. Filip Glazar

3. června 2020

Poděkování

Chtěl bych poděkovat panu Ing. Filipu Glazarovi za vedení mé bakalářské práce. Dále poděkování patří mé rodině a nejbližším přátelům za podporu během celého studia.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 3. června 2020 ……

Odkaz na tuto práci

Dvořák, Tomáš. *Progresivní webová aplikace DrinkApp*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW: ⟨`https://yourcocktails.club`⟩.

Abstrakt

Tato bakalářská práce se zabývá vytvořením progresivní webové aplikace, která svým uživatelům doporučuje koktejly pomocí aktivního kolaborativního filtrování. V práci jsou využívány nejnovější technologie a přístupy určené pro tvorbu webových aplikací. Práce rovněž popisuje vývoj celého konceptu PWA, jeho výhod, nevýhod a porovnává jej s nativními aplikacemi. Součástí práce je i návrh a implementace serverové části. Práce mimo jiné popisuje nasazení celého projektu do produkčního prostředí.

Klíčová slova progresivní webové aplikace, kolaborativní filtrování, pwa, react, redux, service worker, web app manifest

Abstract

This bachelor thesis deals with the building process of a progressive web application that recommends its users different cocktails based on the active collaborative filtering method. The process itself uses the newest technologies and approaches for building web applications. The thesis also describes the development of the PWA concept, its advantages and disadvantages and compares it to native applications. It also contains a proposal and implementation of the server part and describes the deployment of the entire project to a production environment.

Keywords progressive web apps, collaborative filtering, pwa, react, redux, service worker, web app manifest

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Web již dávno neslouží pouze k zobrazování statického obsahu, webové aplikace umožňují svým uživatelům plnit velké množství různorodých úkolů, od napsání a odeslání e-mailu, až po editaci fotografií. Ačkoli je většina webových aplikací responzivní, stále se nepoužívají tak dobře jako ty nativní. Proto vznikly progresivní webové aplikace, které se zaměřují na lepší uživatelský zážitek, který odstraní pomyslný most mezi webovými a nativními aplikacemi.

Cílem této bakalářské práce je návrh a implementace progresivní webové aplikace zaměřené na doporučování koktejlů, alkoholických i nealkoholických, svým uživatelům pomocí kolaborativního filtrování. Tato webová aplikace se od ostatních webových aplikací bude lišit schopností aktivně doporučovat koktejly na základě předchozích uživatelských interakcích, dále funkčností bez připojení k internetu a v neposlední řadě multiplatformní podporou.

První a druhá kapitola práce (Rešerše a Progresivní webová aplikace) je věnována seznámení se s klíčovými technologiemi a principy, které jsou v práci využity. Dále je proveden softwarový návrh, jež je následován implementací s detailním popisem aplikační struktury klientské a serverové části. Implementace je následně podrobena uživatelskému testování. Práce je završena nasazením aplikace a jejím spuštěním pro širší veřejnost.

Kapitola 1

Rešerše

Následující kapitola tvoří teoretický souhrn technologií a principů, které byly použity při realizaci této bakalářské práce. Kapitola popisuje technologie použité jak pro klientskou částí (frontend), tak pro serverovou část (backend).

1.1 Frontend

Jedná se o pojem spadající do webového inženýrství, jedná se o aplikační vrstvu, se kterou uživatel pracuje (klientská část). Frontend se skládá ze dvou hlavních částí: [1]

- Design – vytvoření grafického návrhu (typicky od grafického návrháře), kterou následně programátor implementuje,
- Implementace – proces, při kterém programátor na základě grafického návrhu (designu) vytváří funkční model webové stránky/aplikace, který reflektuje grafický návrh.

Tvorba webové stránky/aplikace je realizována pomocí následujících technologií: [1]

- *HTML/XHTML* – značkovací jazyky, pomocí kterých je definována struktura webové stránky či aplikace,
- *CSS* – jazyk pro popis způsobu zobrazení elementů na stránce,
- *JavaScript* – multiplatformní skriptovací jazyk, který vnáší do webové stránky/aplikace aplikační vrstvu.

Při tvorbě moderních webových aplikací tvoří JavaScript stěžejní bod, protože celá aplikační vrstva se přesouvá na stranu klienta, přičemž serverová část slouží pouze jako Rest API.

1.1.1 React

React (také známy jako React.js nebo ReactJS) je JavaScriptová knihovna pro tvorbu uživatelského rozhraní. Hlavním vývojářem knihovny je Facebook, přičemž do vývoje je zapojena i komunita. [2]

Hlavním pilířem aplikací založených na Reactu jsou znovupoužitelné komponenty, které mají vlastní logiku, stav a vykreslovací funkci (anglicky *render function*). Komponentu ve vykreslovací funkci vrátí buď další komponentu nebo HTML element. Aplikace se pak skládá ze stromu komponent, které si mezi sebou jednosměrně předávají data (známé jako *one-way binding*). [2]

1.1.1.1 Historie

V květnu roku 2013 byl React představen na javascriptové konferenci *JS ConfUS*, kde ale nesklidil příliš velký obdiv. Většina posluchačů vůči němu byla skeptická. Později, koncem roku 2013, David Nolen vyzdvihnul technologické přednosti Reactu v článku „The Future of JavaScript MVC Frameworks“, díky kterému se React v komunitě uchytil. [3]

Dle průzkumu za rok 2019, kterého se účastnilo 63 585 respondentů, se React umístil v kategorii *Webové frameworky* na druhém místě, hned za knihovnou *jQuery*. [4]

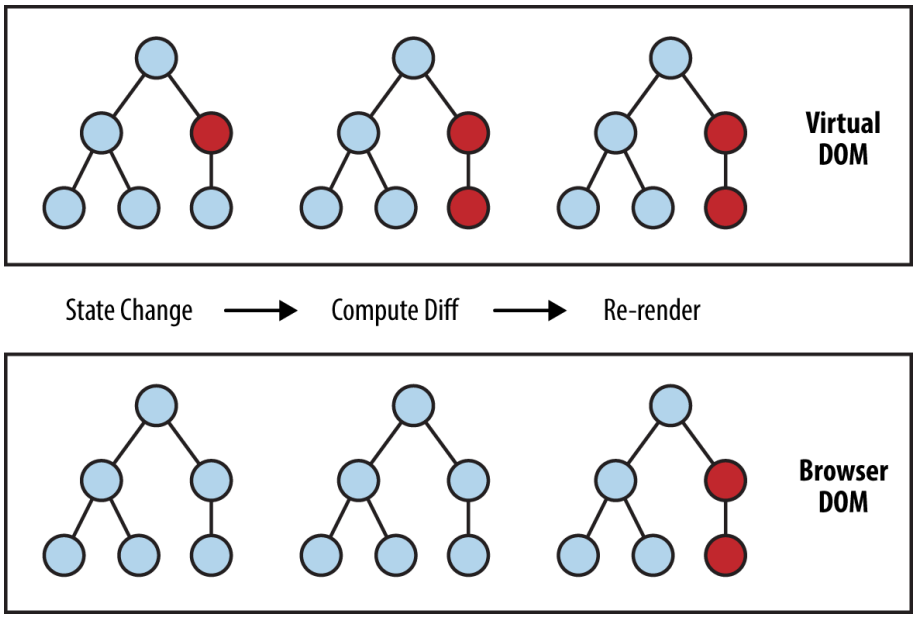
1.1.1.2 Virtuální DOM

React je postaven na konceptu virtuálního DOMu, jež je známý jako VDOM. Myšlenka spočívá ve vytvoření kopie aktuální DOMu, která je reprezentována jako *n-ární* strom, stejně jako DOM. Při změně vnitřního stavu aplikace se provede porovnání mezi virtuální a skutečnou DOM strukturou, pokud jsou odlišné, provedou se příslušné úpravy na DOM struktuře. Proces synchronizace je známý jako *Diffing Algorithm*. React nejdříve porovnává kořeny obou stromů a v případě rozdílných typů elementů (např. *div* vs *span*) provede kompletní vytvoření nového stromu. V takovém případě je každý potomek (komponenta) svou rodičovskou komponentou informován o tom, že bude odstraněna. V případě stejného typu dojde k porovnání jeho atributů. Pokud má element další potomky, rekurzivně se nad nimi zavolá stejný proces.

Shrnutí procesu aktualizace:

1. Dojde k úpravě VDOMu,
2. Dojde k porovnání VDOMu a DOMu (použit je *Diffing Algoritm*),
3. U objektů uvnitř VDOMu, u kterých došlo ke změně, zpropagují svoje změny na příslušný DOM element,

4. Prohlížeč zareaguje na změny které se staly v DOMu a provede jejich překreslení.



Obrázek 1.1: Propagace změny stavu z VDOM na DOM [5]

Díky konceptu VDOM dochází k zrychlení celé aplikace. Především protože je minimalizován počet modifikací v DOM struktuře. Dalším důvodem je fakt, že k úpravě dojde teprve, když je známa konečná podoba DOMu. Prohlížeč tudíž nemusí při každé změně zbytečně překreslovat afektované elementy.

1.1.2 Redux

Redux je JavaScriptová knihovna s otevřeným zdrojovým kódem (anglicky *open source*), která do webových aplikací přidává centralizovaný aplikační stav. Redux byl založen v roce 2015 Danem Albramovem. Knihovna Redux staví na základních třech principech: jediný zdroj pravdy, stav je pouze ke čtení a změny jsou vytvářeny „čistými“ funkcemi. [6]

Jediný zdroj pravdy

Kompletní aplikační stav je uložen v objektu zvaném *store*. Toto rozhodnutí vede ke snažšímu předávání informací mezi různými částmi aplikace.

```
 // vypsání aktuálního stavu aplikace do konzole
console.log(store.getState())

/* Obsah aktuálního stavu
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
*/
```

Výpis kódu 1.1: Redux – výpis aplikačního stavu

Stav je pouze ke čtení

Jediná možnost, jak změnit stav, je emitovat akci. V implementačním slova smyslu je nutné zavolat funkci, která vrací objekt, jež popisuje, co se v aplikaci událo.

```
// Funkce pro vytváření specifické akce
// (Tyto funkce známé jako Action Creators)
function completeTask(id) {
    return {
        type: 'COMPLETE_TODO', // název akce
        id: id, // nastavení parametru
    }
}

// vytvoření akce pomocí funkce completeTask
const action = completeTask(1);

// emitace akce
store.dispatch(action);
```

Výpis kódu 1.2: Redux – ukázka vytvoření akce a její následné emitace

Změny jsou vytvářeny „čistými“ funkcemi

Při emitování akce pomocí metody *dispatch* na objektu *store* (zmíněným v prvním bodě) následuje volání zpracující funkce. *Reducer* je funkce, která neobsahuje vedlejší efekty. Tato funkce přijímá dva argumenty:

- **state** – aktuální stav aplikace,
- **action** – akci, kterou jsme emitovali v předchozím bodě.

Daná funkce následovovně ověří, zda umí danou akci zpracovat. Pokud ano, vrátí novou kopii stavu s provedením patřičných změn. Pokud ne, vrátí nezměněný stav. Viz ukázka 1.3

```
// Reducer zodpovídá za úprava globálního stavu
function todos(state = [], action) {
  switch (action.type) {
    case 'COMPLETE_TODO':
      /*
         Vytvoření nového stavu z původní stavu,
         který při kopírování upraví prvek jehož identifikátor
         odpovídá identifikátoru v přijaté akci
      */
      return state.map(todo => {
        if (todo.id === action.id) {
          return Object.assign({}, todo, {
            completed: true,
          })
        }
        return todo;
      })
    default:
      return state; // vrať původní stav
  }
}

// import funkcí z knihovny redux
import { combineReducers, createStore } from 'redux'

// vytvoření hlavního Reduceru, který se skládá z dílčích Reducerů
const reducer = combineReducers({ todos })

// vytvoření globální stavu
const store = createStore(reducer)
```

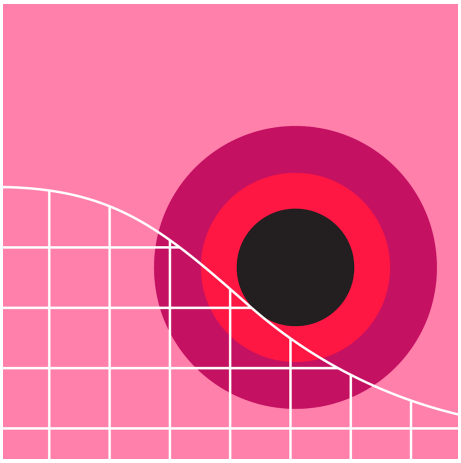
Výpis kódu 1.3: Redux – vytvoření nového aplikačního stavu

1.1.3 Material Design

Material Design je vizuální jazyk, který syntetizuje klasické principy dobrého designu s inovacemi technologie a vědy. Material design zachovává stejné grafické rozhraní napříč různými platformami jako Android, iOS a web. Nedílnou součástí Material designu je také zaručení správné přístupnosti. [7]

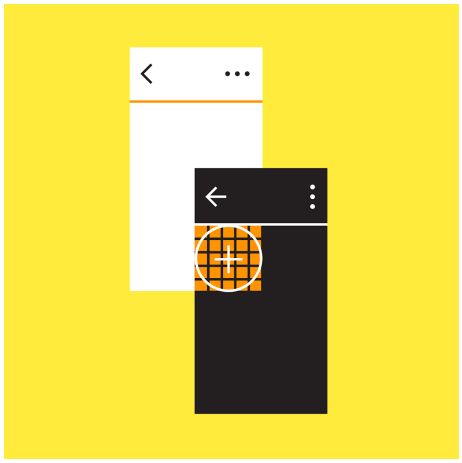
Poprvé byl Material Design představen v roce 2014 na konferenci *Google IO*. Součástí Material Designu je velké množství znovu použitelných komponent. [8] Konkrétní implementace komponent je k dispozici s otevřeným zdrojovým kódem na webovém uložišti github.com. Material Design dále definuje pravidla, jak lze upravovat tyto předpřipravené komponenty tak, aby nedošlo k porušení jeho základních principů.

1.2.1: Typografie



1.2.2: Pohyb udává smysl

1.2.3: Rozšiřitelnost



1.2.4: Multiplatformní UI

Obrázek 1.2: Základní principy Material Designu [7]

1.2 Backend

Jedná se o vrstvu aplikace, která není přístupná uživateli a je zodpovědná za zpracování dat [1].

Dnešní moderní dynamické aplikace stavějí na konceptu vyměňování dat mezi klientskou částí (frontend) a serverovou částí (backend) pomocí HTTP požadavků. Typicky se využívá architektura Rest API.

Rest API

REST *(REpresentational State Transfer)* je architektura rozhraní, navržena pro distribuované prostředí. Rozhraní je orientováno datově, přičemž každý datový zdroj je určený jednoznačnou URI. Tato URI se pak nazývá *endpoint*.

REST implementuje čtyři základní metody pro práci se zdroji – vytvoření záznamu, čtení zdroje, úprava záznamu a smazání záznamu. Tyto operace jsou známé pod označením *CRUD (create, read, update, delete)*. [9]

Metody pro práci se zdroji:

- GET – metoda pro získání dat o zdroji,
- POST – metoda pro požadavek o vytvoření nového záznamu,
- PUT – metoda pro editaci existujícího záznamu,
- DELETE – metoda pro smazání existujícího záznamu.

Každá metoda je jednoznačně určena metodou HTTP požadavku. Mezi hlavní výhody používání REST architektury je oddělení klientské a serverové části, což je zejména výhodné při vývoji na více platformách, konkrétní platforma může mít vlastní implementaci klientské části, přičemž rozhraní pro poskytování dat je jen jedno. Další výhoda je jednoznačná identifikace zdroje podle jeho URI.

1.2.1 Nette Framework

„Nette je rodina vyspělých a samostatně použitelných komponent pro programovací jazyk PHP, které dohromady tvoří framework, vyhodnocený jako 3. nejpopulárnější na světě.“ [10] Autorem frameworku je David Grudl. V současnosti patří framework pod organizaci Nette Foundation. Framework vznikl v roce 2004 a v roce 2008 byl zpřístupněn jako open-source pod licencí BSD. [10]

1.2.1.1 Architektura MVC

Nette Framework využívá návrhový vzor MVC, který rozděluje aplikaci do tří komponent: [11]

- **Controller** – zpracovává příchozí HTTP požadavky a aplikační logiku předává na model, jehož výsledek je dále předán pohledu,
- **Model** – aplikační a datová vrstva aplikace, model o existenci kontroleru ani pohledu neví,
- **View** – pohled, tedy vrstva aplikace, která zobrazuje získaná data.

1.2.2 OAuth

V tradiční architektuře klient-server se při přístupu k chráněným zdrojům (anglicky *protected resources*) musí uživatel nejprve ověřit. Ověření typicky probíhá pomocí přihlašovací jména a hesla. Při poskytování těchto citlivých údajů třetím stranám může dojít k možným bezpečnostním rizikům: [12]

- Pro trvalý přístup je od třetí strany vyžadováno uchovávání uživatelských přihlašovacích údajů (anglicky *user credentials*). Ty jsou typicky ukládány v textové formě, nebo za použití asymetrické šifry[1]. Při úniku těchto dat získá útočník přístup k výše uvedenému zdroji.
- Třetí strana má neomezený přístup ke zdrojům zpřístupněné aplikace.
- Uživatel nijak nemůže konkrétní aplikaci třetí strany odebrat přístup, protože ten je vázán na přihlašovací údaje. Při změně přihlašovací údajů je přístup zamítnut všem třetím stranám.

