Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informačních Technologií

Progresivní webové aplikace

Bakalářská práce

Autor: Stanislav Máša
Studijní obor: ai3-p

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové duben 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 22.4.2020 Stanislav Máša

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Daniele Ponce, Ph.D. za metodické vedení práce a velmi cenné rady.

Anotace

Cílem této bakalářské práce je prozkoumat metodiku vývoje progresivních webových aplikací a vytvořit ukázkovou aplikaci, na které budou získané poznatky demonstrovány. První část práce se zaměřuje na popis jednotlivých technologií a nástrojů, které se při vývoji těchto aplikací využívají. Ve druhé části práce je pak popsána vlastní implementace ukázkové aplikace. Ta je implementována pomocí frameworku ASP.NET Core a s ním spojených technologií. Jako téma pro ukázkovou aplikaci byl zvolen zpravodajský web pro smyšlenou školu. Obsah této práce je určen zejména čtenářům, kteří již mají základní zkušenosti s tvorbou webových aplikací.

Annotation

Title: Progressive Web Apps

The aim of this bachelor thesis is to explore the methodology of the development of progressive web apps and to create a sample app to demonstrate earned knowledge. The first part of the bachelor thesis is focused on the description of the individual technologies and tools used for the development of these apps. The second part of the bachelor thesis describes the implementation of the sample app. This application is implemented by ASP.NET Core framework and related technologies. A news site for a fictional school was chosen as a topic of the sample app. The content of this thesis is intended especially for those who already have basic experience with web app development.

Obsah

1 Úvod

Cílem této práce je prozkoumat a popsat základní metodiky a pojmy v oblasti vývoje progresivních webových aplikací. Dále zjistit, jaké jsou dostupné technologie a nástroje pro tvorbu progresivních webových aplikací a identifikovat základní rozdíly mezi progresivními webovými aplikacemi a nativními mobilními aplikacemi. Na základě získaných poznatků pak bude navržena a vytvořena vlastní ukázková aplikace, kde budou tyto poznatky demonstrovány.

Bakalářská práce je rozdělena na dvě části, a to teoretickou a praktickou. V teoretické části práce jsou zmíněny a popsány základní principy a vlastnosti progresivních webových aplikací. Následně jsou v této části práce popsány dostupné technologie a nástroje pro vývoj progresivních webových aplikací včetně názorných ukázek. Poslední kapitola teoretické části bakalářské práce je zaměřena na popis základních rozdílů mezi progresivními webovými aplikacemi a nativními mobilními aplikacemi.

V praktické části jsou vysvětleny požadavky na referenční aplikaci, popsány použité technologie pro její vývoj, a nakonec jsou zde vysvětleny dílčí části implementace.

Od čtenářů této práce se očekávají základní znalosti v oblasti tvorby webových aplikací. Pojmy jako HTML, Javascript a další nejsou hlavním tématem této práce, a proto nebudou dopodrobna vysvětlovány.

2 Progresivní webové aplikace

2.1 Progresivní webové aplikace

Poprvé termínem „progresivní webové aplikace“ označil v roce 2015 inženýr Alex Russel (Russel, 2015) webové aplikace, které využívají nové webové technologie podporované moderními webovými prohlížeči, díky kterým jsou tyto aplikace schopny pracovat na pomalém internetovém připojení či dokonce bez internetového připojení, jejich vzhled je přizpůsoben pro různé typy zařízení (mobily, tablety apod.) nehledě na to, jaký operační systém uživatelé používají, dají se jednoduše nainstalovat na plochu a spustit mimo webový prohlížeč nebo znovu-zapojovat uživatele k využívání aplikace prostřednictvím notifikací. Obecně by se však dalo uvést, že se stále jedná o webové aplikace, které ovšem přebírají určité výhody a možnosti mobilních aplikací.

2.1.1 Požadavky na označení aplikace PWA

Webovou aplikaci označují moderní webové prohlížeče jako „progresivní“ ve chvíli, kdy webová aplikace splňuje 3 základní požadavky (Mills a kol, 2020):

Obsahuje validní **webový aplikační manifest** – soubor obsahující metadata, které určují chování webové aplikace po její instalaci na plochu uživatelského zařízení. Podrobnější informace o tomto souboru se nachází v samostatné kapitole této práce.
Obsahuje skript **Service Worker** – skript Service Worker slouží jako prostředník v komunikaci mezi klientem a serverem, umožňuje práci s pamětí *cache* a pomocí *Fetch API* umožňuje zachycovat síťové požadavky a vracet na ně vlastní odpovědi (například vlastní offline stránku). Technologie Service Worker je podrobněji popsána ve vlastní kapitole této práce.
Používá zabezpečený **HTTPS protokol** – toto je poslední požadavek na úspěšné označení webové aplikace jako „progresivní“. Zabezpečený HTTPS protokol je potřebný k tomu, aby ve webové aplikaci mohl být spuštěn skript Service Worker, ale i další nové technologie, které tento protokol vyžadují.

2.1.2 Základní vlastnosti progresivních webových aplikací

Jakmile jsou ve webové aplikaci splněny výše zmíněné požadavky, získává webová aplikace následující základní vlastnosti:

1) **Instalovatelnost** – aplikace lze nainstalovat na plochu uživatelského zařízení. Během instalace se na plochu stáhne ikona aplikace, pomocí které ji pak lze spouštět. Nainstalovanou aplikaci je možné spouštět přes celou obrazovku bez kontrolního panelu webového prohlížeče (záleží na implementaci manifestu). Díky absenci kontrolního panelu prohlížeče pak může vzhledem tato aplikace připomínat aplikaci mobilní.
2) **Offline režim** – tuto vlastnost nemusí mít všechny progresivní webové aplikace, jelikož záleží na implementaci scriptu Service Worker. Nicméně právě díky tomuto skriptu lze manipulovat s pamětí cache a ukládat do ní odpovědi na různé síťové požadavky (např. HTML stránky), odkud poté mohou být vráceny, pokud uživatel ztratí internetové připojení.
3) **Zabezpečenost** – progresivní webové aplikace běží vždy na zabezpečeném protokolu HTTPS, jelikož je to podmíněno využíváním služby Service Worker.