OAuth tyto nedostatky řeší pomocí přístupových tokenů (anglicky *access token*). Jedná se o tajný, náhodně vygenerovaný řetězec znaků, který se váže k určitému uživateli a určitému klientovi (označení pro třetí stranu). Přístupový token má také obvykle svou expirační dobu, po které přestává být platný. K danému přístupovému tokenu se váže obnovovací token (anglicky *refresh token*), který slouží pro vygenerování nového přístupového tokenu. [12]

Přístupový token se následně posílá s každým požadavkem na server a v případě blížící se expirace ho klient pomocí obnovovacího tokenu nahradí novým přístupovým tokenem.

[1] kvůli možnosti zpětného dešifrování

1.3 Doporučovací systémy

Doporučovací systémy (anglicky *Recommender Systems*) se snaží predikovat hodnocení či preferenci uživatele vůči danému předmětu. Tato predikce se může vztahovat na různé rozhodovací procesy a není ve své podstatě nijak omezena. Předmětem tedy můžeme rozumět libovolnou věc, kterou lze doporučit – hudbu, článek, tričko či například ubytování. Mezi nejpoužívanější doporučovací techniky patří: [13]

- **Kolaborativní filtrování** (anglicky *Collaborative filtering*) je princip doporučování založený na podobnosti uživatelů. Princip je detailněji vysvětlen v podkapitole Kolaborativní filtrování.

- **Doporučování na základě obsahu** (anglicky *Content-based recommendation*) je princip založený na doporučování předmětů, které jsou podobné těm, o které uživatel v minulosti projevil zájem. Uživatel například zhlédl film z kategorie komedie a tudíž mu systém bude doporučovat další filmy z kategorie komedie.

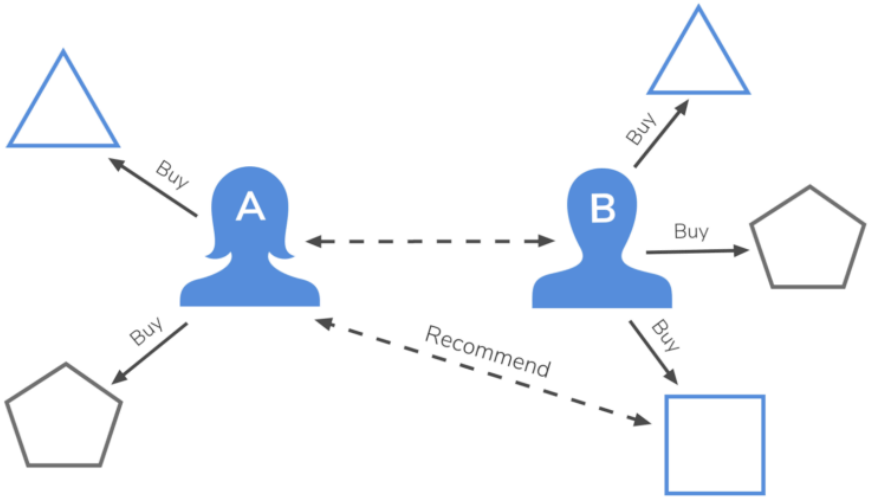
- **Demografické doporučování** (anglicky *Demographic recommendation*) je princip doporučování, jehož předpokladem jsou různé uživatelské potřeby, které se odvíjejí na základě jazyka či země, odkud uživatel pochází. Výhodou tohoto doporučování je jednoduchost získání těchto informací. Například jazyk prohlížeče, který uživatel používá, se automaticky odesílá společně s každým *HTTP* požadavkem. Uživatelovo přibližnou lokaci zase lze extrahovat z IP adresy pomocí služeb jako www.iplocation.net

- **Znalostní doporučování** (anglicky *Knowledge-based recommendation*) je princip doporučování založený na doporučování takových předmětů, které explicitně souvisí s řešením nějakého konkrétního problému. Výhodou tohoto principu je fakt, že narozdíl od doporučovacích systémů jako je *kolaborativní filtrování* nebo *doporučování na základě obsahu* nemá studený start (anglicky *cold start*). To znamená, že ho lze používat instantně bez nutnosti získání určité spodní hranice uživatelů, jako tomu je u výše zmíněných doporučovacích systémů.

- **Hybridní doporučovací systém** (anglicky *Hybrid recommender*) kombinuje prvky různých doporučovacích systémů, aby se předešlo nedostatkům jednotlivých systémů. Příkladem hybridního systému může být kombinace *demografického doporučování*, které se použije při první návštěvě uživatele na internetovém obchodě, dále může následovat *znalostní doporučování*, které se aktivuje při návštěvě určité podstránky obchodu, která se týká specifické oblasti, ke kterému obchod nabízí řešení. Pokud se uživatel stane pravidelným zákazníkem a provede sérii objednávek, obchod může začít využívat *kolaborativního filtrování*.

1.3.1 Kolaborativní filtrování

Kolaborativní filtrování (anglicky *Collaborative filtering*) je jedna z nejpopulárnějších technik doporučovacích systémů [13]. Funguje na principu predikování položek na základě korelace mezi lidmi s podobnými preferencemi. Jinými slovy je založena na myšlence, že uživatelé, kteří v minulosti projevili zájem o stejné předměty, projeví stejný zájem i v budoucnu. Uživateli A může být s velkou mírou doporučen předmět, o který uživatel B nově projevil zájem.



Obrázek 1.3: Ukázka kolaborativního filtrování [14]

Doporučování na základě podobnosti uživatelů

Tato metoda spočívá v nalezení takové podmnožiny uživatelů, jejichž hodnocení je nejvíce podobné právě vůči zkoumanému jedinci. Tuto množinu uživatelů označujeme jako nejbližší sousedy (anglicky *nearest neighbors*). Z těchto hodnocení je vybráno k-předmětů, který zkoumaný jedinec ještě neohodnotil a ty jsou mu doporučeny.

Míra kvality doporučení je pak určena podobností funkcí $f : U \times U \rightarrow \mathbb{R}$, kde U je uživatel. Výsledek pravděpodobností funkce je pak reálné číslo vyjadřující míru podobnosti.

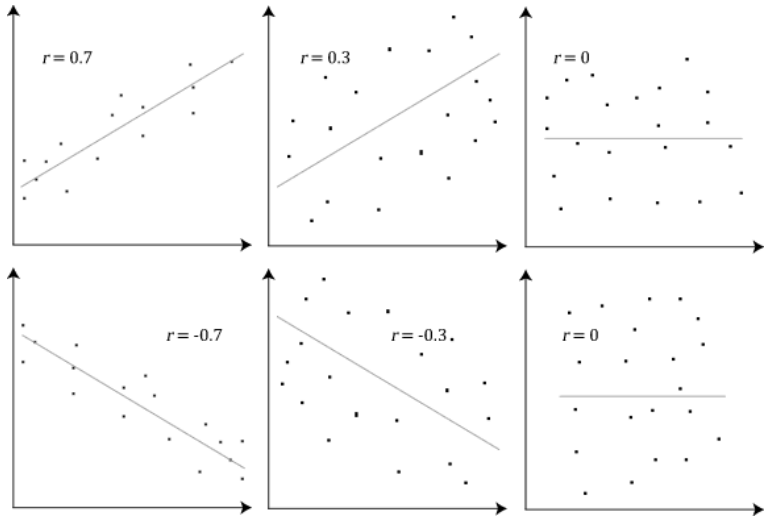
Jedna z nejpopulárnějších podobnostních funkcí je **Pearsonův korelační koeficient**. Jedná se o funkci, jejíchž výsledek je reálné číslo na intervalu $< -1, 1 >$ a je označován jako korelační koeficient. [15]

Označme X a Y jako dva vstupní náhodné vektory představující uživatele X a Y. Dále označme $\overline{X}$ jako výběrový průměr hodnocení uživatele X a $\overline{Y}$ jako výběrový průměr hodnocení uživatele Y. Vzorec pro výpočet Pearsonova korelačního koeficientu je definován následovně:

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2(Y_i - \overline{Y})^2}} \tag{1.1}$$

Korelační koeficient pak můžeme kategorizovat do tří skupin na základě toho, jak se bude chovat proměnná X_i při růstu proměnné Y_i.

- **Pozitivní korelace** nastává, když $\rho \in (0, 1>$, pak proměnná X_i má tendenci růstu (kladná lineární závislost).
- **Záporná korelace** nastává, když $\rho \in <\text{-}1, 0)$, proměnná X_i má tendenci klesat (záporná lineární závislost).
- **Nulová korelace** nastává, když $\rho = 0$, proměnná nemá tendenci růst, ale ani klesat.



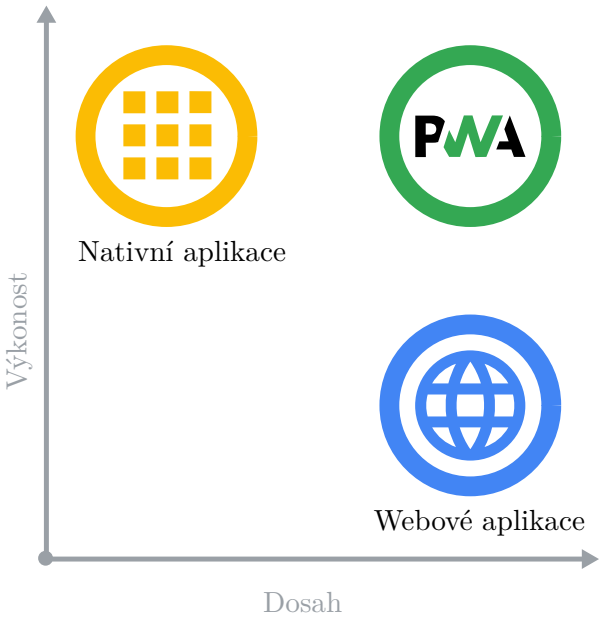
Obrázek 1.4: Vizualizace lineární závislosti. Převzato z [16]

Systémy založené na kolaborativním filtrováním jsou také známé pod pojmem *automatické doporučovací systémy*, protože automaticky doporučují předměty na základě chování/interakcí svých uživatelů. [13]

Kapitola 2

Progresivní webové aplikace

Progresivní webová aplikace, označováno jako PWA, kombinuje prvky webové aplikace a prvky nativní aplikace, přičemž staví na principech *SPA (Single page aplikací)*. Výstupem je taková aplikace, která je uživatelsky maximálně přívětivá a působí jako nativní aplikace. [17]



Obrázek 2.1: Pomyslné zařazení PWA [17]

2.1 Kritéria

Společně s konceptem progresivních webových aplikací Berriman a Russell vydefinovali devět základních vlastností, která by každá webová aplikace, jež

chce být progresivní webovou aplikací, měla splňovat: [18]

- Responzivní,
- Nezávislá na připojení,
- Progresivní (jako nativní aplikace),
- Aktuální,
- Bezpečná,
- Dohledatelná,
- Podporuje znovuzapojování uživatele,
- Instalovatelná,
- Prolinkovaná.

2.1.1 Responzivita

Aplikaci lze používat na různých typech zařízení (stolní počítač, tablet, telefon), protože se vždy adekvátně přizpůsobí obrazovce. Responzivity lze dosáhnout pomocí technologie Media Queries, které se aplikují v kaskádových stylech. [19]

Díky vysoké podpoře dotazů na médium v prohlížečích (98,15% [20]) lze jednoduše detekovat, zda zařízení splňuje určité podmínky. Typicky se jedná o podmínky, zda zařízení má menší, resp. větší, šířku obrazovky (nebo výšku obrazovky). V případě splnění podmínky se provede sada pravidel, která provede úpravu příslušných stylů.

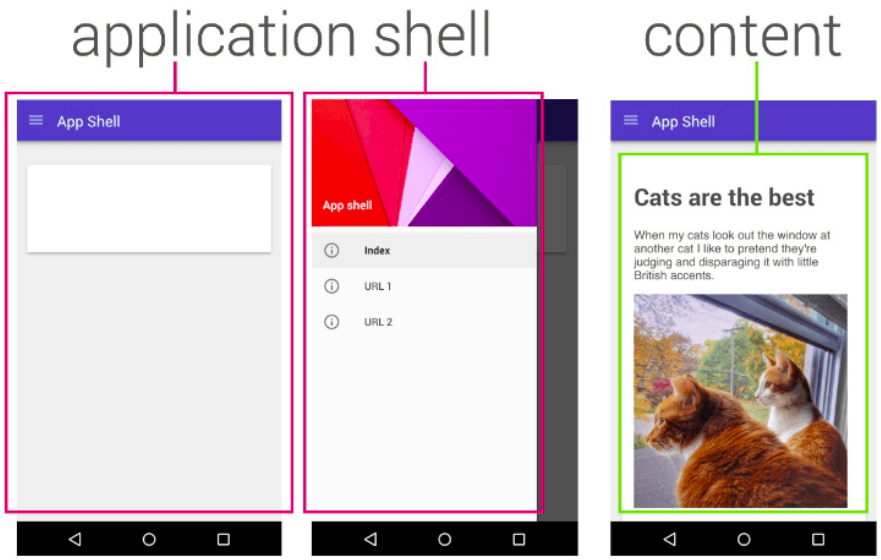
2.1.2 Progresivita (působit jako nativní aplikace)

Zajištění, aby aplikace působila jako nativní, je abstraktní pojem, který se ale dá lépe specifikovat tím, že aplikace má podobné rozhraní, jako aplikace nativní. Tato architektura se nazývá *aplikační schránka* (application shell).

Základním principem této architektury je rozdělení vizuální části aplikace do dvou částí.

- **Aplikační schránka** (anglicky *application shell*) – statický – menu, hlavička, patička.
- **Obsah** (anglicky *content*) – dynamický obsah.

Aplikační schránka má být co nejrychleji uživateli zobrazena a následně uložena do keše, aby při dalším načtení aplikace nedocházelo k opětovanému načítání prostředků nutných k zobrazení aplikační schránky. [21]



Obrázek 2.2: Znázornění architektury Aplikační schránka [21]

Tato architektura úzce souvisí s konceptem *single page aplikací* (známé také jako *one page aplikace*), kde se pouze jednou provedlo načtení aplikační schránky(menu, hlavička, apod.) a následně se dynamicky měnil pouze obsah. Progresivní webová aplikace tento princip rozšiřují o uložení aplikační schránky do keše.

Obsahová část je dynamická a během načítání uživateli dává zpětnou vazbu o průběhu, typicky pomocí animovatelných prvků. S těmito principy pak aplikace působí více jako nativní.

2.1.3 Nezávislost na připojení

Pomocí technologie *Service worker* lze uložit zdroje aplikace do persistentního uložiště prohlížeče, ze kterého pak jsou data použity při dalším načtení. Pokud nebude k dispozici připojení k internetu a aplikace bude mít svou *aplikační schránku* již uloženou, pak může uživateli zobrazit hlášku o tom, že není připojen k internetu či mu zobrazit již nakešované data.

Technologie Service worker je v konceptu PWA klíčová a proto bude detailněji rozebrána v samostatné kapitole.

2.1.4 Aktuálnost

Aplikace je aktualizována na pozadí a je tak vždy aktuální. Tento proces je prováděn pomocí technologie Service worker, který je schopen interagovat s prohlížečem i po zavření aktuální stránky. [18]

2.1.5 Bezpečnost

Webová aplikace je uživateli servírovaná přes protokol HTTPS. [18]

2.1.6 Dohledatelnost

Stránka deklaruje, že je aplikací tím, že obsahuje textový soubor *manifest.json* ve formátu JSON, který vyhledávače a prohlížeče mohou přečíst. Soubor *manifest.json* je umístěný v kořenovém adresáři aplikace a obsahuje název aplikace, popis aplikace, odkazy na ikony, které se mají použít při instalaci a další.

2.1.7 Podporuje znovuzapojení uživatele

Progresivní webové aplikace využívají push notifikace, pomocí kterých zasílají uživateli upozornění či oznámení. Push notifikace také může obsahovat ovládací prvky. [18]

2.1.8 Instalovatelnost

V případě mobilního zařízení či tabletu lze přidat aplikaci na plochu, ze které lze pak aplikaci znovu spustit. Pro nedotykové zařízení se chování liší v závislosti na typu webového prohlížeče. Ikona a její možné velikosti jsou definovány v souboru *manifest.json*. Pro používání aplikace není vyžadována její instalace.

2.1.9 Odkazovatelnost

Aplikace lze jednoduše sdílet pomocí *URL*, která jednoznačně odkazuje na danou stránku. [18]

2.2 Technické požadavky

Původní požadavky na vymezení, co lze považovat za progresivní webovou aplikaci, nebyly jednoznačné. V roce 2017 Jeremy Keith navrhl spodní hranici, při které lze označit webovou aplikaci za progresivní: [22]

- komunikace je zajištěna přes zabezpečený protokol *HTTPS*,
- aplikace obsahuje *Service worker*,
- aplikace obsahuje validní soubor *manifest.json*.

V případě nedodržení z některých výše uvedených vlastností prohlížeč nemusí instalaci nabídnout.

2.2.1 Service worker

Service worker je událostmi řízený (anglicky *event-driven*) worker, který pro webovou aplikaci slouží jako kešovací proxy server. Service worker, stejně jako worker, je reprezentován JavaScriptovým souborem, jehož instrukce jsou vykonávány v prohlížeči přiděleném vlákně. Service worker se vždy vztahuje k doméně, na které byl zaregistrován, přičemž doména může mít v jeden moment aktivního pouze jednoho Service workera. Podmínkou pro registraci je komunikace na zabezpečeném kanále (HTTPS). [23]

Service worker může přerušovat síťové požadavky, které jsou vyvolané pomocí Fetch API a spadají do rozsahu, ke kterému má Service worker přístup. Rozsah je jednoznačně určen umístněním soubor, kterým je Service worker reprezentován. V případě umístění souboru do kořenového adresáře aplikace získává Service worker kontrolu nad všemi síťovými požadavky. V případě umístění do pod adresáře může Service worker přerušovat pouze požadavky směřující do tohoto pod adresáře a jeho dalších pod adresářů. [23]

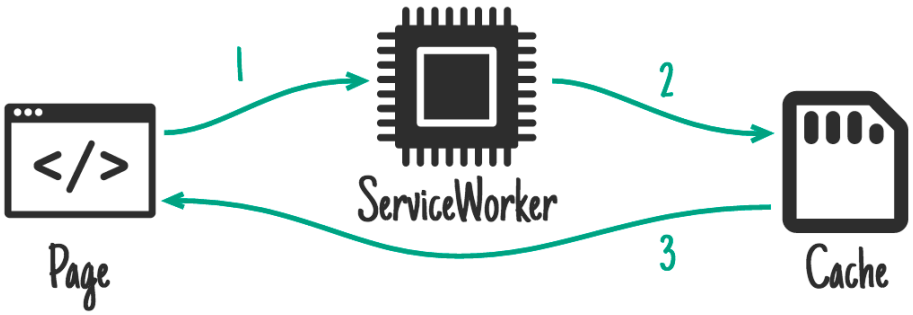
2.2.1.1 Kešovací strategie

Při zaslání požadavku na server, pomocí Fetch API, je požadavek přerušen Service workerem [24]. Ten na základě příslušné kešovací strategorie rozhodne, jak bude požadavek zpracovávat. Vzhledem k různorodosti požadavků vzniklo několik strategií, jakým způsobem požadavky efektivně zpracovávat.

Pouze keš

Pouze keš (anglicky *cache only*) je strategie, ve které jsou na všechny požadavky vráceny data z keše [24]. Tento přístup je vhodný pro načítání statických souborů (například k vykreslení aplikační schránky (obrázek 2.2)). Data jsou do keše uloženy při instalaci Service workera, jehož životní cyklus je popsán v kapitole 2.2.2.

Nevýhodou této metody je situace, při které je v keši uložený neplatný či nefunkční záznam. Vzhledem k povaze strategie nedojde k jeho přenačtení.

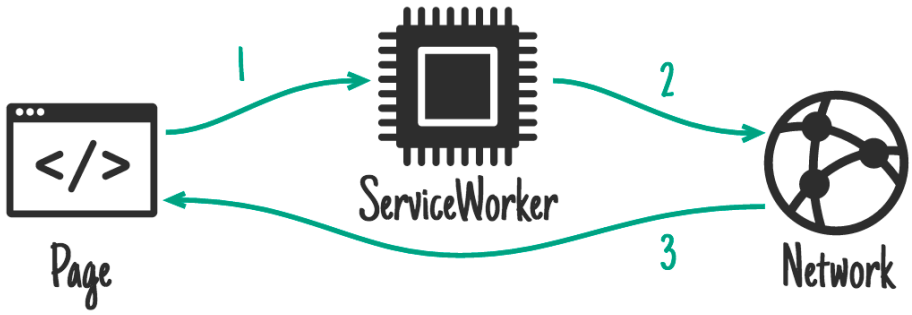


Obrázek 2.3: Service worker – pouze keš [24]

Pouze síť

Strategie pouze síť (anglicky *network only*) je přímým opakem strategie *Pouze keš*. Tato strategie je vhodná pro všechny požadavky, které není možné kešovat. Typicky se jedná o požadavky, které vytvářejí nebo upravují obsah [24].

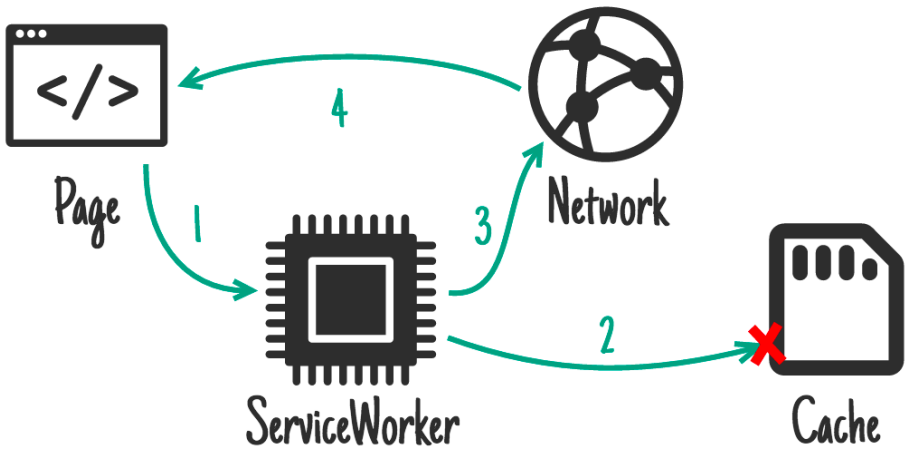
Nevýhodou této strategie je nemožnost zpracovat požadavek při ztrátě internetového připojení nebo nedostupnosti serveru.



Obrázek 2.4: Service worker – pouze síť [24]

Nejdříve keš

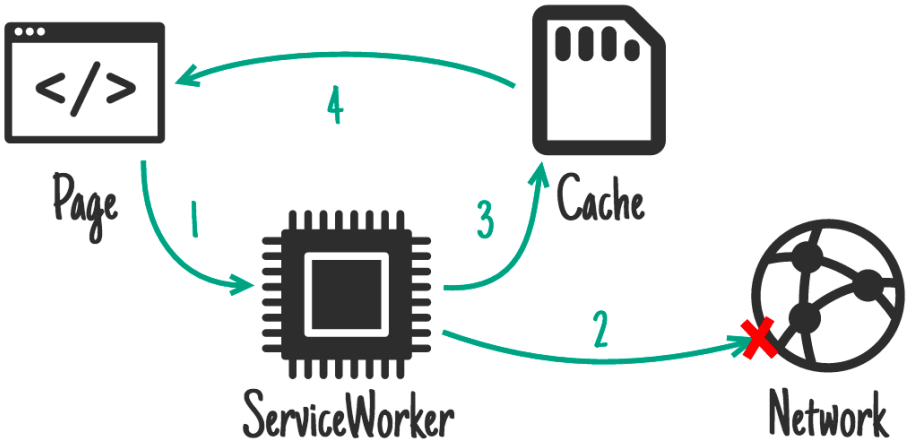
Nejdříve keš (anglicky *Cache first*) je strategie, při které Service worker nejdříve zkontroluje, zda je pro daný požadavek uložená odpověď v keši. Pokud ano, vrátí tuto zakešovanou odpověď. V opačném případě je požadavek odeslán na server. Pokud je požadavek úspěšně odbaven a ze serveru přijde validní odpověď, Service worker ji uloží do keše a použije ji při dalším požadavku. Odpověď je následně předána žadateli. [24]



Obrázek 2.5: Service worker – nejdříve keš [24]

Nejdříve síť

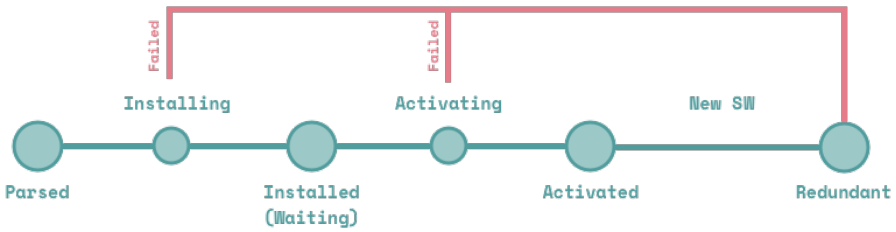
Nejdříve sít (anglicky *Network first*) je strategie, při kterém je požadavek vždy odeslán na server. V případě jeho selhání je navrácena zakešovaná odpověď. V opačném případě Service worker uloží tuto odpověď do keše a následně ji vrátí žadateli. [24]



Obrázek 2.6: Service worker – nejdříve síť [24]

2.2.2 Životní cyklus

Životní cyklus (anglicky *life cycle*) Service workera začíná při jeho zaregistrování a je kompletně oddělen od životního cyklu webové aplikace. [25] Na základě stavu, ve kterém se Service worker nachází, je možné vykonávat příslušné akce v aplikaci. Jednotlivé stavy a jejich návaznosti jsou zachyceny na následujícím obrázku.



Obrázek 2.7: Service worker – životní cyklus [26]

1. **Načtení skriptu** (anglicky *Parsed*) je počáteční stav životního cyklu Service workera. Do počátečního stavu se Service worker dostane, pokud

je webovou aplikací úspěšně načten, nikoliv však zpracován.

2. **Instalování** (anglicky *Installing*) je stav, ve kterém se Service worker nachází po úspěšném načtení skriptu. Tato fáze je vhodná pro nahrání statických zdrojů do keše. Pro servírování statických zdrojů se využívá kešovací strategie *Nejdříve keš*.

3. **Instalace – čekání** (anglicky *Installed – waiting*) je stav, ve kterém je instance Service workera validní, nikoliv však aktivována. Service worker tedy v tento moment ještě nemá kontrolu nad dokumentem a nemůže přerušovat žádné požadavky.

4. **Aktivování** (anglicky *Activating*) je stav, při kterém probíhá aktivace. Do tohoto stavu Service worker přejde, pokud je splněna alespoň jedna z následujících podmínek:

   - žádný jiný Service worker není aktivní,
   - je programovou cestou vynuceno přeskočení[2] čekací fáze,
   - od okamžiku, kdy Service worker vstoupil do čekací fáze, uživatel uzavřel všechny záložky, ve kterých je aplikace spuštěna a opětovně stránku otevřel.

5. **Aktivováno** (anglicky *activated*) je stav, ve kterém má Service worker plnou kontrolu nad aplikací napříč všemi instancemi (otevřenými okny, záložkami). Může tedy přerušovat síťové požadavky vyvolané pomocí technologie Fetch API.

6. **Nadbytečný** (anglicky *redundant*) je poslední fáze životní cyklu, do které Service worker přejde v případě selhání instalační či aktivační fáze nebo v případě jeho nahrazením novým Service workerem. [26]

2.3 Podpora v prohlížečích

Přestože PWA není ze své podstaty založená na jedné konkrétní technologii, ale spíše na myšlence nejlepší uživatelské zkušenosti, můžeme i přesto za jádro PWA považovat technologii Service worker. Považujeme-li i za nezbytné mít možnost instalovat aplikace na plochu, pak vyžadujeme i technologii jménem Web App Manifest. V případě Push notifikací se jedná o technologii Push API.

- **Service worker** – skript běžící na pozadí, odděleně od webové stránky. V kontextu progresivních webových aplikací zodpovídá za doručování

[2] přeskočení čekací fáze může mít fatální následek pro staré instance, jimž byla část obsahu servírována starou verzí Service workera

push notifikací, odkládání posílání požadavků na server v případě slabého připojení a především slouží jako programovatelná proxy služba [23]. Viz kapitola 2.2.1

- **Manifest** – manifest je krátký soubor ve formátu *JSON*, který popisuje webovou aplikaci. Obsahuje informace o sadě ikon (které mají být při instalaci použity), výchozí URL (použita při zapínání aplikace) a zobrazovací režim (na výšku, na šířku) [18]. Prohlížeč *Safari* od společnosti Apple tento standard zatím neimplementuje, což způsobuje značné komplikace, viz kapitola 2.3.1

Tabulky 2.1 a 2.2 zachycují podporu těchto technologií v prohlížečích. Tabulky rovněž obsahují informaci o globálním zastoupením jednotlivých prohlížečů, přičemž jsou vyfiltrovány prohlížeče jejichž celkové zastoupení je menší 1,99 %. Kompletní tabulky s výčtem všech prohlížečů se nacházejí v příloze této práce.

Tabulka 2.1: Podpora technologie Service worker v prohlížečích [27]

Prohlížeč	**Zastoupení**	
	Podporované	Celkem
Chrome – Android (od verze 81)	34,16 %	34,16 %
Chrome (od verze 45)	30,06 %	30,35 %
iOS Safari (od verze 11,4)	12,83 %	13,63 %
Firefox (od verze 44)	4,07 %	4,37 %
Safari (od verze 11.1)	3,71 %	4,04 %
Samsung Internet (od verze 4)	3,53 %	3,53 %
Edge (od verze 17)	2,05 %	2,12 %
UC Browser (od verze 12.12)	1,99 %	1,99 %
Ostatní prohlížeče	0,76 %	5,81 %
Celkem	**92,40 %**	**94,19 %** (100 %)

Tabulka 2.2: Podpora technologie Web App Manifest v prohlížečích [28]

Prohlížeč	**Zastoupení**	
	Podporované	Celkem
Chrome – Android (od verze 81)	34,16 %	34,16 %
Chrome (od verze 73)	28,58 %	30,35 %
iOS Safari	0,00 %	13,63 %
Firefox	0,00 %	4,37 %
Safari	0,00 %	4,04 %
Samsung Internet	0,00 %	3,53 %
Edge	0,00 %	2,12 %
IE	0,00 %	2,05 %
UC Browser – Android	0,00 %	1,99 %
Ostatní prohlížeče	0,00 %	3,76 %
Celkem	**62,75 %**	**96,24 %** (100 %)

Data z tabulek 2.1 a 2.2 byly extrahovány přímo z datových zdrojů služby caniuse.com. Důvodem toho rozhodnutí je fakt, že daná webová aplikace provádí ve výpočtech významné zaokrouhlování a vynechává verze prohlížečů, jejichž celkový podíl zastoupení je menší než 0.1%. Dále aplikace neposkytuje informaci o procentuálním zastoupení prohlížeče od konkrétní verze, což je ve zkoumání podpory výše uvedených technologií klíčové.

2.3.1 Prohlížeč Safari a systém iOS

O světovém technologickém gigantu Apple je známo, že valnou většinu technologií implementují s podstatnou časovou prodlevou a jejich rozhodnutí mnohdy zabraňuje růstu nových technologií. [29] Nechť je příkladem rozhodnutí, říkající že „Veškeré aplikace procházející web musí využívat WebKit framework a Webkit Javascript [30]“, čímž znemožňují ostatním vývojářům využívat jiné vykreslovací jádro. Taktéž není možné změnit výchozí prohlížeč v zařízeních se systémem iOS.

V souvislosti s PWA a jejich klíčovými prvky jako Service worker či Manifest (zmíněné v předchozí části) se u prohlížeče Safari naráží na určité nedostatky či nestandardní implementaci. Dle tabulky 2.1 prohlížeč Safari implementuje technologii Service worker, nepodporuje ale technologie s ní související – *Push notifikace* [31], *Background sync* [32]. Dle tabulky 2.2 prohlížeč Safari rovněž neimplementuje podporu *Web App Manifest*. Ten lze do jisté míry nahradit pomocí specifických meta/link značek v hlavičce HTML sou-

boru, přičemž jejich množství je oproti Manifestu značně omezené. Je nutné podotknout, že tyto meta značky nebyly primárně vytvořeny pro účely PWA i přes jejich vzájemnou paralelu. Výčet meta značek je shrnut následovně: [33]

- **apple-mobile-web-app-capable (meta tag)** nastavuje, zda se má aplikace zobrazovat v režimu celé obrazovky (anglicky *full screen*). Možné hodnoty jsou „yes“ a „no“. Částečně odpovídá vlastnosti *display: "standalone"* v Manifestu.
- **apple-mobile-web-app-title (meta tag)** nastavuje název aplikace. Ten se následně zobrazuje jako jméno ikony „nainstalované“ aplikace.
- **apple-touch-startup-image (link tag)** nastavuje načítací obrazovku (anglicky *splash screen*), která se zobrazí po otevření aplikace z plochy.
- **apple-mobile-web-app-status-bar-style (meta tag)** nastavuje barvu pozadí v horní části obrazovky. Výběr tvoří pouze tři možnosti *default* – bílá, *black* – černá, *black-translucent* – barva reflektující pozadí stránky (konkrétně HTML značky *<body>*).

Zásadním rozdílem mezi prohlížeči podporující základní funkcionalitu PWA je ten, že v případě prohlížeče Safari nedojde k automatické výzvě ani jinému signálu pro instalaci. Pouze je zde možnost přidání webové stránky na plochu, která při svém přidávání provede instalaci, přičemž nevyvolává událost *appinstalled*, na kterou by aplikace mohla patřičně zareagovat. Nevyvolá jí z toho důvodu, že prohlížeč Safari neimplementuje technologii Web App Manifest.

Z důvodu častého neporozumění rozdílu mezi pojmy *instalace* a *přidat na plochu* docházelo k využívání značek (zmiňovaných v seznamu 2.3.1) nevhodným způsobem, které měly za následek znemožnění procházení celé aplikace. Nemožnost přímé instalace, ale pouze její přidání na plochu, má za následek i další odlišnosti:

- Uživatel si při procesu přidávání aplikace na plochu může zvolit vlastní jméno aplikace.
- Výchozí URL je nastaveno podle URL, ze které uživatel provedl přidání na plochu [34]. Programátor tedy toto chování musí[3] řešit na úrovni aplikačního kódu.

2.4 Případové studie

Případová studie (anglicky *case study*) je jednou z metod kvalitativního výzkumu. Zkoumáme při ní do detailu, jak daná služba či produkt přispěly

[3]pouze pokud vývojář chce zachovat konzistentní chování s prohlížeči, které podporují technologii Web App Manifest

koncovému zákazníkovi. Dle [35] jsou případové studie třetí nejefektivnější marketingový kanál v komunikaci firma-firma (anglicky *business-to-business*).