I nadále progresivním webovým aplikacím zůstávají vlastnosti běžných webových aplikací. Jmenovitě například snadné sdílení aplikace pomocí odkazu nebo fakt, že uživatel používá vždy nejaktuálnější verzi aplikace (nemusí stahovat aktualizace).

2.1.3 Webový aplikační manifest

Webový aplikační manifest je soubor JSON obsahující metadata, která určují chování progresivní webové aplikace v zařízení uživatele. Metadata, která se v manifestu definují, určují například název aplikace, popis, cesty k ikonám aplikace a jejich velikosti, barvu pozadí startovní obrazovky, startovní URL adresu aplikace nebo v jakém zobrazovacím režimu se má aplikace spustit. Všechny tyto parametry lze nalézt na webových stránkách konsorcia W3 (Caceres, 2020). Vytvoření souboru manifest je velmi jednoduché a lze ho vytvořit ručně pomocí jakéhokoliv textového editoru, nicméně na internetu lze najít několik online generátorů tohoto souboru. Ukázka definování základních parametrů v manifestu je zobrazena na obrázku 1.

```json
{
  "name": "Školní kurýr - zpravodaj ze školy",
  "short_name": "Školní kurýr",
  "description": "Mějte aktuální informace ze školy po ruce",
  "theme_color": "#000000",
  "background_color": "#ffffff",
  "display": "standalone",
  "orientation": "portrait",
  "Scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "content/defaults/icons/icon-128x128.jpg",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "content/defaults/icons/icon-192x192.jpg",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "content/defaults/icons/icon-512x512.jpg",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Obrázek 1: Nastavení parametrů v souboru manifest, Zdroj: vlastní tvorba

Popis parametrů v ukázce na obrázku 1 je následující:

- *name* – definuje název webové aplikace
- *short_name* – definuje zkrácený název aplikace, pokud by byl klasický název příliš dlouhý a špatně se zobrazoval na zařízení
- *theme_color* – určuje barvu horního panelu mobilního zařízení
- *background_color* – určuje barvu pozadí spouštěcí obrazovky
- *display* – nastavuje, jak se má aplikace zobrazit. Hodnota *„standalone“* spustí aplikaci přes celou obrazovku zařízení, vidět jsou pak jen ovládací prvky uživatelského zařízení a horní panel uživatelského zařízení se stavem baterie (pokud se jedná o mobilní telefon)
- *orientation* – nastavuje, zda se aplikace zobrazí na výšku (hodnota *„portrait“*) nebo na šířku (hodnota *„landscape“*)

- *scope* – nastavuje rozsah URL adres, pro které se má využívat tento manifest (hodnota „/“ určuje, že tento manifest má být aplikován pro všechny URL adresy nacházející se v dané webové aplikaci)
- *start_url* – určuje výchozí URL adresu aplikace, která se bude spouštět (v tomto případě je start_url nastaveno na hodnotu „/“, což znamená, že se spustí hlavní stránka aplikace)
- *icons* – jak je již z názvu patrné, parametr *icons* nastavuje ikonu, která se bude na úvodní obrazovce zobrazovat. U tohoto parametru lze nastavit více ikon (např. pro různě velká zařízení), u každé je však nutné určit cestu k souboru ikony, velikost ikony a typ obrázku.

2.1.4 Service Worker

Service Worker je služba ve formě scriptu psaná v jazyce JavaScript, která běží na pozadí webového prohlížeče a slouží jako komunikační prostředník mezi prohlížečem a serverem. Jeho účelem v oblasti progresivních webových aplikací je zachycovat webové http požadavky (tzv. requesty) a na základě dostupnosti sítě podnikat příslušná opatření pro vrácenou/nevrácenou odpověď (tzv. response). Použití Service Workera ve webové aplikaci může mít pozitivní dopad na celkový výkon aplikace (Geddes, 2019). Především kvůli možnosti ukládat odpovědi na webové požadavky včetně získaného obsahu do paměti cache, odkud je lze později ihned vracet. Z paměti cache lze vracet data i v případě, že není k dispozici internetové připojení, takže uživatel může využívat aplikaci nadále. V případě, že není internetové připojení k dispozici a odpověď na požadavek se nenachází v paměti cache, je Service Worker schopný zobrazit vlastní offline stránku. Vše výše zmíněné ovšem záleží na jeho implementaci. Dále je prostřednictvím služby Service Worker možné obsluhovat takzvané *Push Notifikace* nebo synchronizovat data na pozadí. Jelikož tato služba běží na pozadí, odděleně od vlastní aplikace, není možné jejím prostřednictvím přímo ovládat DOM webové aplikace.

Service Workera je možné používat pouze v případě, že webová aplikace běží pod zabezpečeným protokolem HTTPS nebo HTTP/2. Je to z důvodu bezpečnostních opatření před útoky typu „man-in-the-middle“ (Gaunt, 2019).

2.1.4.1 Registrace a životní cykly

Aby služba mohla obsluhovat události webové aplikace, musí být zaregistrována v prohlížeči. Od registrace služby až do momentu, kdy je služba schopna zachycovat události webové aplikace, prochází Service Worker několika stavy, kterým se dohromady říká životní cykly (ang. *lifecycle*) (Aderinokun, 2016). Tyto jednotlivé stavy jsou: *installing*, *installed/waiting*, *activating*, *activated* a *reduntant* a budou popsány v této kapitole.

Jelikož služba Service Worker není podporována ve všech prohlížečích, je nutné nejprve zjistit, zda je ji vůbec možné pro daný prohlížeč registrovat. Ukázka registrace služby včetně zjištění její podpory prohlížečem, je zobrazena na obrázku 2.