Pro kvalitní uživatelský zážitek (anglicky *user experience*) z webové aplikace či webové stránky jsou z technického hlediska důležité především následující faktory:

- **Množství přeneseného obsahu** (anglicky *bandwitch usage*) – minimální množství dat, které je nezbytné k fungování aplikace či stránky.
- **Čas prvního vykreslení obsahu** (anglicky *First contentful paint*) – čas, kdy se na obrazce uživatele vykreslí první libovolný obsah, například text stránky.
- **Čas první smysluplného vykreslení obsahu** (anglicky *First meaningful paint*) – čas, kdy je většina obsahuje viditelná, přičemž některé části se ještě mohou donačítat.
- **Čas do první interakce** (anglicky *Time to interactive*) – čas, po kterém je možné se stránkou začít pracovat. Z technického hlediska to znamená, že hlavní jádro není blokované a dokáže v určitém časovém intervalu reagovat na uživatelské vstupy. Například zareagovat na stisknutí klávesy nebo pohybu myši.

Z vyšší míry uživatelského zážitku lze očekávat i vyšší míru konverze. Mezi nejznámější kvalitně zdokumentované studie týkající se porovnání webové aplikace s progresivní webovou aplikací patří případová studie společnosti Pinterest a případová studie společnosti Twitter.

2.4.1 Pinterest

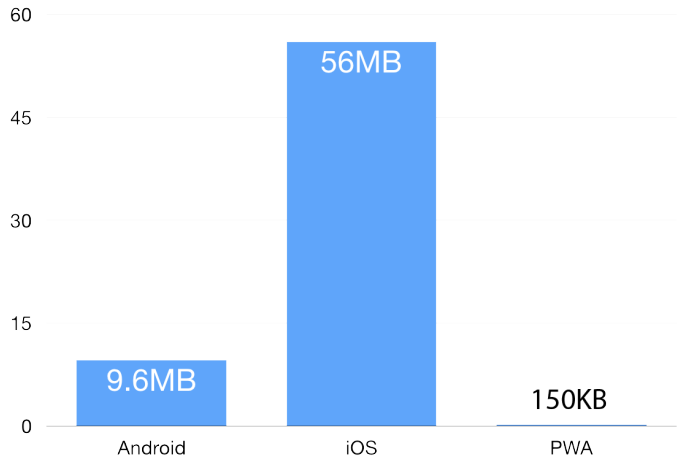
Pinterest je sociální síť, kde lidé hledají inspiraci a nápady pro jejich záliby a koníčky. Síť byla založena v lednu roku 2010. Jejím tvůrcem je Ben Silbermann. [36] Princip této sociální sítě spočívá v myšlence, že uživatel do systému nahraje fotografii, kterou patřičně popíše a přidá k ní klíčová slova (anglicky *keywords)*. Podle těchto klíčových slov mohou ostatní uživatelé tento příspěvěk vyhledat. Uživatelé si dále mohou z těchto příspěvků vytvářet tématické kolekce, které mohou sdílet s ostatními uživateli.

Analýza nové PWA oproti staré webové aplikaci: [36]

- Množství stráveného času přesahující hranici 5 minut – nárůst o 40%,
- Množství prokliků na reklamy – nárůst o 50%,
- Znovu zapojení uživatele – nárůst o 60%.

Rychlostí porovnání mezi PWA a starou webovou aplikací: [36]

- Čas prvního vykreslení – 1,8 sekund oproti původním 4,2 sekund, zrychlení o 57 %,
- Doba do interaktivity – 5,6 sekund oproti původním 23 sekund, zrychlení o 76 %,
- Celková velikost přenášených kaskádových stylů – 6 kB oproti původním 150 kB,
- Celková velikost přenášených javascriptových souborů – 150 kB oproti původním 620 kB.



Obrázek 2.8: Porovnání velikostí PWA s nativními aplikacemi [36]

V případě porovnání velikosti nové progresivní webové aplikace s velikostmi nativních verzí aplikace Pinterest pozorujeme znatelný velikostní rozdíl. Je nutné podotknout, že v případě PWA se kód načítá dodatečně, tedy podle toho, na které stránce se uživatel nachází.

2.4.2 Twitter

Twitter je sociální síť založená na principu mikroblogování. Jedná se o systém, pomocí kterého uživatel může číst, posílat a sdílet příspěvky, které se označují jako *tweety*. Tweet by se dal volně přeložit jako „cvrlikání“ či „štěbetání“. Síť byla založena v březnu roku 2006 [37]. Twitter má přibližně 328 milionů aktivních uživatelů měsíčně [38].

Twitter Lite je progresivní webové aplikace, napsána v Reactu, která byla vydána v dubnu roku 2017. Důvodem vytvoření aplikace Twitter Lite byl

fakt, že velká část světové populace nemá k dispozici rychlé ani spolehlivé internetové připojení [39].

Dle [38] Nicolas Gallagher, hlavní vývojový technik pro Twitter Lite poznamenal „Twitter Lite je nyní nejrychlejší, nejjednoduší a nejspolehlivější cesta jak používat Twitter. Webová aplikace soupeří s výkonem té nativní, ale její velikost je menší než 3% oproti nativní aplikaci Twitter pro Android.“

Shrnutí přínosu PWA pro společnost Twitter: [38]

- 65% nárůst v počtu navštívených stránek na jedno uživatelské sezení (anglicky *session*),
- 75% nárůst v počtu odeslaných Tweetů,
- 20% pokles opuštění stránky,
- Na 3G síti se aplikace plně načte do pěti sekund,
- Velikost plně funkční PWA je pouze 600 kB, přičemž velikost nativní aplikace Twitter pro Android je 23.5 MB,
- Při opětovaném navštívení aplikace z pomalého internetového připojení se aplikace načte do tří sekund.

2.5 Porovnání s nativními aplikacemi

S příchodem progresivních webových aplikací se může zdát, že mohou nativním aplikacím plně konkurovat, odpověď ale není tak přímočará.

Od doby, kdy se web přestal využívat pouze pro čtení dokumentů se začala značná část jeho funkcionalit přebírat z nativních aplikací, což částečně přispělo k rozvoji konceptu PWA. Nechť jsou příkladem následující technologie: Push notifikace, Geolokace, Vibrace, Bluetooth nebo WebRTC.

Dle interních metrik prohlížeče Google Chrome mají dialogy s žádostí o instalaci PWA přibližně 5–6×větší konverze než dialogy cílící na instalaci nativní aplikace. Tento fakt může být zapříčiněný znatelně delším instalačním procesem a řádově větším datovým objemem, který musí být u nativní aplikace přenesen [40].

Přehled předností a nedostatků PWA je popsáno v tabulce 2.3, za níž následuje popis předností a nedostatků nativních aplikací 2.4.

Tabulka 2.3: Výhody a nevýhody PWA

Progresivní webová aplikace	
Klady	**Zápory**
+ Nevyžaduje instalaci	- Nižší výkon
+ Automatická aktualizace aplikace na pozadí	- Nekonzistentní podpora v prohlížečích (viz kapitola 2.3)
+ Jeden sdílený kód pro všechny platformy	- Omezený přístup k systémovým prostředkům
+ Nižší vývojová cena	- Omezená velikost uložiště
+ Nízká velikost	
+ Lze vyhledat a sdílet	

Tabulka 2.4: Výhody a nevýhody nativních aplikací

Nativní aplikace	
Klady	**Zápory**
+ Vysoká spolehlivost	- Vyšší náklady na vývoj
+ Nativní zážitek	- Pro každou platformu jiný kód
+ Vysoký výkon	- Menší dosah aplikace
+ Stabilnější základ pro vývoj	- Vyšší velikost aplikace
+ Neomezená velikost uložiště	

Progresivní webové aplikace mají své klady a zápory, při plánování projektu je tedy nutné udělat rozhodnutí, které nejvíce reflektuje potřeby daného projektu. Jestliže je kladen důraz na rychlé vydání a pokrytí široké škály zařízení a není vyžadována přílišná výpočetní zátěž, pak je PWA vhodným kandidátem. Je-li nutno využívat náročných výpočetních operací a mít zaručenou podporu všech funkcionalit, pak je vhodným kandidátem nativní aplikace. Vždy je ale dobré udělat si případnou rešerši, zda jsou příslušné technologie v prohlížečích podporovány.

Podíváme-li se na případové studie společností jako je Pinterest, Twitter, Uber, Trivago a další, zjistíme, že i přes existenci nativní aplikace dává mnohdy smysl i tvorba progresivní webové aplikace.

Kapitola 3

Softwarový návrh

Softwarový návrh je jeden z termínů užívaných pro návrh aplikace. Jedná se o proces, který pomocí ověřených způsobů napomáhá přesně specifikovat jednotlivé požadavky a zjednodušit tak proces implementace.

Účelem softwarového návrhu je transformovat vstupní požadavky na jednotlivé části, které zjednoduší následnou implementaci a díky kterému je možné předejít možným chybám. Softwarový návrh je tvořen dvěma částmi: analýzou požadavků a následným návrhem řešení. [41]

3.1 Analýza požadavků

Analýza požadavků je proces, jehož cílem je především vymezení hranic systému, vyjasnění si zadání a zachycení případných omezení, které jsou na aplikaci kladeny. Analýza softwaru obsahuje informace o funkcích, kterými má systém disponovat, a rovněž o funkcích, které přímo nesouvisejí s požadavky. Proto je pro lepší konkrétnost analýza požadavků rozdělována do dvou kategorií: [41]

- **Funkční** požadavky tvoří výčet funkcionalit, kterými má aplikace ve finálním stavu disponovat. Například: „Systém bude evidovat knihy. Nebude však již evidovat jejich dodavatele“.
- **Nefunkční** požadavky, někdy označované jako obecné, tvoří výčet technických požadavků, které musí být dodrženy. Například: „Systém bude dostupný jako webová aplikace zobrazitelná v internetovém prohlížeči (IE 11, FF 48), přičemž není vyžadována responzivita“.

3.1.1 Souhrn požadavků

Následující sekce tvoří souhrn funkčních (tabulka 3.1) a nefunkčních požadavků (tabulka 3.2), kterou jsou kladeny na klientskou a serverovou část.

Tabulka 3.1: Funkční požadavky aplikace

	Popis	**Priorita**
F1	**Registrace:** Uživatel by měl mít možnost se do aplikace zaregistrovat. K registraci je nutné vyplnit následující položky: jméno, příjmení, země, e-mail a přezdívku. E-mail a přezdívka musí být napříč celou databází uživatelů unikátní.	Vysoká
F2	**Přihlášení a odhlášení:** Uživatel by měl mít možnost se přihlásit pomocí příslušného přihlašovacího jména a hesla, přičemž jako přihlašovací jméno je i možné použít e-mail nebo přezdívku.	Vysoká
F3	**Seznam všech koktejlů:** Aplikace umožňuje uživateli zobrazit všechny koktejly evidované v databázi. Seznam koktejlů má být implementován pomocí stránkování. Na jedné stránce se nachází maximálně 24 záznamů. URL adresa se mění podle aktuální stránky. Výjimku tvoří první stránka, která v URL nesmí obsahovat číslo.	Vysoká
F4	**Seznam všech koktejlů podle kategorie:** Aplikace umožňuje uživateli zobrazit všechny koktejly evidované v databázi spadající do uživatelem zvolené kategorie. Způsob procházení je stejný, jako pro požadavek *Seznam všech koktejlů*, přičemž kategorie koktejlů musí být v URL poznamenána.	Vysoká
F5	**Detail koktejlu:** Aplikace umožňuje uživateli zobrazit konkrétní stránku koktejlu, která bude jednoznačně určena svojí URL a bude možno jej sdílet. Stránka obsahuje titulní fotografii, instrukce a použité ingredience.	Vysoká

F6	**Doporučené koktejly:** Aplikace na hlavní stránce zobrazuje přihlášenému uživateli doporučené koktejly, které jsou vybrány doporučovacím systémem. Nelze-li tento seznam vyhodnotit, pak jsou uživateli doporučeny náhodné koktejly.	Vysoká
F7	**Hodnocení koktejlů:** Aplikace umožňuje přihlášenému uživateli provést hodnocení koktejlů na stupnici 1 až 5. Přičemž 1-nejhorší, 5-nejlepší. Uživatel své hodnocení může později změnit.	Vysoká
F8	**Odstranění hodnocení:** Přihlášený uživatel může své staré hodnocení kdykoliv odstranit. Odstranění musí být zahrnuto alespoň na detailu koktejlu.	Vysoká
F9	**Vyhledávání koktejlů:** Přihlášený uživatel má možnost vyhledávat koktejly podle jména.	Normální
F10	**Oblíbený koktejl:** Přihlášený uživatel může označit koktejl jako oblíbený. Později může koktejl z oblíbených položek vyřadit.	Normální
F11	**Přehled hodnocení:** Aplikace obsahuje stránku s kompletní historií uživatelova hodnocení, seřazené sestupně podle datumu, kdy uživatel koktejl ohodnotil nebo své hodnocení upravil.	Normální
F12	**Osobní nastavení:** Přihlášený uživatel má možnost změnit své osobní údaje - jméno, příjmení a přezdívku.	Normální
F13	**Změna hesla:** Přihlášený uživatel má možnost změnit své heslo.	Nízká

Tabulka 3.2: Nefunkční požadavky aplikace

	Popis	**Priorita**
N1	**Podpora prohlížečů:** Aplikace bude podporovat následující verze prohlížečů: Google Chrome (od verze 80), Firefox (od verze 74), Microsoft Edge (od verze 18) a Safari (od verze 13).	Vysoká
N2	**Knihovna React:** Aplikace bude využívat knihovnu React (verze 16.3.0 nebo vyšší).	Vysoká
N3	**Lokalizace aplikace:** podklady pro jazykové mutace musí být uloženy jako editovatelná tabulka v systému Google Sheets. Při sestavení aplikace je tato tabulka převedena do překladových souborů, které aplikace využívá.	Vysoká
N4	**PWA:** aplikace bude progresivní a bude jí možné v zařízeních s operačním systémem Android nainstalovat. V případě operačního systému iOS bude možné přidat aplikaci na plochu.	Vysoká
N5	**Responzivita:** aplikace bude plně responzivní a tedy přístupná z mobilních telefonů, tabletů a počítačů.	Vysoká
N6	**Backend:** backend bude implementovat architekturu REST API. Jeho jádro bude tvořit český Nette framework. Programovací jazyk PHP, verze 7.1.0 nebo vyšší.	Vysoká
N7	**Bezpečnost:** klientská a serverová část bude servírovaná přes protokol HTTPS.	Vysoká

Kapitola 4

Realizace

4.1 Architektura Frontendu

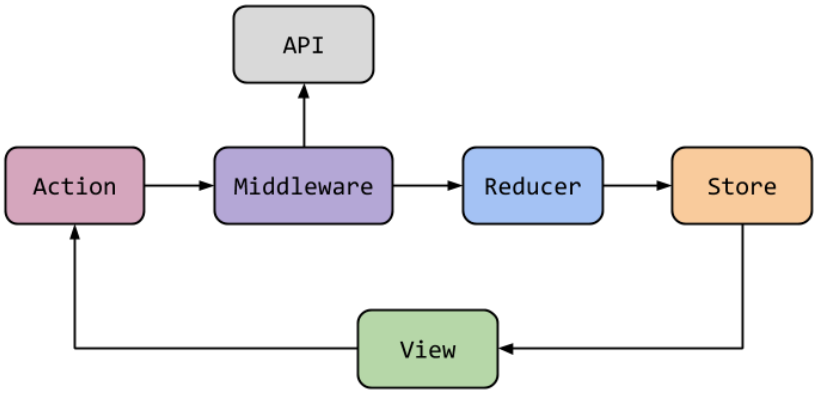
Klientskou část aplikace tvoří knihovna React (kapitola 1.1.1), jež je založena na principu znovupoužitelných komponent a deklarativním přístupu. Jednotlivé komponenty jsou v aplikaci zapouzdřeny v modulech, přičemž každý modul exportuje jenom určitou množinu těchto komponent (anglicky *component*). Komponenty jsou typicky obaleny kontejnerem (anglicky *container*), který do dané komponenty předává aplikační logiku či data z aplikačního stavu (knihovna Redux, viz kapitola 1.1.2), která komponenta následně zobrazuje. Aplikační logikou rozumíme množinu funkcí, jejichž vykonání má za následek emitování akce s příslušnými daty. Na emitovanou akci příslušně zareaguje *middleware*, který typicky provede asynchronní (tedy neblokující) volání na server. Po obdržení výsledků je emitována další akce, která popisuje výsledek volání. Na základě těchto akcí se mění stav a na základě stavu se komponenty překreslují.

4.1.1 Modulární architektura

Pro větší přehlednost a snažší udržitelnost je aplikace rozdělena na moduly. Každý modul definuje své závislosti (prostřednictvím souboru *dependencies.js*) a části, které exportuje (souborem *index.js*).

Všechny komponenty/kontejnery/funkce své závislosti importují pouze ze souboru *dependencies.js*, čímž je částečně zamezeno cyklickým závislostem[4]. Rovněž při změně implementace jednoho modulu postačuje provést úpravy pouze v *dependencies.js*. Moduly mohou být různého typu, přičemž daný typ jednoznačně určuje strukturu modulu.