```
1  if('serviceWorker' in navigator){
2      //Služba je v prohlížeči podporována
3      navigator.serviceWorker.register('/service-worker.js')
4          .then((swRegistration) => {
5              console.log('Služba byla zaregistrována');
6          })
7          .catch((error) => {
8              console.log('Chyba během registrace služby', error);
9          })
10 }
```

Obrázek 2: Registrace Service Workera v aplikaci,
Zdroj: vlastní tvorba

V ukázce registrace služby na obrázku 2 je nejprve pomocí podmínky zjištěno, zda prohlížeč podporuje službu Service Worker. Následně je příkazem na řádku 3 zavolána jeho registrace. Jako parametr je do metody *register* použita cesta k javascriptovému souboru této služby. Pokud je potřeba, aby Service Worker obsluhoval veškeré části webové aplikace, je nutné jej umístit do kořenového adresáře webu. Příkaz *register* buď uspěje a vrátí objekt představující nově vytvořenou službu (řádek 4), nebo selže s chybou (vzniklou např. při neúspěšném stažení zadaného skriptu), která pak může být vypsána do konzole (řádky 7-8). Během registrace prochází postupně Service Worker již výše zmíněnými stavy.

Installing – v tomto stavu je služba během zpracování události *install*. Událost *install* je zpracovávána již samotným Service Workerem. Při zpracování této události se obvykle vytváří nová paměť cache, do které se uloží soubory, které mají být pro webovou aplikaci dostupné i ve chvíli, kdy uživatel nemá přístup k internetu (Mills a kol, 2014). Těmito soubory mohou být například soubory kaskádových stylů, fonty, hlavní stránka webu či vlastní offline stránka. Po dokončení instalace přechází Service Worker do stavu *installed/waiting*.

Installed/Waiting – je stav, kdy už byl dokončen proces instalace nové služby. V tuto chvíli služba čeká, až uživatel zavře všechna otevřená okna s webovou aplikací (nebo již samotnou nainstalovanou aplikaci), aby mohl přejít do fáze *activate*. Je to z toho důvodu, že webovou aplikaci již může obsluhovat jiná verze služby, ale zároveň ji nesmí obsluhovat více služeb najednou (Archibald, 2019). Proto se čeká, až nebude uživatel aplikaci používat, aby mohla být nová verze služby aktivována a ta stará naopak odstraněna.

Activating – je stav, resp. událost typu *activate*, která nastane, jakmile jsou zavřena všechna okna s danou webovou aplikací, či je zavřena samotná již nainstalovaná aplikace (uživatel v tuto chvíli aplikaci nepoužívá). Typ události *activate* opět zachytí Service Worker a provede výměnu sebe sama za novější verzi. Během této výměny je možné například smazat starou verzi paměti cache, která může obsahovat již nepotřebná data. Událost *activate* lze i vynutit příkazem *self.skipWaiting()* uvnitř obsluhy události *install*, která zapříčiní, že výměna Service Workerů může proběhnout i během používání aplikace (Contractor, 2020). Po výměně Service Workerů přechází nová verze do stavu *activated* a stará verze do stavu *redundant*.

Activated – pokud dosáhne Service Worker stavu *activated*, znamená to, že převzal kontrolu nad webovou aplikací a může obsluhovat veškeré události na webové aplikaci – například obsluhovat události *fetch*.

Redundant – v tomto stavu je Service Worker ve chvíli, kdy byl nahrazen svojí novější verzí nebo pokud selhal proces jeho instalace či aktivace a není tedy používán.

2.1.4.2 Zachycování webových požadavků

Aby bylo možné zachycovat požadavky webové stránky na různé zdroje (obrázky, soubory kaskádových stylů, jiné html stránky apod.) nacházející se na serveru, je potřeba zaregistrovat si ve skriptu Service Worker zachycování takzvané *fetch* události. Následně začne Service Worker odchytávat všechny http požadavky. Každým http požadavkem vznikne nový objekt *FetchEvent*, který obsahuje informace o daném požadavku a zároveň poskytuje metodu *respondWith()*, díky které je možné na odchycený požadavek reagovat vlastním způsobem (Scholz, 2014). Na obrázku 3 je znázorněna registrace zachycování *fetch* událostí a příklad vlastního zpracování webového požadavku.

```
1   self.addEventListener('fetch', (event) => {
2       event.respondWith(
3           caches.match(event.request)
4           .then((cachedResponse)=> {
5               return cachedResponse || fetch(event.request)
6                   .then((networkResponse) => {
7                       caches.open('CacheMojiAplikace').then((cache) => {
8                           cache.put(event.request,networkResponse.clone());
9                           return networkResponse;
10                      })
11                  })
12          })
13      );
14  });
```

Obrázek 3: Registrace zachycování fetch událostí a vlastní zpracování webového požadavku, Zdroj: vlastní tvorba

Prvním řádkem v ukázce na obrázku 3 se zaregistruje zachycování *fetch* události. V příkazu *event.respondWith* na řádcích 2-13 je vytvořený nový *Promise*, který je následně vrácen webové stránce. Příkazem na řádku 3 je nejprve zjištěno, zda se zdroj, na který webová stránka odeslala požadavek, nachází v paměti cache. V případě, že se zde zdroj nacházel, je vrácen webové stránce. V opačném případě je požadavek příkazem *fetch(event.request)* přeposlán na server, který vrátí požadovaný zdroj. Tento zdroj *(networkResponse)* je nejprve uložen do paměti cache (řádky 6-8) a posléze vrácen webové stránce. Díky uložení tohoto zdroje do cache jsou pak data při dalších requestech webové stránky na tentýž zdroj vrácena z cache

a nebude tak zatěžován server. Tomuto postupu se říká „*cache falling back to network*“ a jedná se o jednu z několika tzv. *caching strategies* pro zpracování požadavků (Jaiswal, 2019).

2.1.4.3 Strategie cachování

Jak již bylo zmíněno, v Service Workeru lze vlastním způsobem zpracovávat a vracet zachycené požadavky webové stránky na různé zdroje. Existuje ovšem několik návrhů, které se obvykle využívají. Těmto návrhům se říká tzv. „*caching strategies*“ a každý návrh je vhodný pro jiné druhy obsahu. Velmi přehledně tyto návrhy popisuje ve svém příspěvku Jake Archibald (Archibald, 2019).

Cache only – Tento návrh je vhodný především pro získávání zdrojů, které byly během instalace Service Workera vloženy do paměti cache. Jakmile Service Worker zachytí požadavek na tento zdroj, vrací zdroj z cache a vůbec neodesílá požadavek na server. Získávání zdrojů z cache je výrazně rychlejší než získávání ze serveru (Rawat, 2018), proto se tento návrh zpravidla využívá pro statické soubory, které se ve webové aplikaci nemění (například soubory kaskádových stylů, úvodní obrázek, statické webové stránky apod.).

Network only – Jedná se v podstatě o opačné řešení předchozího návrhu. Zachycený požadavek je vždy odeslán na server a odpověď ze serveru je vrácena bez uložení do paměti cache. Toto řešení je vhodné pro všechny requesty, u kterých se vyžaduje, aby byly zpracovány na serveru. Network only návrh se využívá pro HTTP POST requesty, protože POST requesty nelze ukládat do paměti cache, odkud by mohly být později načítány (Russell a kol, 2020).

Cache falling back to network – Tento návrh byl popsán v přechozí kapitole. Jde v podstatě o rozšíření návrhu *„cache only“*. Pokud neexistuje odpověď na zachycený request v cachi, je request poslán na server, který vrátí požadovanou odpověď. Tato odpověď je následně uložena do cache, takže při dalším requestu je již vrácena odtud. Tento návrh je vhodný pro *„offline first“* aplikace, kdy není potřeba vracet uživateli nejaktuálnější data.

Network falling back to cache – Tento typ návrhu rozšiřuje *network only* návrh. Zachycený request je odeslán na server. Pokud server vrátí response, je response

vrácena uživateli a zároveň uložena do cache. Pokud při dalším requestu na tento zdroj server nic nevrátí (uživatel je např. offline), vrátí se alespoň odpověď uložená v cachi. Tento návrh je vhodný, pokud má aplikace vracet uživateli nejaktuálnější data, ale v případě že je není možné vrátit, vrátí alespoň naposledy stažená data.