[4]stav, kdy soubor A vyžaduje soubor B, přičemž soubor B zpětně vyžaduje soubor A



Obrázek 4.1: Životní cyklus aplikace – React, Redux, Redux-Saga

4.1.2 Prezentační modul

Mezi nejrozšířenější typ modulů napříč celou aplikací je prezentační modul. Ten exportuje kontejnery – což jsou komponenty, které předávají data do prezenčních komponent. Adresářová struktura takového modulu je znázorněná na následujícím obrázku.

```
modul/
├── index.js ..... soubor exportující části modulu
├── dependencies.js ..... závislosti modulu
├── components/
│   ├── …
│   └── index.js ..... soubor exportující komponenty
└── containers/
    ├── …
    └── index.js ..... soubor exportující kontejnery
```

Obrázek 4.2: Klientská část – ukázka struktury prezentačního modulu

Vhodným příkladem užití takového modulu může být zobrazení detailu koktejlu. Proces vedoucí k zobrazení detailu se skládá z následujících částí:

1. Uživatel přejde na příslušnou adresu,
2. Směrovač (anglicky *Router*) ověří požadavek a vyhledá příslušnou komponentu, která je k danému adresnímu vzoru přiřazena a provede její vykreslení,
3. Provede se kód v kontejneru (viz 4.1).

```
// src/modules/cocktails/containers/CocktailDetail.jsx
const CocktailDetail = () => {
    const cocktailId = useCurrentCocktailId();
    const requestState = useFetchCocktail(cocktailId);
    return (
        <ErrorCard {...requestState.error}>
            <Loader show={requestState.inProgress}>
                <CocktailDetailComponent id={cocktailId} />
            </Loader>
        </ErrorCard>
    );
};
```

Výpis kódu 4.1: Ukázka kontejneru prezentačního modulu

V případě chyby při zpracování požadavku je zobrazena komponenta *ErrorCard*. V případě načítání je zobrazena komponenta *Loader* a v případě úspěšného načtení je zobrazena komponenta *CocktailDetailComponent*. Viz ukázka 4.2.

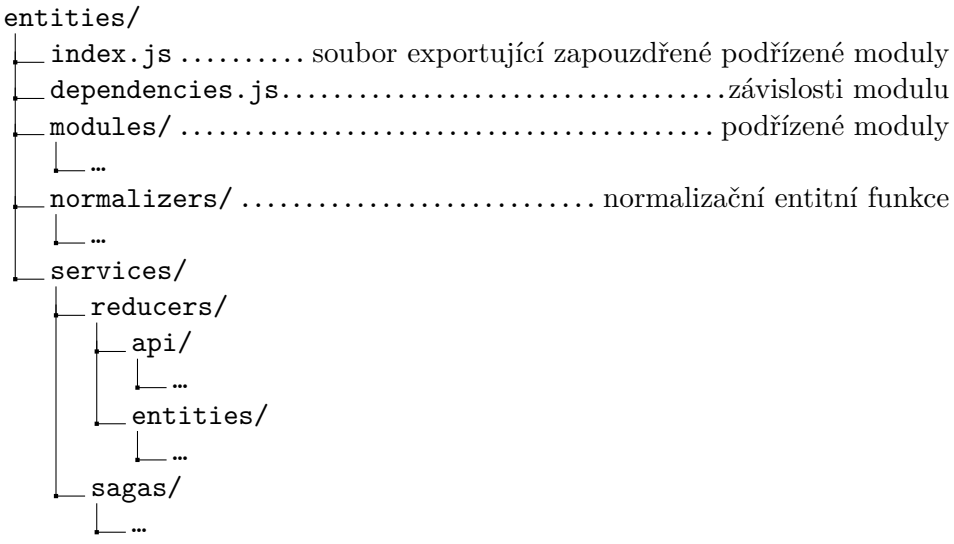
```
// src/modules/cocktails/components/CocktailDetail.jsx
const CocktailDetail = ({ id }) => {
    const data = useSelector(state =>
        cocktailDetailSelector(state, id)
    );
    return <Typography>{data.name}</Typography>;
});
```

Výpis kódu 4.2: Ukázka komponenty prezentačního modulu

4.1.3 Entitní modul

Entitní modul je speciální případ modulu, jež reprezentuje jednotlivé entity, které jsou získávány ze serverové části. Tento modul zodpovídá za zpracování požadavků na server a zpracování jeho odpovědí. Entitní modul není nijak závislý na prezentačním modulu, jedná se tedy o čistě aplikační vrstvu. Entitní modul dále obsahuje podřízené moduly (anglicky *submodules*), přičemž každý podřízený modul reprezentuje právě jeden typ entity. Zachycení jednotlivých entit v aplikačním stavu je znázorněno v příloze této práce (obrázek B.1 a B.2).

Funkcionalitu toho modulu využívají prezentační moduly, které stavy/data entit zobrazují uživateli. Adresářová struktura entitního modulu je znázorněna na následujícím obrázku.



Obrázek 4.3: Klientská část – ukázka struktury entitního modulu

4.1.4 Service worker

Technologie Service worker je v klientské části aplikace rozdělena na několik částí. Modul *service-worker* zodpovídá za registrační a aktualizační proces. Za generování finální verze Service workera zodpovídá knihovna Workbox.

4.1.4.1 Registrační a aktualizační cyklus

Proces registrace a aktualizace lze kategorizovat do tří skupin, které vznikly na základě experimentování a studií dokumentací. Rozdělení skupin je následující:

Service worker nebyl nikdy zaregistrován:

1. **Registrace** – v případě podpory technologie Service worker v prohlížeči vyčkej na načtení stránky a poté proveď registraci.
2. **Instalace** – vyčkej na nainstalování Service workera a poté zobraz uživateli hlášku – „Aplikace byla uložena“.
3. **Vyčkávání** – v pravidelných intervalech kontroluj, zda není k dispozici nová verze Service workera. Rovněž poslouchej na změny registračního objektu.

Service worker je již zaregistrován:

1. **Registrace/Instalace** – totožně jako v předchozím případě je provedena registrace.

2. **Rozhodnutí** – detekuj, zda je aplikace kontrolována jiným Service workerem, pokud ne, proveď instalaci, pokud ano, zobraz uživateli aktualizační výzvu. V daný moment je nový Service worker ve stavu *waiting*. V případě kladného rozhodnutí uživatele, dojde k okamžité aktivaci nového Service workera, přičemž předešlý Service worker přejde do stavu *redundant*. Všechny aktuální otevřené záložky[5] jsou okamžitě upozorněny na změnu aktuálního Service workera. Na upozornění reagují přenačtením aktuální stránky[6], které je téměr okamžité, protože v instalační fázi nový Service worker uložil potřebné statické podklady do aplikační keše.

Service worker čeká na aktivaci:

1. **Registrace** – registrační proces proběhne totožně jako v předchozích bodech. Tento bod je vyžadován pro získání registračního objektu.

2. **Rozhodnutí** – z registračního objektu detekujeme, že registrace eviduje čekajícího Service worker. Uživateli je následně zobrazena aktualizační výzva, která je totožná s výzvou v předchozím bodě.

4.1.4.2 Generování pravidel

Samotný soubor (*service-worker.js*), který modul registruje, je generován na základě příslušného konfiguračního souboru, jež definuje pravidla, podle kterých je finální Service worker vygenerován. Generování zajišťuje knihovna Workbox, konkrétně jeden z jejich balíčků – *workbox-webpack-plugin*, který spolupracuje s nástrojem Webpack.

Webpack při sestavování prochází jednotlivé moduly a transformuje je do podoby, se kterou umí prohlížeče pracovat (např. TypeScript na Javascript). Balíček *workbox-webpack-plugin* následně zahrne tyto výsledné statické soubory do svého seznamu zdrojů a aplikuje na ně strategii *pouze keš* (vizte kapitolu 2.2.1.1). Jednotlivé revize statických souborů jsou jednoznačně určeny jejich názvem, který je při sestavování zvolen pomocí příslušné hešovací funkce (anglicky *hash function*). Rovněž transformuje námi definovaná pravidla pro běhovou (dynamickou) keš do výsledného seznamu pravidel.

Výsledný soubor, který po registraci přebírá roli Service workera, tedy přerušuje síťové požadavky a aplikuje na ně námi zvolené kešovací strategie.

[5] rovněž okna a stránky

[6] pokud by nedošlo k přenačtení, došlo by k míšení starého a nového obsahu

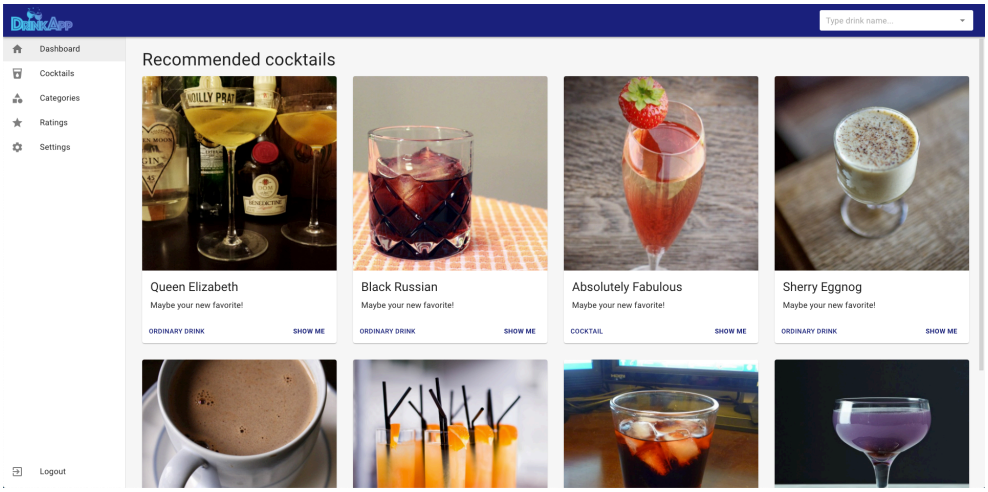
Následující výpis kódu zachycuje nastavení pro všechny požadavky směřující na serverovou část.

```
// config/transformWebpackConfig.js
...
{
    urlPattern: new RegExp(`^https://api.yourcocktails.club`),
    method: 'GET', // HTTP method
    handler: 'NetworkFirst', // strategy type
    options: {
        networkTimeoutSeconds: 6,
        cacheName: 'api',
        expiration: {
            // set lifetime of cached data for 14 days
            maxAgeSeconds: 60 * 60 * 24 * 14,
        },
    },
}
...
```

Výpis kódu 4.3: Ukázka zjednodušené konfigurace Workboxu

4.1.5 Výsledek

Klientská část naplňuje všechny funkční a nefunkční požadavky, které na ni byly kladeny (viz kapitola č. 3). Snímky obrazovky byly pořízené ze zařízení MacBook Pro a Samsung Galaxy S10e. Aplikace byla dále testována na zařízeních iPhone 8 (iOS 12.4.1) a iPhone X (iOS 13.4.1).



Obrázek 4.4: Úvodní obrazovka aplikace DrinkApp

4.2 Architektura Backendu

Serverová část webové aplikace je vytvořena v jazyce PHP, jejíž jádro tvoří Nette Framework, popsaný v sekci 1.2.1. Serverová část implementuje principy Rest API popsané v sekci 1.2. Architektura serverové části je tvořena modulovým systémem, přičemž jednotlivé moduly reprezentují jednotlivé verze rozhraní. Moduly jsou umístěné v adresáři *module*, přičemž každý modul obsahuje prezentační a aplikační vrstvu. Diagram tříd je k dispozici v příloze této práce (obrázek B.3).

Prezentační vrstva je tvořena sadou tříd, přičemž každá třída reprezentuje právě jeden zdroj. Jednotlivé třídy jsou pojmenovány podle vzoru *ZdrojPresenter*. Každá prezentační třída musí obsahovat postfix *Presenter*, tímto frameworku signalizuje, že se má daná třída zaregistrovat jako prezentační. Při přijetí požadavku je činnost delegována na příslušný model (aplikační vrstvu). Ten požadavek odbaví a vrátí výsledek zpět prezentační vrstvě, která data ve formátu JSON předá žadateli.

Směrování (anglicky *routing*) se ve frameworku Nette odvíjí podle pravidel definované v konfiguračním souboru. Vzhledem k využívání balíčku *ublaboo/api-router* je výchozí způsob směrování nahrazen. Balíček *ublaboo/api-router* generuje pravidla pro směrování na základě anotací metod v prezentační třídě. Z těchto anotací je dále, pomocí balíčku *ublaboo/api-docu*, automaticky generována dokumentace. Mezi další vlastnosti balíčku *ublaboo/api-router* je i automatické validace parametrů dle validačních pravidel specifikovaných v dané anotaci. Tato pravidla se rovněž promítnou v dokumentaci.

4.2.1 Autorizace

Vzhledem k povaze aplikace je pro většinu akcí (kromě registrace a přihlášení) vyžadovaná autorizace[7]. Při oddělení klientské a serverové části a bezstavovosti protokolu HTTP je autorizace řešena pomocí zasílání HTTP hlavičky *Authorization*. Hlavička obsahuje řetězec identifikující uživatele, tento řetězec se označuje jako přístupový token. Serverová část pro autentizaci využívá otevřený protokol OAuth, jehož principy jsou vysvětleny v kapitole 1.2.2.

Implementaci OAuth protokolu v rámci aplikace zajišťuje balíček *drahak/OAuth2*. Způsob ověřování příchozí autentizační hlavičky je pak implementována v modelu *AuthModel.php* metodou *verifyAccessToken*. Metoda je rovněž anotována a je tedy zahrnuta i v dokumentaci pro daný modul.

Požadavky týkající se vygenerování přístupového tokenu odbavuje prezentační třída *AuthnPresenter.php*, která rovněž odbavuje požadavky pro vygenerování přístupového tokenu na základě obnovovacího tokenu.

Pro vygenerování přístupového tokenu je využit způsob známý jako *Client credentials*, který pro vygenerování přístupového tokenu vyžaduje přihlašovací jméno a heslo. Životnost přístupových a obnovovacích tokenů je definovaná

[7] ověření, že žadatel má na vykonání dané akce příslušné oprávnění

v konfiguračním souboru *config.neon*. Všechny tokeny jsou po vygenerování uložené do persistentního uložiště.

4.2.2 Směrování

Při přijetí požadavku framework vyhledá příslušný modul a předá proces směrování na knihovnu *ublaboo/api-docu*, která vyhledá příslušnou třídu a její metodu na základě anotací. Anotace musí obsahovat absolutní cestu ke zdroji a HTTP metodu, se kterou lze příslušnou URL volat. Dále je možné specifikovat parametry v URL (včetně validačního pravidla zapsaného pomocí regulárního výrazu), stavové kódy, značky, ukázku výstupy a další.

```
/**
 * @ApiRoute(
 *    "/v1/users/<id>",
 *    method="GET",
 *    parameters={
 *        "id"={
 *          "requirement": "\d+",
 *          "type": "integer",
 *          "description": "User ID",
 *      }
 *    },
 *    response_codes={
 *      200="Success",
 *      400="Bad request",
 *      403={
 *          "403e1": "Invalid access token",
 *          "403e2": "User is not allowed to do this",
 *      },
 *      500="Internal server error"
 *    },
 *    section="Users"
 * )
 */
```

Výpis kódu 4.4: Serverová část – ukázka anotace pro zdroj /users/<id>

4.2.3 Zpracování požadavku – prezentační vrstva

Po nalezení příslušné metody prezentační třídy je proces zpracování tvořen pouhou delegací úkonu na model (viz ukázka 4.7). Tělo (anglicky *body*) je vždy obaleno klauzulí *try-catch*, důvodem je případné zachycení vyjimek (anglicky *exceptions*) z modelu.

```php
/**
 * Anotace zmíněná v předchozí sekci
 */
public function actionGetUser($id)
{
    try {
        $user = $this->authModel->getUserById($id);
        return new ApiResponse($this->formatPayload($user));
    } catch (\Exception $e) {
        return new ApiResponse($this->formatException($e));
    }
}
```

Výpis kódu 4.5: Serverová část – ukázka metody pro zdroj /users/<id>

Následně je výsledek/výjimka předána formátovací metodě, která provede transformaci na požadovaný tvar (vyextrahuje určité vlastnosti do podoby, jež vyžaduje konstruktor metody třídy ApiResponse). Viz následující ukázka těla odpovědi serveru v případě úspěšného vyřízení požadavku pro získání uživatele s identifikačním číslem 10.


```json
{
    "user_id":10,
    "username":"cferier9",
    "email":"cferier9@pagesperso-orange.fr",
    "first_name":"Carr",
    "last_name":"Ferier",
    "country":"Poland",
    "last_update": {
        "date":"2019-05-07 17:41:24.000000",
        "timezone_type":3,
        "timezone":"Europe Prague"
    },
    "role":"user"
}
```


Výpis kódu 4.6: Serverová část – ukázka těla odpovědi

V případě, kdy dojde k chybě, protože daný zdroj neexistuje či autentizační token je neplatný, pak je tělem odpovědi chybová zpráva (položka *message*) a nepovinný chybový kód (položka *code*), který slouží pro lepší identifikaci problému, který nastal při zpracování požadavku.

Pokud by součástí odpovědi nebyl specifický chybový kód, žadatel by na základě stavového kódu nemohl rozhodnout, zda chyba nastala, protože daný ověřovací token již není platný nebo pokud uživatel nemá dostatečné oprávnění. Při chybějícím doplňujícím kódu by tento problém bylo možné řešit pomocí zobrazení obecné chybové hlášky. Jednalo by se ale pouze o krátkodobé řešení, jehož nedostatky by se projevily například při lokalizování aplikace.