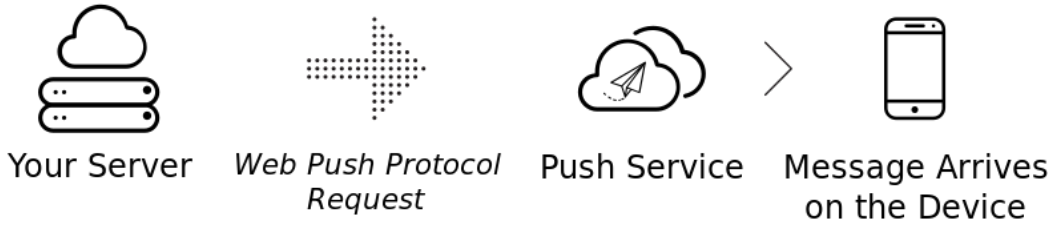
Může se stát, že selže jak vrácení odpovědi ze serveru (uživatel je například offline), tak vrácení z cache (odpověď například nebyla dosud stažena). V těchto případech je vhodné využít dalšího návrhu, který se nazývá **Generic fallback**. V tomto návrhu se uživateli vrací nějaká výchozí odpověď. Tou může být například vlastní offline stránka nebo pouhý výpis chyby do konzole prohlížeče. Všechny tyto návrhy lze mezi sebou různě kombinovat podle toho, o jaký požadavek se jedná, a tím zajistit vhodné chování celé aplikace.

2.1.5 Push notifikace

Push notifikace je vyskakovací oznámení, které je zasíláno přímo do zařízení (resp. prohlížeče) uživatele. V tomto oznámení se může vyskytovat například upozornění na nově přidaný článek na webu, na slevu k produktu apod. Díky Service Workeru a dalším technologiím je dnes zasílání takzvaných webových push notifikací dostupné nejen pro progresivní webové aplikace, ale i pro běžné webové aplikace. Díky využívání push notifikací je pak možné docílit častějšího využívání webové aplikace uživateli (Medley, 2019). Notifikace se mohou zobrazit, i když uživatel zrovna webovou aplikaci nepoužívá a má zavřený prohlížeč.
K zasílání notifikací do uživatelského zařízení jsou nutné 3 kroky. Tím prvním je, že si uživatel musí ve svém prohlížeči povolit zasílání oznámení. Toho je docíleno využitím metody *subscribe()* v rozhraní **Push API** (Medley a kol, 2019). Následně je nutné získat tzv. *„PushSubscription“*, která obsahuje informace potřebné k zasílání notifikací a tato data uložit na server, odkud jsou později využívány pro zasílání notifikací. Tento proces je zobrazen na obrázku 4.

Obrázek 4: Proces získání dat k zasílání notifikací, Zdroj: https://developers.google.com/

Po povolení zasílání notifikací a uložení dat o jejich odběru je již možné ze serveru odesílat oznámení. K tomu je využívána tzv. *„push service"*. Jedná se o API poskytované výrobci internetových prohlížečů, které slouží jako prostředník mezi aplikačním serverem a koncovým zařízením. Při povolení zasílat uživateli oznámení totiž dojde nejen k odeslání *PushSubscription* na aplikační server, ale i k registraci tohoto zařízení (prohlížeče) do *push service*. Zaslání notifikace z aplikačního serveru až do prohlížeče uživatele je zobrazeno na obrázku 5.



Obrázek 5: Průběh zaslání notifikace do koncového zařízení, Zdroj: https://developers.google.com/

K autentifikaci zaregistrovaných odběrů využívá služba *push service* dvojici klíčů zvaných VAPID keys (viz kapitola 2.1.5.1 VAPID Keys). Na aplikačním serveru se vytvoří nová notifikace, která se má odeslat do prohlížeče uživatele. Tato notifikace je pomocí *Web Push Protokolu* (Thomson, Damaggio a Raymor, 2015) zaslána do *push service*. Ta najde v zaregistrovaných odběrech požadovaný prohlížeč a přepošle data notifikace přímo do prohlížeče.

K zobrazení nově příchozí notifikace se v Service Workeru využívá *Notifications API* (Mills a kol, 2019). Pomocí něho se vytváří na základě příchozích dat konečné

zobrazení notifikace a její chování (například když uživatel na notifikaci klikne, otevře se konkrétní URL adresa apod).

2.1.5.1 VAPID Keys

Voluntary Application Server Identification for Web Push (zkráceně *VAPID*) je dvojice klíčů, které slouží k autorizaci notifikací přijímaných službou *push service* z aplikačního serveru (Thomson a Beverloo, 2016). Tato dvojice je tvořena privátním klíčem a veřejným klíčem. K privátnímu klíči má přístup pouze aplikační server a veřejný je dostupný například v souboru *site.js* obsluhujícím registraci nových odběrů. Během vytvoření nového odběru se tomuto odběru přiřadí veřejný klíč tak, jak je znázorněno na obrázku 6.