4.2.4 Zpracování požadavku – aplikační vrstva

Aplikační vrstva je tvořena modely, které jsou v kontextu Nette Frameworku zodpovědné především za komunikaci s databází. Závislosti se v modelech předávají pomocí *Dependency Injection*. Z tohoto důvodu je nutné příslušný model zaregistrovat v konfiguračním souboru *config.neon*, framework se pak postará o automatické předávání závislostí. U prezentačních tříd je předávání řešeno automaticky.

```
public function getUserById($id)
{
    $user = $this->getUserByToken();
    if ($id === NULL) {
        return $user;
    }

    $requestedUser = $this->userModel->getUserById($id);

    if (!$requestedUser) {
        throw new \NoDataException('User does not exist!');
    }

    if ($requestedUser->id !== $user->id
        && $user->role !== Roles::ADMIN) {
        throw new \AccessDeniedException();
    }

    return $requestedUser;
}
```

Výpis kódu 4.7: Serverová část – ukázka metody modelu

4.2.5 Implementované rozhraní

Následující tabulka shrnuje rozhraní, které serverová část nabízí. U zdrojů vyžadující parametr pro identifikování uživatele byla vytvořena alternativa i bez toho parametru. Důvodem je fakt, že identifikující parametr je použit pouze ve chvíli, kdy jeden uživatel zaměňuje informace jiného uživatele, přičemž tuto možnost má pouze administrátor. Tento parametr tedy není nutný, pokud uživatel edituje sám sebe, protože je systémem jednoznačně identifikován na základě autentizačního tokenu v hlavičce požadavku. Seznam všech cest ke zdrojům, které serverová část implementuje, je sepsán v tabulkách 4.1, 4.2 a 4.3.

Tabulka 4.1: Serverová část – přehled cest pro zdroj uživatel

Cesty ke zdroji uživatel		
/v1/users		GET, POST
GET	získání všech uživatelů	
POST	vytvoření nového uživatele	
/v1/users/<id>		GET, POST
GET	získání konkrétního uživatelů	
UPDATE	aktualizace uživatelského profilu	
/v1/users/<id>/cocktails/ratings		GET
GET	získání všech hodnocení daného uživatele	
/v1/users/<id>/cocktails/ratings		GET
GET	získání všech hodnocení daného uživatele	
/v1/users/<id>/cocktails/<cocktailId>/ratings		
GET	získání uživatelského hodnocení pro daný koktejl	
DELETE	odstranění uživatelského hodnocení pro daný koktejl	
UPDATE	úprava uživatelského hodnocení pro daný koktejl	
POST	vytvoření uživatelského hodnocení pro daný koktejl	
/v1/users/<id>/cocktails/<cocktailId>/ratings		GET
GET	získání všech hodnocení daného uživatele	
/v1/users/<id>/cocktails/<count>		GET
GET	získání zadaného počtu doporučených koktejlů	

Tabulka 4.2: Serverová část – přehled cest pro zdroj koktejl

Cesty ke zdroji koktejl		
/v1/cocktails		GET
GET	získání všech koktejlů v databázi	
/v1/cocktails/<id>		GET
GET	získání detailu konkrétního koktejlu	
/v1/cocktails/<id>/ratings		GET
GET	souhrnný přehled hodnocení daného koktejlu	

Tabulka 4.3: Serverová část – přehled cest pro zdroj kategorie

Cesty ke zdroji kategorie		
/v1/categories		GET
GET	získání všech kategorií v databázi včetně celkového počtu koktejlů	
/v1/categories/<id>		GET
GET	získání informací o konkrétní kategorii	
/v1/cocktails/<id>/cocktails		GET
GET	získání všech koktejlů spadající pod danou kategorii	

Kapitola 5

Testování

Následující kapitola popisuje principy, pomocí kterých je testována klientská a serverová část aplikace. Obě části byly testovány pomocí automatických testů, přičemž klientská část byla podrobena i manuálnímu testování. Veškeré chyby či selhání klientské části aplikace jsou nahlašovány do cloudového uložiště aplikace Sentry a chyby způsobené na serverové části jsou zaznamenávány pomocí nástroje Tracy.

5.1 Klientská část

Testování klientské části probíhá pomocí nástroje Jest, kterým je testována aplikační vrstva aplikace. Jedná se tedy zejména o části, kterou jsou zodpovědné za testování funkcí, jež přistupují nebo přímo upravují aplikační stav. Tyto funkce jsou krokovány[8] a při pozastavení kontrolovány. Automatické testy pokrývají oblasti, které byly definovány ve funkčních požadavcích (viz kapitola 3). Aplikace je složena z 12 testovaných oblastí, přičemž každá oblast testuje několik svých podčástí. Celkem aplikace zahrnuje 40 testovacích případů.

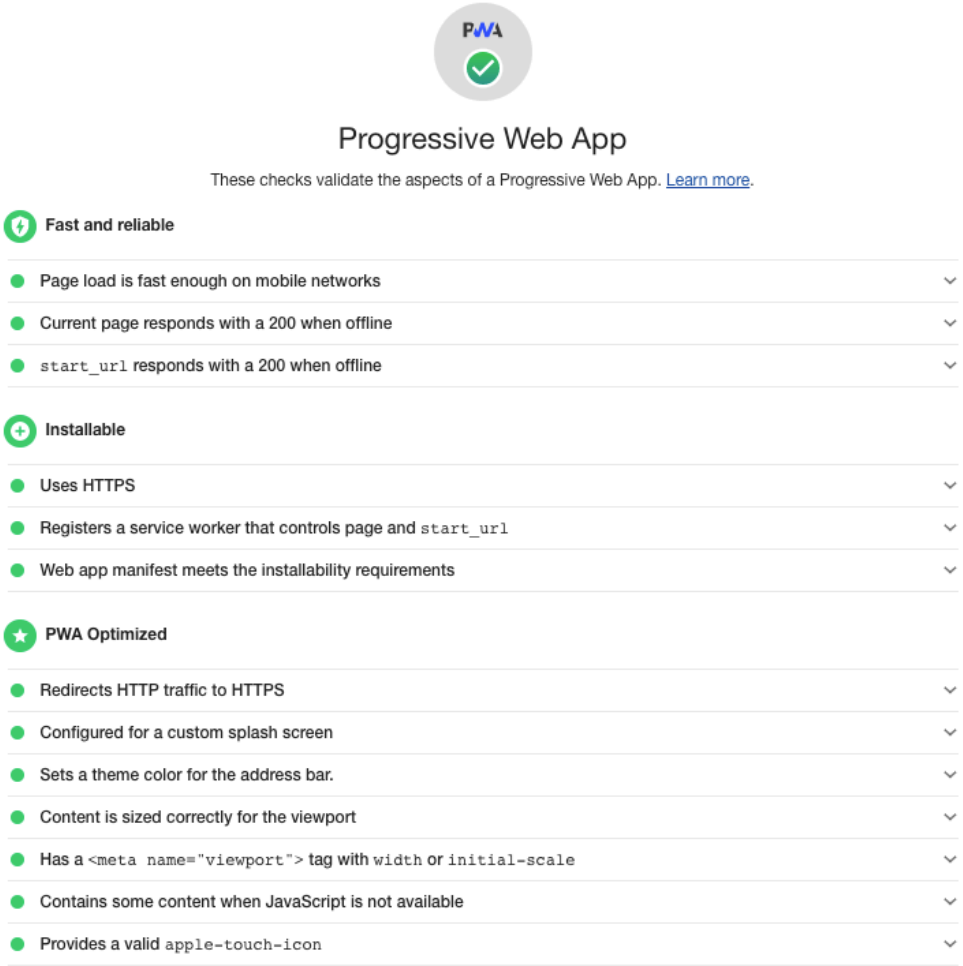
5.1.1 Test PWA

Progresivita webové aplikace byla testována pomocí manuálního testování a dále pomocí nástroje Lighthouse. Manuální testování prověřovalo body stanovené v kapitole 2.1. Součástí manuálního testování bylo rovněž prověření funkčnosti aktualizací a používání aplikace bez připojení k internetu. Manuální testování probíhalo na následujících zařízení:

[8] jedná se o generátory – funkce jež je možné v daném místě pozastavit pomocí klíčového slova *yield*

- MacBook Pro 2018 (Google Chrome 83.0.4103.61),
- iPhone X (iOS 12.4.1),
- iPhone 8 (iOS 12.4.1),
- Samsung Galaxy S10e (Google Chrome 83.0.4103.83).

Automatické testování probíhalo pomocí nástroje Lighthouse od společnosti Google, který provádí kontrolu stěžejních bodů konceptu PWA. Nástroj Lighthouse převážně ověřuje technické aspekty konceptu PWA [42]. Proto bylo prováděno i manuální testování.



Obrázek 5.1: Testování aplikace pomocí nástroje Lighthouse

5.2 Serverová část

Testování serverové části probíhá pomocí automatizovaných testů, které jsou vykonávány pomocí nástroje Nette Tester. Ten je doporučován pro aplikace postavené na Nette Frameworku. Testy prověřují funkčnost jednotlivých modelů, přičemž celkově je aplikace podrobena 20 testům. Testy se automaticky spouštějí před přidáním revize (do verzovacího systému Git) a před samotným sestavením aplikace na serveru. Veškeré chybové stavy jsou zaznamenávány knihovnou Tracy.

5.3 Uživatelské testování

Principem uživatelského testování je pozorování uživatelů aplikace, pomocí kterého lze odhalit chyby, které při vývoji mohly být přehlédnuty. Tento druh testování je vhodné aplikovat již při vývoji aplikace, čímž lze předcházet budoucím problémům. [43]

Do testování bylo zapojeno 5 účastníků, kteří byli po dokončení scénářů (tabulka 5.1) požádáni o zpětnou vazvu.

Tabulka 5.1: Testovací scénáře

	Název	**Popis**
T1	**Registrace/Přihlášení**	Proveďte registraci v systému, následně se odhlašte a proveďte přihlášení
T2	**Změna osobních údajů**	Proveďte změnu osobních údajů a následně proveďte změnu hesla
T3	**Vyhledávání a hodnocení**	Vyhledejte koktejl jménem „Ace“, kterému následně přiřaďte libovolné hodnocení
T4	**Historie**	Vyhledejte tabulku zachycující vaše dosavadní hodnocení a aplikujte filtr, který zobrazí koktejly ohodnocené 5 hvězdičkami
T5	**Instalace**	Proveďte instalaci aplikace a následně proveďte stažení všech příbuzných dat

5.3.1 Průběh testování

Následující část popisuje průběh testování jednotlivých scénářů, který rovněž obsahuje uživatelskou odezvu.

- **T1 – Registrace/Přihlášení** Všemi uživateli zvládnut ve velmi krátkém čase (pod 30 sekund), přičemž byla kladně hodnocena komponenta pro výběr země, ze které uživatel pochází. Rovněž uživatelé kladně hodnotili automatické přihlášení po dokončení registrace.
- **T2 – Změna osobních údajů a hesla** Všemi uživateli zvládnut okamžitě. Tento fakt byl uživateli vysvětlen tím, že editace osobních údajů a hesla byla logicky umístěna pod položkou „Nastavení".
- **T3 – Vyhledávání a hodnocení** Stejně jako předchozí scénář, byl i tento zvládnut bez problémů. Pouze jeden účastník zhruba 5 sekund nemohl najít pole pro vyhledávání. Hodnocení pomocí hvězdiček (které se zvětšují po najetí myši) bylo dvěma uživateli označeno jako interaktivní.
- **T4 – Historie** V tomto scénáři měli uchazeči připraveni účet, který obsahoval vyšší množství hodnocení (více než 30). Většina účastníků (4 z 5) neměli s úkolem problém, pouze jeden uživatel označil ikonu pro filtrování za nevhodnou.
- **T5 – Instalace** Instalační scénář byl nejnáročnějším scénářem ze všech uvedených. Dva uživatelé se systémem iOS (celkově 3 z 5) nevěděli, jak aplikaci přidat na plochu. Ostatní účastníci se systémem Android neměli s přidáním a následným stažením dat problém. Toto chování bylo pravděpodobně zapříčiněno tím, že systém iOS nevyzývá uživatele k přidání na plochu.

5.3.2 Shrnutí testování

Aplikace byla uživateli velmi dobře přijata a zpracování výše uvedených scénářů probíhalo bez značných problémů. Výjimku tvořil pouze scénář, týkající se instalace aplikace, který byl pro uživatelé systému iOS nejednoznačný. Na základě této uživatelské vazby byl do aplikace implementován určený speciální dialog, který pro systém iOS znázorňuje, jak provést přidání na plochu (instalaci).

Kapitola 6

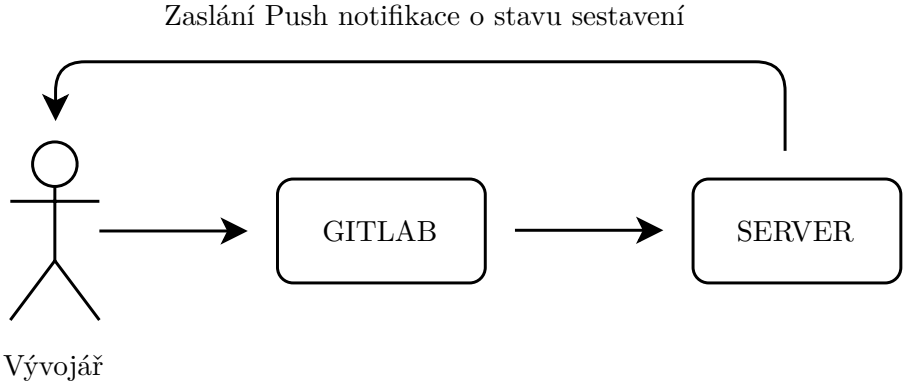
Nasazení aplikace

Následující kapitola popisuje nasazení aplikace do reálného provozu.

6.1 Verzování aplikace

Klientská a serverová část aplikace byla po celý čas vývoje verzována pomocí verzovacího systému Git jako privátní repozitář ve službu Gitlab.

Při přidání změn do hlavní vývojové větve (anglicky *master branch*) dojde k odeslání požadavku na server, který automaticky provede sestavení nové verze. Při přijmutí požadavku je před sestavením ověřen původ zprávy a při neautorizovaném požadavku nedojde k sestavení. Serverový skript detekuje o jaký typ projektu se jedná (React aplikace, Nette aplikace, statická stránka) z čehož vyvodí příslušné příkazy pro sestavení. Skript rovněž rozlišuje, na jaké větvi ve verzovacím systému se projekt nachází, což má za následek automatické nasazování vývojové a produkční verze projektu.



Obrázek 6.1: Proces verzování a sestavení aplikace

6.2 Doména a CDN

Za účelem zveřejnění aplikace byla zakoupena doména yourcocktails.club. Doméně byly po zakoupení přenastaveny DNS servery na DNS servery společnosti Cloudflare. Následně byly nakonfigurovány DNS záznamy (typu A, AAAA a MX) na produkční server. Cloudflare funguje jako reverzní proxy a tudíž maskuje koncový server (nelze dohledat původní IP adresu serveru). Cloudflare rovněž funguje jako CDN.

6.3 Server

Jádro produkčního serveru tvoří operační systém Linux Debian verze 10. Serverová a klientská část je servírována webových serverem Apache. Komunikace s klientskou částí (yourcocktails.club) a serverovou částí (api.yourcocktails.club) je servírovaná přes protokol HTTPS, přičemž certifikát byl vydán certifikační autoritou *Let's Encrypt*. Server periodicky (1x za tři měsíce) vyžádá nový certifikát a provede jeho instalaci.

Závěr

Tato bakalářská práce se primárně zabývala vytvořením progresivní webové aplikace, která svým uživatelům doporučuje koktejly na základě aktivního kolaborativního filtrování. Mezi cíle práce rovněž patřil i návrh a realizace serverové části, přičemž obě měly být následně uvedeny do provozu pro širší veřejnost. Práce všechny výše uvedené body splňuje a o některé části jej rozšiřuje.

Progresivní webová aplikace byla vytvořena pomocí moderních technologií a reflektuje aktuální designerské trendy. Jádrem aplikace je knihovna React (Facebook) a její struktura je modulární. Modifikace aplikační stavu je zajišťováno knihovnou Redux. Vedlejší efekty způsobené mutací aplikačního stavu jsou zpracovávány v ságách – knihovnou Redux Saga. Kešování statických a dynamických zdrojů zajišťuje knihovna Workbox (Google). Uživatelské rozhraní je založeno na Materiálním designu (Google), díky čemuž v uživateli evokuje pocit nativní aplikace. Aplikaci je možné si nainstalovat a používat ji bez připojení k internetu, finální velikost (bez obrázků koktejlů) činí necelých 5 MB. Aktualizace jsou v rámci aplikace řešeny na pozadí, tudíž uživatel vždy používá nejaktuálnější verzi. Případné zaregistrované chyby, které u uživatele nastaly, jsou okamžitě společně s chybovým protokolem odeslány do cloudového uložiště aplikace Sentry, na základě kterých jsou řešeny opravy. Uživatelská návštěvnost je měřena pomocí Google Analytics, kromě návštěvnosti je zaznamenána i doba od ztracení připojení k internetu, která poskytuje zpětnou vazbu o využívání aplikace bez připojení k internetu.

Serverová část implementuje principy Rest API a její jádro tvoří český Nette Framework. Pro autorizaci uživatelů je využíván protokol OAuth 2.0. Persistentním uložištěm dat je MySQL databáze, se kterou serverová část pracuje.