```
55
56  function registerPushSubscription() {
57
58      const appServerPublicKey =
59          'BIQyA5ul6H3SbO3hwDkFVZBfbneRtSqAOPu-V8OadUB_DJ9w3be2xI8PAhnIJbAWa-MePu7hv4e409FptiFg7NU';
60
61      var subscribeParams = {
62          userVisibleOnly: true,
63          applicationServerKey: appServerPublicKey
64      };
65
66      swRegistration.pushManager.subscribe(subscribeParams)
67          .then((subscription) => {
68
69              //Zaslání dat nového odběru na aplikační server
70
```

Obrázek 6: Předání veřejného VAPID klíče do nového odběru notifikací, Zdroj: vlastní tvorba

Nový odběr se s tímto parametrem uloží ve službě *push service*. Když je pak z aplikačního serveru odesílána nová notifikace, je potřeba aby se zároveň zaslal veřejný i privátní klíč. Díky tomu je možné zamezit odesílání notifikací z jiných serverů nežli je ten, ve kterém se nachází privátní klíč. Klíče VAPID keys lze získat z různých online generátorů.

2.1.5.2 Knihovna WebPush

WebPush je knihovna vytvořená pro jazyk C#, která usnadňuje vývojářům zasílání push notifikací z aplikačního serveru prostřednictvím *Web Push Protokolu* a jejich šifrování pomocí protokolu *Message Encryption for Web Push* (Thomson, 2017).

Tato knihovna je veřejně dostupná z GitHubu[1] nebo NuGetu[2]. Pro vytvoření notifikace, která může být posléze úspěšně odeslána, se používají tři třídy z této knihovny. První z nich je třída *VapidDetails*, do které je nutné vložit privátní a veřejný klíč (viz kapitola 2.1.5.1) a ještě vyplnit parametr *subject*, do kterého se zadává kontaktní email správce aplikace. Druhou třídou je *PushSubscription*, do které je nutné vložit informace ze získaného odběru, a nakonec třetí třída *WebPushClient* a její metoda *SendNotification* přijímající jako parametry dvě výše zmíněné třídy, která se postará o zašifrování dat a odeslání do *push service*.

2.2 Dostupné nástroje pro vývoj PWA

Pro vývoj webových aplikací existuje spousta nástrojů, které různým způsobem ulehčují nebo zefektivňují práci vývojářům. V této kapitole bude popsáno jen několik nástrojů, které pomáhají s vývojem nejen běžných webových aplikací, ale i těch progresivních.

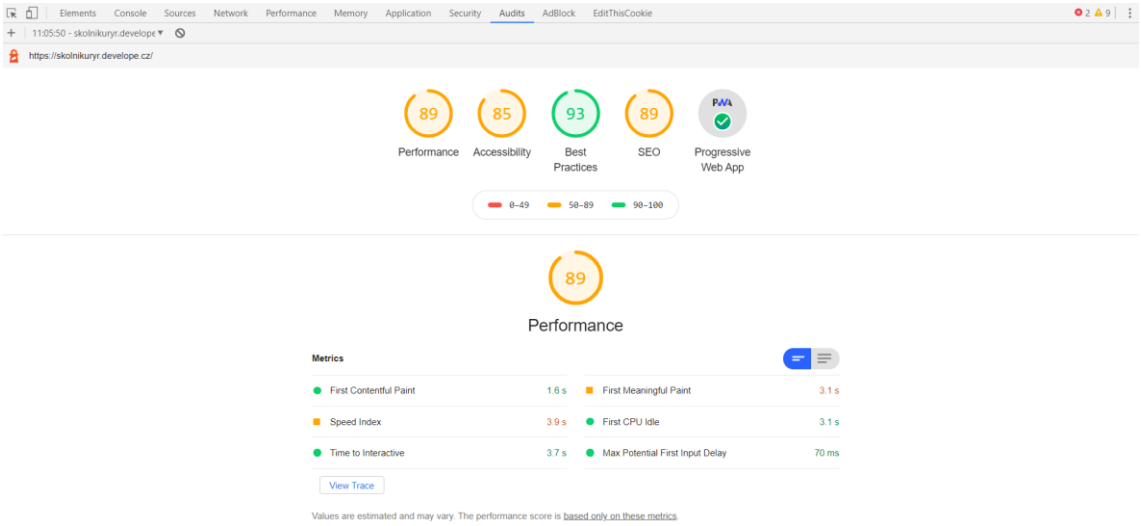
2.2.1 Lighthouse

Prvním z nástrojů, který je užitečný pro vývoj webových aplikací včetně těch progresivních, je open-source diagnostický nástroj Lighthouse od společnosti Google. Analyzuje webovou aplikaci v několika oblastech: výkon webu, přístupnost webu, tzv. „*best practices*“, SEO („*Search Engine Optimalization*“ – optimalizace pro vyhledávače) a nakonec PWA. Pro každou z oblastí následně vygeneruje report, ve kterém sděluje výsledky jednotlivých testů a případně navrhuje, jakým způsobem lze nalezené problémy opravit (Michálek, 2018).

Lighthouse například zjistí, jak veliké soubory se do stránky stahují a jaký to má vliv na rychlost načtení webu, zda web používá správný barevný kontrast mezi textem a pozadím, zda mají veškeré obrázky vyplněný atribut `alt`, zda je web validní z hlediska HTML kódu (stránka obsahuje tag `<title>` apod) nebo zda je web validní z hlediska základních SEO praktik.

[1] GitHub - https://github.com/
[2] NuGet - https://www.nuget.org/

Oblast PWA je v nástroji Lighhouse rozdělena do tří sekcí. V sekci „*Fast and Reliable*“ je možné zjistit, jak si načítání webové aplikace vede na pomalejším připojení nebo je-li dostupná hlavní stránka aplikace bez internetového připojení. V sekci „*Installable*“ jsou kontrolovány základní požadavky na označení webové aplikace jako progresivní. Těmi jsou (jak již bylo zmíněno v kapitole *2.1.1*): přítomnost webového aplikačního manifestu, přítomnost skriptu Service Worker a zda aplikace běží na protokolu HTTPS. V poslední sekci „*PWA Optimized*“ je možné zjistit, jak si aplikace vede z hlediska dalších doporučení při vývoji PWA. Ukázka z reportu webové stránky je zobrazena na obrázku 7.



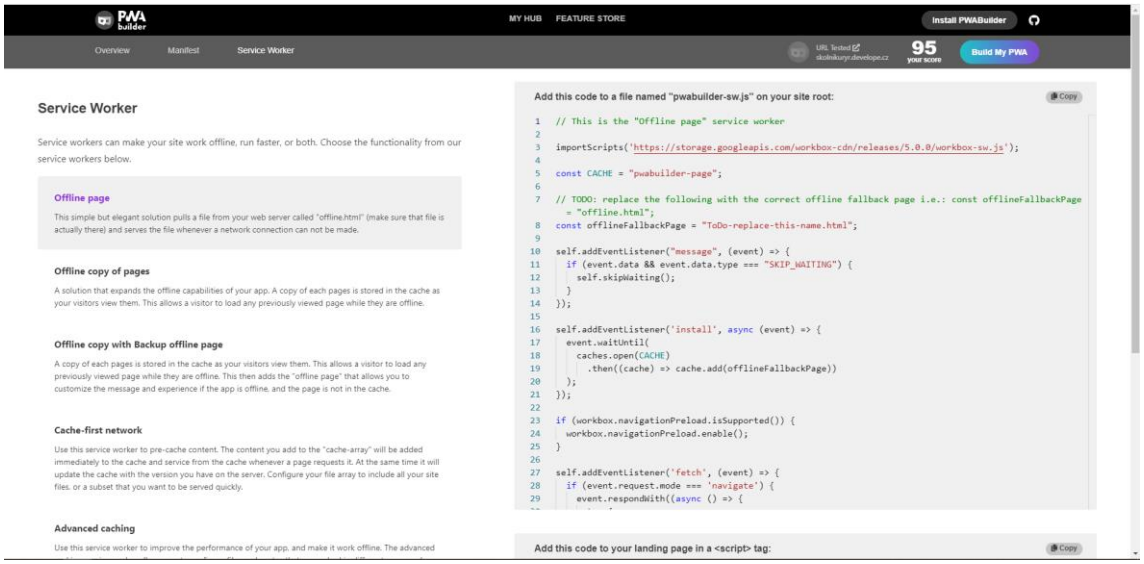
Obrázek 7: Ukázka výsledků z reportu nástroje Lighthouse, Zdroj: vlastní tvorba

2.2.2 PWABuilder

Velmi rychlý a jednoduchý upgrade webové aplikace na progresivní nabízí open source nástroj od společnosti Microsoft. Jeho uživatelské prostředí je zobrazeno na obrázku 8. V tomto nástroji je možné rychle vytvořit soubor manifest pouhým vyplněním formuláře nebo si vybrat jednu z několika variant kódu skriptu Service Worker, který se posléze jen zkopíruje do souboru v aplikaci. Jednou z těchto variant je například Offline copy of pages, která ukládá navštívené stránky aplikace do paměti cache, odkud pak mohou být vráceny, když bude uživatel offline.

Další z variant, které lze pro webovou aplikaci vybrat, je varianta *Cache first*, která využívá stejnojmennou strategii viz kapitola 2.1.4.3.

V generovaných kódech využívá PWA builder knihovnu *Workbox*, která bude popsána v následující kapitole.



Obrázek 8: Uživatelské prostředí nástroje PWABuilder, Zdroj: vlastní tvorba

2.2.3 Workbox

Workbox je JavaScriptová sada knihoven vyvinutá společností Google, kterou lze využít ke snazšímu spravování zdrojového kódu uvnitř skriptu Service Worker. Nabízí pohodlnější práci s pamětí cache a dokáže vytvářet rozsáhlé reporty v konzoli prohlížeče (Posnick, 2018). Pomocí Workboxu lze pohodlně určovat, které soubory mají být ukládány do paměti cache, za jakých podmínek a na jak dlouho. Workbox nabízí několik cachovacích strategií, které je možné využít pro různé typy požadavků. Tím lze snadno docílit požadovaného chování webové aplikace.

Workbox lze získat několika způsoby. Jedním z nich je naimportování knihovny z CDN příkazem `importScripts('url-adresa-cdn')` ve skriptu Service Workera, kde je parametr `'url-adresa-cdn'` nahrazen konkrétní URL adresou CDN. Tento import vytvoří objekt `workbox`, přes který lze přistupovat k dalším potřebným knihovnám a funkcionalitám, které Workbox nabízí. Na obrázku 8 je znázorněno, jakým způsobem lze například pracovat se zpracováváním requestů webové stránky na obrázky.