Klientská a serverová část aplikace byla nasazena na Linuxový server, který kromě servírování obsahu i automaticky sestavuje nové verze aplikace ze zdrojových kódu přebíraných ze služby Gitlab. Pro testovací účely byla rovněž vytvořena i testovací subdoména, na které bude při dalším vývoji probíhat tes-

tování. Celá doména (`https://yourcocktails.club`) je pod záštitou veřejné reverzní proxy (služby Cloudflare). Server periodicky nasazuje bezpečnostní certifikáty od certifikační autority *Let's Encrypt*.

Vize do budoucna

Při tvorbě bakalářského projektu byl již od počátku brán zřetel na znovu využitelnost a možný pohodlný budoucí rozvoj. Klientská část od prvotní verze podporuje vícejazyčné prostředí, které je generováno z překladových tabulek umístěných ve službě Google Sheets, což má za následek pohodlný způsob práce pro překladatele. Aplikace rovněž interně řeší i proces aktualizování.

Aplikaci by do budoucna bylo vhodné rozšířit o následující funkcionality:

1. **Oblíbené koktejly** – uživatel si může vést vlastní seznam oblíbených koktejlů a ten sdílet s ostatními.
2. **Uživatelské profily** – uživatelé by mohli procházet profily ostatních uživatelů a zobrazovat si jejich oblíbené nápoje a základní informace o nich.
3. **Komentování koktejlů** – uživatelé by komentovat jednotlivé koktejly a číst komentáře ostatních
4. **Plánovací kalendář** – uživatel by si mohl vytvářet v kalendáři události a k ním si ukládat příslušné koktejly. Uživatel poté bude s časovým předstihem informován pomocí *Push notifikací*.

Přínos práce

V práci je rozebrán koncept PWA, který je porovnán s nativními aplikacemi. Práce dále řeší podporu napříč různými prohlížeči a zachycuje případové studie velkých společností (Twitter, Pinterest), které koncept PWA implementovali. Obsah práce rovněž může sloužit jako orientační bod v nepřeberném množství webových technologií a čtenáři tak doporučit řadu moderních knihoven a principů, které lze uplatit nejen v konceptu PWA.

Bibliografie

1. *What is Frontend? What is Backend?* [online]. Frontend GmbH [cit. 2020-04-09]. Dostupné z: https://www.frontend-gmbh.de/en/blog/what-is-frontend-what-is-backend/.
2. *React – A JavaScript library for building user interfaces* [online]. Facebook, 2018 [cit. 2020-04-14]. Dostupné z: https://reactjs.org/.
3. HÁMORI, Ferenc. *The History of React.js on a Timeline* [online]. RisingStack Engineering, 2020 [cit. 2020-04-18]. Dostupné z: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/.
4. *Stack Overflow Developer Survey 2019* [online]. Stack Overflow, 2019 [cit. 2020-04-19]. Dostupné z: https://insights.stackoverflow.com/survey/2019.
5. *Learning React Native* [online]. OREILLY, 2020 [cit. 2020-04-19]. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html.
6. ABRAMOV, Dan. *Redux* [online]. 2020 [cit. 2020-05-06]. Dostupné z: https://redux.js.org/introduction/three-principles.
7. *Introduction - Material Design* [online]. Google, 2016 [cit. 2020-04-09]. Dostupné z: https://material.io/design/introduction.
8. AVRAM, Abel. *Google Introduces Material Design at Google I/O* [online]. InfoQ, 2014 [cit. 2020-04-13]. Dostupné z: https://www.infoq.com/news/2014/06/google-material-design-android/.
9. MALÝ, Martin. REST: architektura pro webové API. *Zdroják* [online]. 2009 [cit. 2020-04-20]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.
10. NETTE FOUNDATION. *Pohodlný a bezpečný vývoj webových aplikací v PHP* [online]. 2020 [cit. 2020-04-21]. Dostupné z: https://nette.org/cs/.

11. NETTE FOUNDATION. *Životní cyklus aplikace* [online]. 2020 [cit. 2020-04-22]. Dostupné z: https://nette.org/cs/3.0/request-lifecycle.

12. IETF. *The OAuth 2.0 Authorization Framework* [online]. 2012 [cit. 2020-05-11]. Dostupné z: https://tools.ietf.org/html/rfc6749.

13. RICCI, Francesco; ROKACH, Lior; SHAPIRA, Bracha. Recommender Systems Handbook. In: 2010, sv. 1-35, s. 1–35. Dostupné z DOI: 10.1007/978-0-387-85820-3_1.

14. MACDONALD, Jon. 21 Ecommerce Product Recommendations Best Practices. *The Good* [online]. 2019 [cit. 2020-04-25]. Dostupné z: https://thegood.com/insights/ecommerce-product-recommendation/.

15. NORWAY GRANTS. *Pearsonův korelační koeficient* [online]. 2019 [cit. 2020-04-26]. Dostupné z: https://mathstat.econ.muni.cz/media/12657/pear_cor.pdf.

16. *Pearson Product-Moment Correlation* [online]. Laerd Statistics, 2018 [cit. 2020-04-28]. Dostupné z: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php.

17. LEPAGE, Pete. *What are Progressive Web Apps?* [online]. 2020 [cit. 2020-04-02]. Dostupné z: https://web.dev/what-are-pwas/.

18. GRIGSBY, Jason. *Progressive Web Apps*. 28. vyd. New York: A Book Apart, 2018. ISBN 978-1-937557-72-0.

19. RICHARD, Sam. *Responsive Web Design Basics* [online]. 2019 [cit. 2020-04-07]. Dostupné z: https://developers.google.com/web/fundamentals/design-and-ux/responsive#css-media-queries.

20. DEVERIA, Alexis. *Can I use - Media Queries* [online]. 2020 [cit. 2020-05-02]. Dostupné z: https://caniuse.com/#feat=css-mediaqueries.

21. OSMANI, Addy. *The App Shell Model* [online]. Google, 2019 [cit. 2020-04-08]. Dostupné z: https://developers.google.com/web/fundamentals/architecture/app-shell.

22. KEITH, Jeremy. *Adactio*. What is a Progressive Web App? [online]. 2017 [cit. 2020-04-04]. Dostupné z: https://adactio.com/journal/13098.

23. GAUNT, Matt. *Service workers: an Introduction | Web Fundamentals* [online]. Google, 2019 [cit. 2020-04-27]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers.

24. ARCHIBALD, Jake. *The Offline Cookbook* [online]. Google, 2019 [cit. 2020-04-07]. Dostupné z: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook#stale-while-revalidate.

25. MOZZILLA. *Service worker API* [online]. 2020 [cit. 2020-04-05]. Dostupné z: https://developer.mozilla.org/en-US/online/Web/API/Service_Worker_API.

26. ADERINOKUN, Ire. *The Service worker Lifecycle* [online]. bitsofcode, 2019 [cit. 2020-05-10]. Dostupné z: https://bitsofco.de/the-service-worker-lifecycle/.

27. DEVERIA, Alexis. *Can I use - Service worker* [online]. 2020 [cit. 2020-05-07]. Dostupné z: https://caniuse.com/#feat=serviceworkers.

28. DEVERIA, Alexis. *Can I use - Web App Manifest* [online]. 2020 [cit. 2020-05-04]. Dostupné z: https://caniuse.com/#feat=web-app-manifest.

29. WILLIAMS, Owen. *Apple Is Trying to Kill Web Technology* [online]. OneZero, 2019 [cit. 2020-05-14]. Dostupné z: https://onezero.medium.com/apple-is-trying-to-kill-web-technology-a274237c174d.

30. INC, Apple. *App Store Review Guidelines* [online]. 2020 [cit. 2020-04-26]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/#third-party-software.

31. DEVERIA, Alexis. *Can I use - Push API* [online]. 2020 [cit. 2020-04-26]. Dostupné z: https://caniuse.com/#feat=mdn-api_pushevent.

32. DEVERIA, Alexis. *Can I use - Push API* [online]. 2020 [cit. 2020-04-26]. Dostupné z: https://caniuse.com/#feat=serviceworkers.

33. *Safari HTML Reference* [online]. Apple Inc., 2014 [cit. 2020-05-06]. Dostupné z: https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariHTMLRef/Articles/MetaTags.html.

34. FIRTMAN, Maximiliano. *Don't use iOS meta tags irresponsibly in your Progressive Web Apps* [online]. Medium, 2017 [cit. 2020-04-28]. Dostupné z: https://medium.com/@firt/dont-use-ios-web-app-meta-tag-irresponsibly-in-your-progressive-web-apps-85d70f4438cb.

35. OSMANI, Addy. *2016 Benchmarks, Budgets, and Trends—North America* [online]. Dev Channel, 2016 [cit. 2020-05-01]. Dostupné z: https://contentmarketinginstitute.com/wp-content/uploads/2015/09/2016_B2B_Report_Final.pdf.

36. OSMANI, Addy. *A Pinterest Progressive Web App Performance Case Study* [online]. Dev Channel, 2017 [cit. 2020-04-29]. Dostupné z: https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154.

37. What is Twitter and why should you use it? *AHRC* [online] [cit. 2020-05-03]. Dostupné z: https://esrc.ukri.org/research/impact-toolkit/social-media/twitter/what-is-twitter/.

38. *Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage* [online]. Google, 2017 [cit. 2020-04-28]. Dostupné z: `https://developers.google.com/web/showcase/2017/twitter`.

39. GALLAGHER, Nicolas. *How we built Twitter Lite* [online]. Twitter, 2017 [cit. 2020-05-02]. Dostupné z: `https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html`.

40. RUSSELL, Alex. *Why Are App Install Banners Still A Thing?* [online]. Dev Channel, 2017 [cit. 2020-05-04]. Dostupné z: `https://medium.com/dev-channel/why-are-app-install-banners-still-a-thing-18f3952d349a`.

41. MLEJNEK, Jiří. *Návrh softwarových systémů* [online]. 2020 [cit. 2020-04-26]. Dostupné z: `https://moodle-vyuka.cvut.cz/pluginfile.php/225511/mod_resource/content/4/05.prednaska.pdf`.

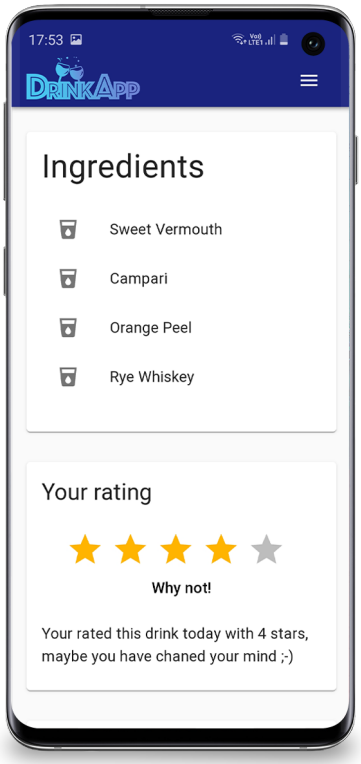
42. *Lighthouse PWA Analysis Tool | Web | Google Developers* [online]. Google, 2010 [cit. 2020-05-16]. Dostupné z: `https://developers.google.com/web/ilt/pwa/lighthouse-pwa-analysis-tool`.

43. KRUG, Steve. *Nenuťte uživatele přemýšlet!: praktický průvodce testováním a opravou chyb použitelnost webu* [online]. Computer Press, 2010 [cit. 2020-05-20].

Příloha A

Ukázka výsledné aplikace

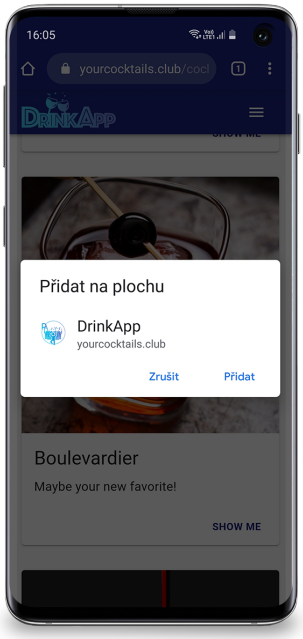
A.1.1: Detail koktejlu



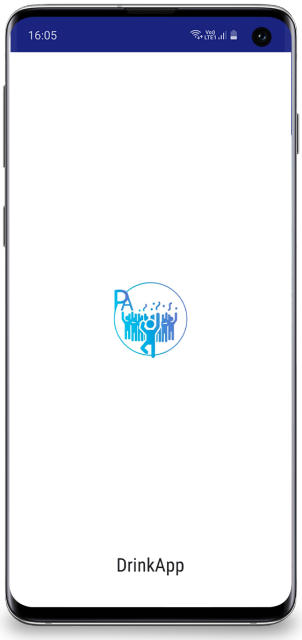
A.1.2: Ingredience koktejlu

Obrázek A.1: Detail koktejlu – mobilní telefon

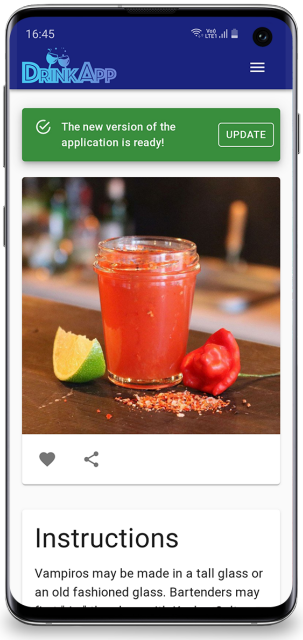
A.2.1: Notifikační lišta



A.2.2: Přidání aplikace

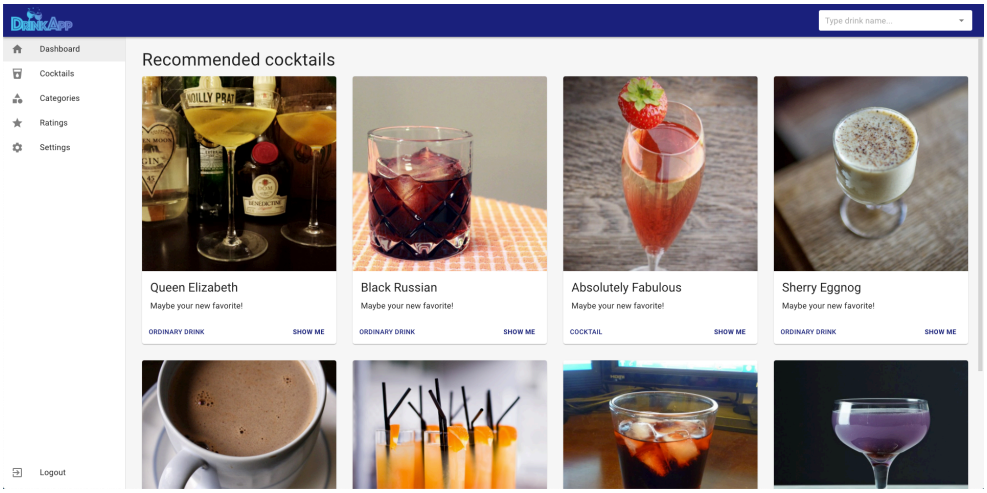


A.2.3: Načítací obrazovka

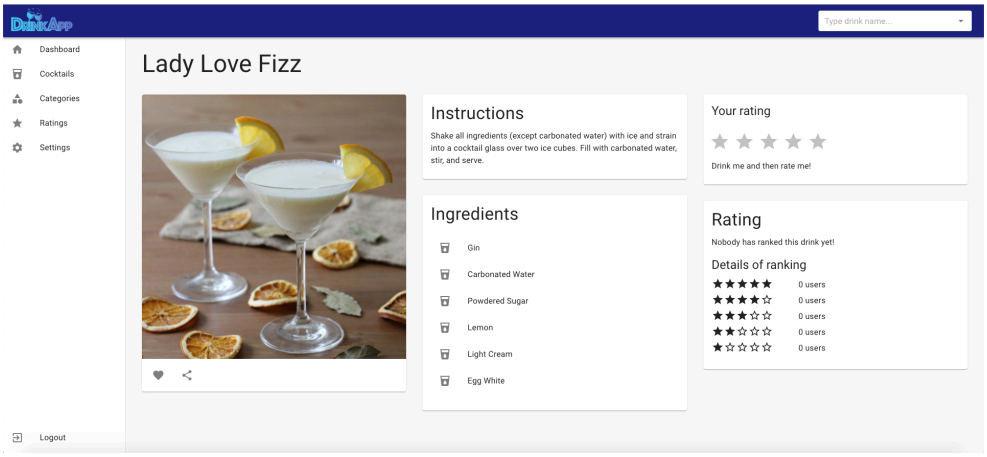


A.2.4: Výzva k aktualizaci

Obrázek A.2: Instalace a spuštění aplikace DrinkApp – mobilní telefon

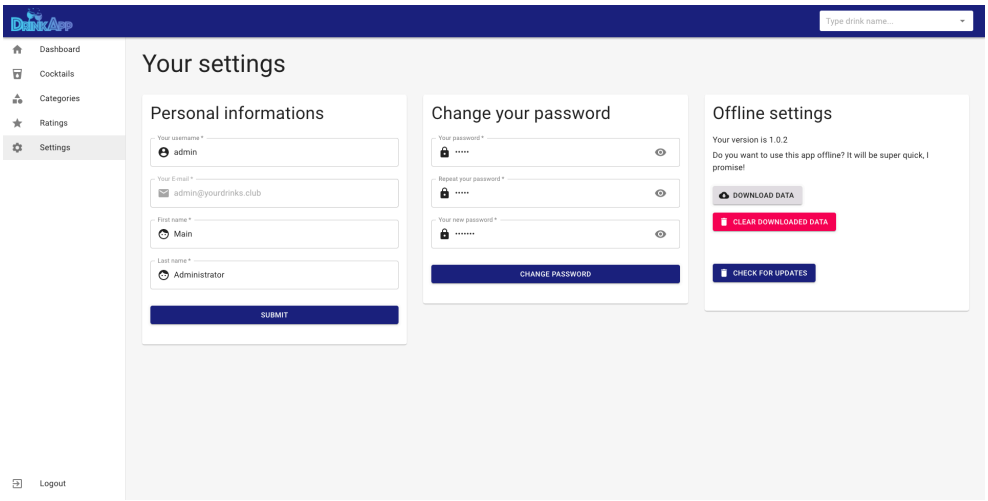


Obrázek A.3: Úvodní obrazovka aplikace DrinkApp – počítač



Obrázek A.4: Detail koktejlu v aplikaci DrinkApp – počítač

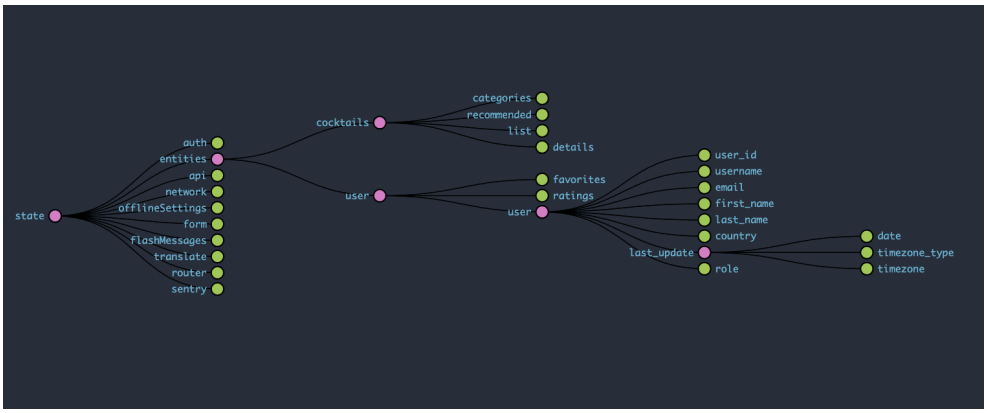
Obrázek A.5: Tabulka hodnocení koktejlů v aplikace DrinkApp – počítač



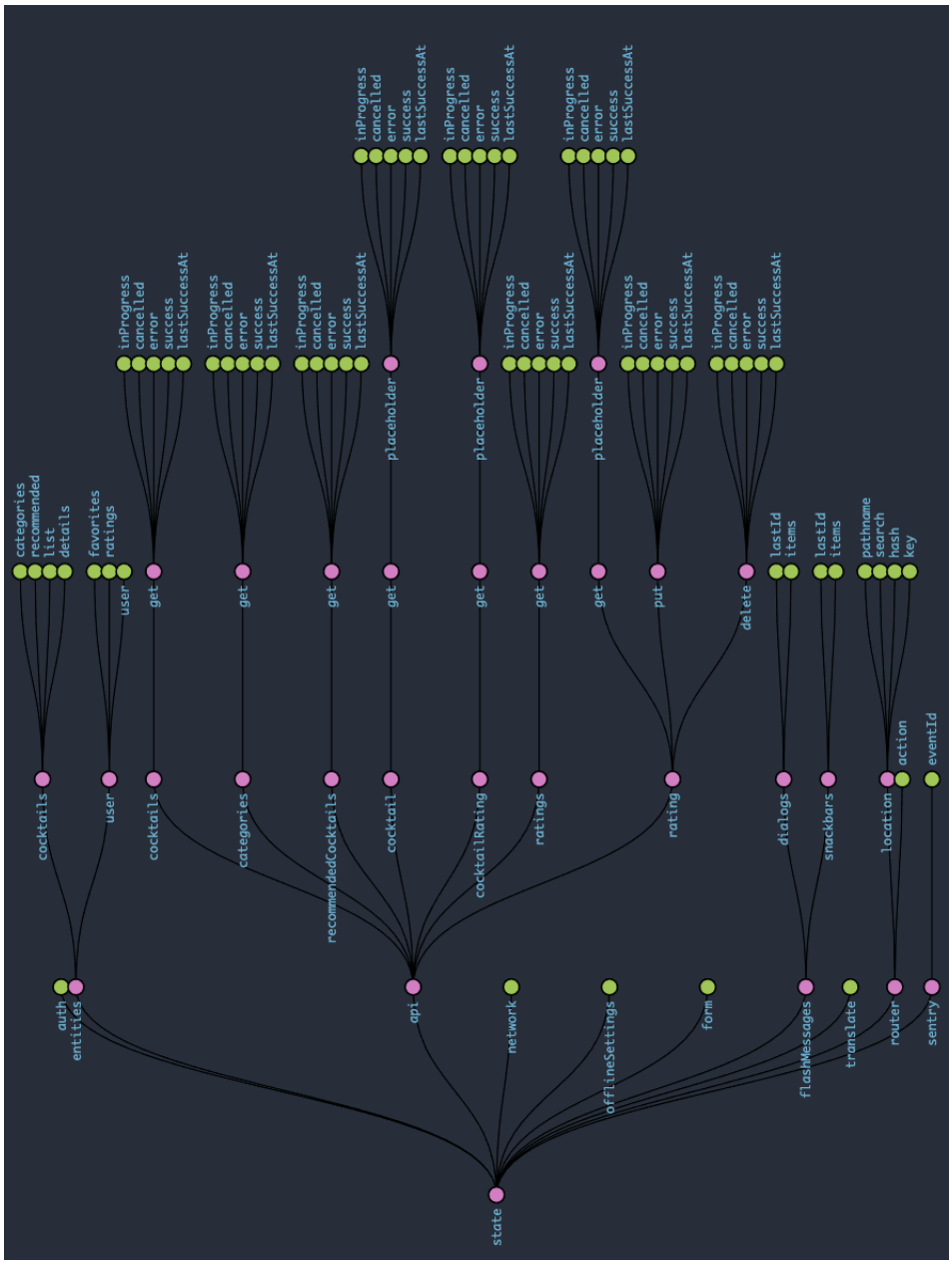
Obrázek A.6: Nastavení aplikace DrinkApp – počítač

Příloha B

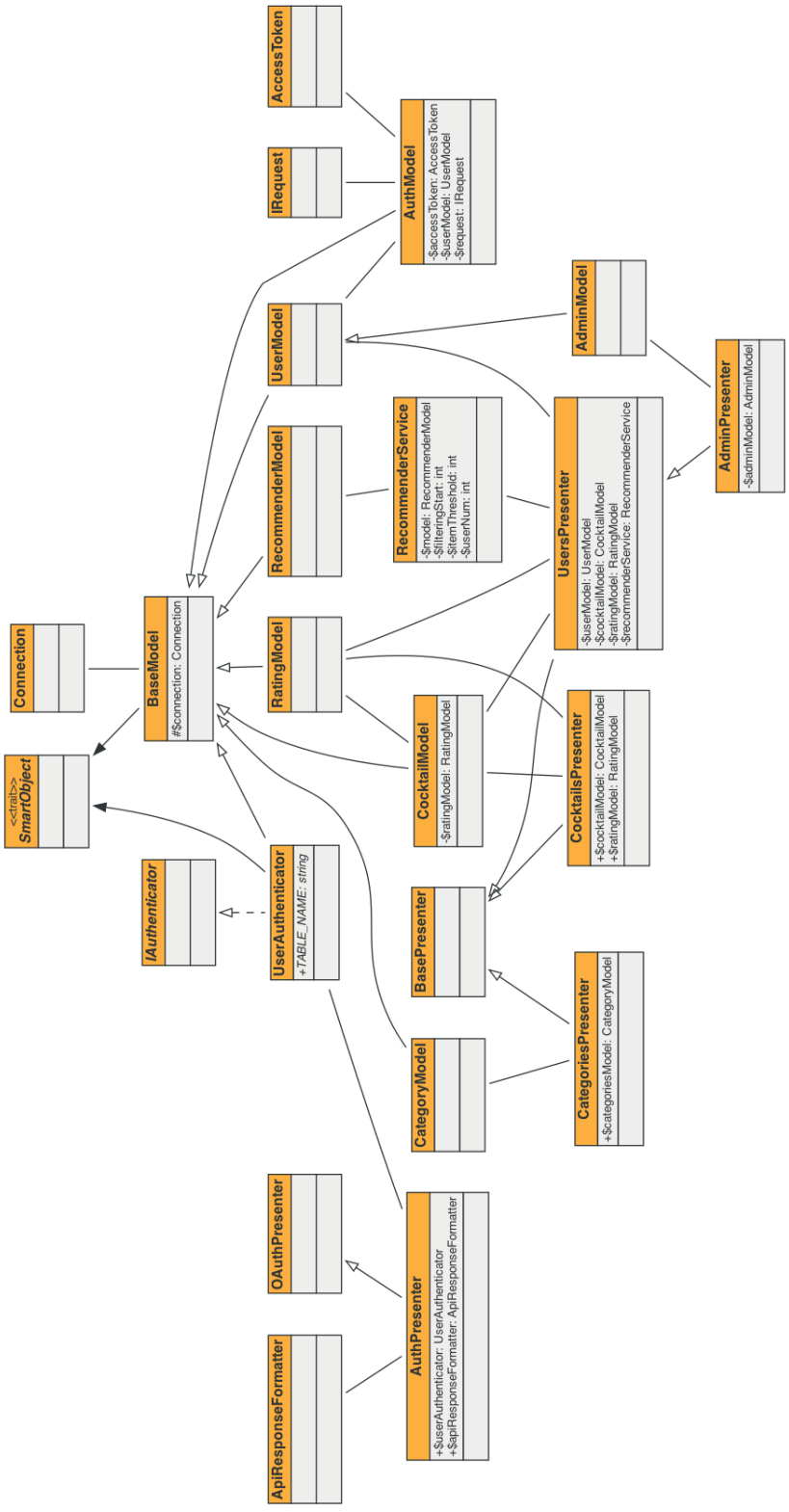
Diagramy a vizualizace



Obrázek B.1: Aplikační stav klientské části (vizualizace entit)



Obrázek B.2: Aplikační stav klientské části (vizualizace API požadavků)



Obrázek B.3: Diagram tříd serverové části

Příloha C

Podpora webových prohlížečů

Tabulka C.1: Kompletní podpora technologie Service worker v prohlížečích [27]

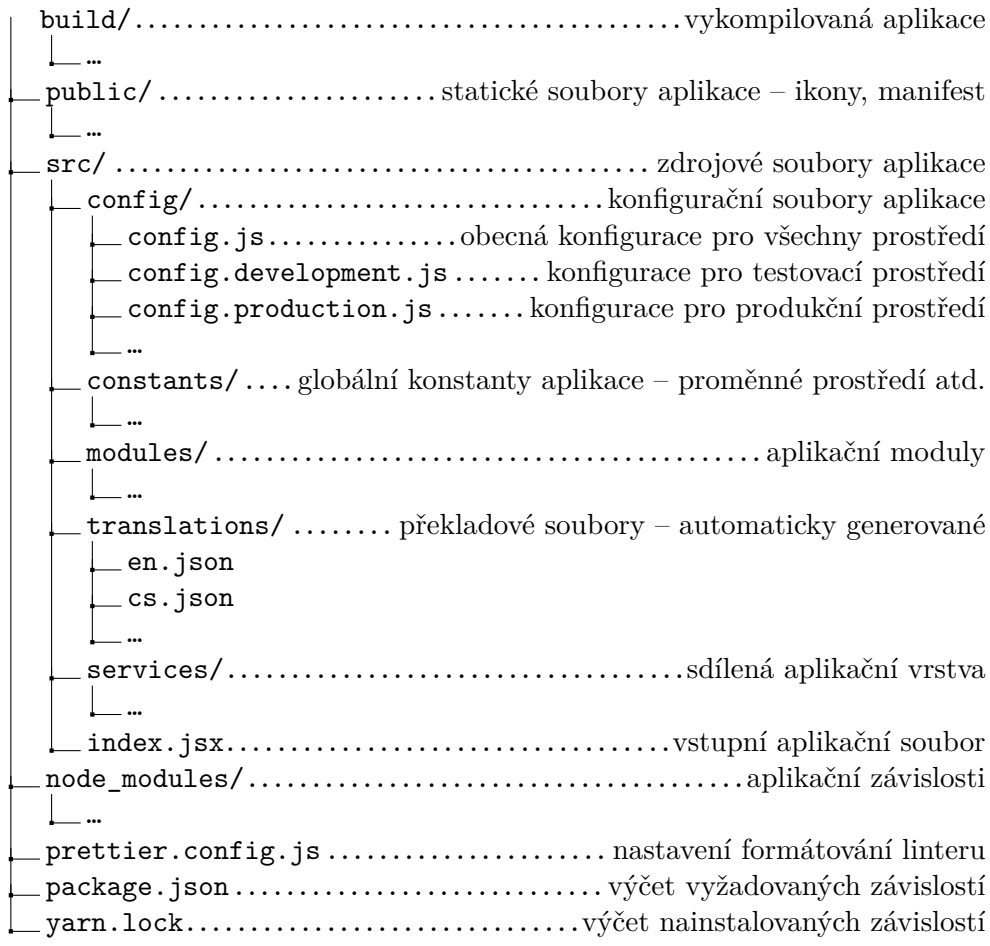
Prohlížeč	**Zastoupení**	
	Podporované	Celkem
Chrome – Android (od verze 81)	34.16 %	34.16 %
Chrome (od verze 45)	30.06 %	30.35 %
iOS Safari (od verze 11.4)	12.83 %	13.63 %
Firefox (od verze 44)	4.07 %	4.37 %
Safari (od verze 11.1)	3.71 %	4.04 %
Samsung Internet (od verze 4)	3.53 %	3.53 %
Edge (od verze 17)	2.05 %	2.12 %
UC Browser (od verze 12.12)	1.99 %	1.99 %
Opera (od verze 32)	0.54 %	0.66 %
Firefox – Android (od verze 68)	0.22 %	0.22 %
Internet Explorer	0.00 %	2.05 %
Android Browser	0.00 %	0.28 %
QQ Browser	0.00 %	0.23 %
KaiOS Browser	0.00 %	0.17 %
Opera Mobile	0.00 %	0.01 %
Ostatní prohlížeče	0.00 %	2.18 %
Celkem	**93.17%**	**97.82 %** (100 %)

Tabulka C.2: Kompletní podpora technologie Web App Manifest v prohlížečích [28]

Prohlížeč	**Zastoupení**	
	Podporované	Celkem
Chrome – Android (od verze 81)	34.16 %	34.16 %
Chrome (od verze 73)	28.58 %	30.35 %
iOS Safari	0.00 %	13.63 %
Firefox	0.00 %	4.37 %
Safari	0.00 %	4.04 %
Samsung Internet	0.00 %	3.53 %
Edge	0.00 %	2.12 %
IE	0.00 %	2.05 %
UC Browser – Android	0.00 %	1.99 %
Opera	0.00 %	0.66 %
Android Browser	0.00 %	0.28 %
QQ Browser	0.00 %	0.23 %
Firefox – Android	0.00 %	0.22 %
KaiOS Browser	0.00 %	0.17 %
Opera Mobile	0.00 %	0.01 %
Ostatní prohlížeče	0.00 %	2.18 %
Celkem	**62.75%**	**97.82 %** (100 %)

Příloha D

Adresářová struktura



Obrázek D.1: Adresářová struktura klientská části

```
readme.md ..... dokumentace s popisem, jak uvést projekt do provozu
www/ ..... veřejně přístupný adresář
    index.php ..... vstupní soubor delegující požadavek na bootstrap.php
    doc/ ..... dokumentace zdrojů
tests/ ..... adresář testovacích skriptů
log/ ..... adresář obsahující logovací soubory
temp/ ..... adresář obsahující dočasné soubory aplikace
vendor/ ..... adresář pro knihovny třetích stran
app/ ..... adresář s aplikační logikou
    bootstrap.php ..... závaděcí soubor aplikační logiky
    config/ ..... konfigurační soubory
        config.neon ..... konfigurační soubor frameworku a balíčků
    presenters/ ..... obecná prezentační vrstva
        ErrorPresenter.php
        BasePresenter.php
    module/ ..... aplikační moduly
        v1/ ..... modul zapouzdřující logiku pro rozhraní verze 1
            presenters/ ..... prezentační vrstva
                CocktailsPresenter.php
                UsersPresenter.php
                CategoriesPresenter.php
                AuthPresenter.php ..... generování a obnova tokenů
            model/ ..... modelová vrstva a její třídy
                CategoryModel.php
                RecommenderModel.php
                UserAuthenticator.php
                CocktailModel.php
                UserModel.php
                RatingModel.php
            services/
                RecommenderService.php
    exceptions/ ..... chybové třídy
        …
    utils/ ..... pomocné třídy
        ApiResponseFormatter.php
        ApiResponse.php
    router/ .3 RouterFactory.php ..... nastavení směrování v aplikaci
```

Obrázek D.2: Adresářová struktura serverové částí

Příloha E

Seznam použitých zkratek

API Application programming interface

CDN Content Delivery Network

HTTP Hypertext Transfer Protocol

JS JavaScript

JSON JavaScript Object Notation

PHP Hypertext Preprocessor

PWA Progressive Web Application

REST REpresentational State Transfer

UI User Interface

URL Uniform Resource Locator

Příloha F

Obsah přiloženého CD

- `project/` ..... zdrojové kódy práce
  - `frontend/` ..... zdrojový kód klientské části
  - `backend/` ..... zdrojový kód serverové části
- `thesis/` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `latex` ..... zdrojová forma práce ve formátu LaTeX