```
1   workbox.routing.registerRoute(
2       new RegExp('\.jpg'),
3       workbox.strategies.cacheFirst({
4           cacheName: 'mojeCache-obrazky'
5       })
6   );
```

Obrázek 9: Ukázka použití Workboxu, Zdroj: vlastní tvorba

Na prvním řádku v obrázku 9 se pomocí objektu *workbox* a příkazem *registerRoute* zaregistruje do Service Workeru URL adresa requestu, pro kterou bude následně zvolena strategie *cache first*. Regulérním výrazem na druhém řádku je určeno, že tato strategie bude platit pro všechny obrázky s příponou *jpg*. Příkazem *cacheFirst* na třetím řádku je pak definováno, s jakou pamětí (řádek 4) má Service Worker pro dané requesty pracovat.

2.3 PWA versus nativní mobilní aplikace

Progresivní webové aplikace a mobilní aplikace jsou rozdílné v mnoha ohledech. V této kapitole budou nastíněny alespoň základní rozdíly mezi těmito aplikacemi.

Prvním rozdílem je samotný způsob vývoje. Zatímco PWA (stejně jako běžná webová aplikace) je vytvářena za pomoci technologií standardizovanými webovým konsorciem W3.org (HTML, JavaScript, CSS a další), nativní mobilní aplikace jsou vytvářeny pomocí programovacích jazyků určených pro danou platformu (Michálek, 2017).

Z hlediska distribuce a dostupnosti jsou mezi těmito aplikacemi také jisté rozdíly. Zatímco mobilní aplikace mohou být distribuovány pomocí obchodů s mobilními aplikacemi (App Store, Google Play a další), odkud si je následně uživatel musí do svého zařízení stáhnout a nainstalovat, PWA jsou uživatelům dostupné ihned po zadání jejich URL adresy do prohlížeče. Navíc za pomoci technologie Trusted Web Activity (TWA) lze progresivní webové aplikace dostat i do obchodu Google Play (Mclachlan, 2019).

Přestože se PWA díky možnosti efektivní práce s pamětí cache a vlastnímu zpracování webových požadavků dokáže načítat rychleji než běžné webové aplikace, nevyrovná se její výkon aplikacím mobilním.

Zásadní rozdíl mezi webovými a mobilními aplikacemi je v přístupu k systémovým prostředkům a k dalším procesům mobilního zařízení. Mobilní aplikace mají v tomto ohledu převahu a oproti PWA umožňují přístup k většině funkcí mobilního zařízení. Příkladem může být například Geofencing[3], pomocí kterého lze spouštět v mobilní aplikaci různé události. Dále PWA oproti mobilním aplikacím neumožňují komunikovat s jinými aplikacemi v zařízení (například není možné přes PWA přistupovat k telefonním kontaktům uložených v mobilním telefonu). Přehledný výpis funkcí, ke kterým má dnešní web přístup, vytvořil na svých webových stránkách Adam Bar (Bar, 2020).

[3] Geofencing – umožňuje sledovat pohyb zařízení v určitém vymezeném prostoru

3 Vývoj ukázkové aplikace

Tato kapitola bakalářské práce je věnována vývoji ukázkové progresivní webové aplikace. Budou zde popsány jednotlivé fáze vývoje, použité technologie a vysvětleny principy PWA.

Cílem této bakalářské práce není vytvořit plnohodnotnou aplikaci, nýbrž nastínit jakým způsobem lze progresivní webovou aplikaci vytvořit. Proto se také nebude jednat o aplikaci určenou k nasazení v reálném prostředí.

3.1 Požadavky a návrh aplikace

3.1.1 Základní požadavky na aplikaci

Jak již bylo zmíněno, cílem práce je vytvořit ukázkovou aplikaci, na které budou demonstrovány poznatky získané v teoretické části práce.

Tématem ukázkové aplikace je školní zpravodajský web, jehož účelem je zlepšit informovanost rodičů, studentů školy i veřejnosti. Na takovém webu budou moci čtenáři najít nejnovější příspěvky ze školních událostí (například články o výletech, sportovních akcích, přednášek a jiné) a oznámení (například o budoucích rodičovských schůzkách apod).

Aby bylo možné pracovat s informacemi na webu, bude vytvořen jednoduchý redakční systém, do něhož budou mít přístup pouze učitelé a administrátor. Vstup do systému pro správu obsahu (CMS) bude chráněn uživatelským emailem, heslem a uživatelskou rolí. Ty se budou v celé webové aplikaci vyskytovat celkem 4 a jejich oprávnění bude popsáno níže.

V uživatelské části aplikace bude kladen důraz na vlastnosti progresivních webových aplikací. Aplikace tedy bude mít responzivní webdesign, musí se dát nainstalovat na plochu uživatelského zařízení, vybraný obsah aplikace se bude moci zobrazovat i bez připojení k internetu a celá aplikace poběží na zabezpečeném protokolu HTTPS. K tomu budou v aplikaci implementovány i push notifikace, které budou uživatelům zobrazovat oznámení o nejnovějším obsahu na webu.

3.1.2 Rozdělení uživatelských rolí a práv

Oprávnění uživatelů v ukázkové aplikaci bude řízeno pomocí uživatelských rolí. V celé aplikaci se budou vyskytovat 4 role, které budou povolovat nebo zakazovat přístup do jednotlivých částí aplikace nebo k jednotlivým funkcím.

- *„Member“* – tato role popisuje zaregistrovaného uživatele, který bude mít přístup pouze do uživatelské části webu. V této části bude mít přístup ke všem funkcím a může si zažádat o zasílání oznámení o přidání nového příspěvku.
- *„Editor“* – tato role popisuje uživatele, jemuž bude povolen přístup do redakčního systému. Bude mu zde povoleno vytvářet, upravovat a mazat příspěvky. Tato role bude určena především pro učitele.
- *„Admin“* – jak již název napovídá, jedná se o administrátorskou roli celé aplikace. Administrátor bude mít přístup do všech částí redakčního systému, ale i ke všem funkcím v aplikaci. Výběrově se bude jednat například o vytváření a editaci nových kategorií pro příspěvky. Dále tato role bude umožňovat správu uživatelských účtů.
- Poslední „rolí“ v aplikaci je běžný návštěvník, který si nevytvořil účet. Ten má přístup pouze do klientské části aplikace, ve které mu je umožněno pouze procházet a číst příspěvky. Na rozdíl od registrovaných uživatelů mu tedy není umožněno získávat oznámení o nově přidaných příspěvcích nebo si nemůže ukládat své oblíbené příspěvky. Nicméně nainstalovat si aplikaci do svého zařízení může jako kterýkoliv jiný uživatel.

3.1.3 Redakční systém

Redakční systém bude vizuálně odlišný od uživatelské části aplikace. Pro snadnější a přehlednější administraci bude u všech výpisů článků, kategorií nebo uživatelů vytvořeno stránkování, vyhledávání, filtrace a řazení.

3.1.4 Uživatelská část

Uživatelská část aplikace bude mít responzivní design, neboť se očekává, že přístup do aplikace bude především z mobilního zařízení. Hlavní stránka bude obsahovat

výpis nejnověji přidaných příspěvků. V horní části uživatelského rozhraní bude navigační menu, pomocí kterého se uživatel bude moci pohybovat mezi výpisy příspěvků a oznámení. U těchto výpisů pak bude implementováno i vyhledávání a filtrace.

3.2 Implementace aplikace

3.2.1 Použité technologie

V této části jsou popsány stěžejní technologie použité pro vývoj ukázkové webové aplikace.

3.2.1.1 ASP.NET Core MVC

ASP.NET Core je multiplatformní framework od společnosti Microsoft, vyvíjený jako open-source a je vhodný pro tvorbu výkonných webových aplikací (Roth, Anderson, Luttin, 2019). Tento framework nabízí pro tvorbu webových aplikací několik vzorů a technologií, pomocí kterých lze webové aplikace tvořit. Jednou z nich je architektura MVC (Model-View-Controller). Tato architektura rozděluje celou aplikaci do tří hlavních částí: modelů, pohledů a řadičů. Model se stará především o business logiku aplikace a přístup k informacím. Pohledy se starají o zobrazení zpracovaných dat a pomocí nich uživatel interaguje s aplikací. Řadiče se pak starají o komunikaci mezi modely a pohledy.

Pro efektivní práci s tímto frameworkem i ostatními použitými technologiemi bylo zvoleno vývojové prostředí Microsoft Visual Studio 2019.

3.2.1.2 Entity Framework Core

Entity Framework Core je multiplatformní verze ORM (Object-Relational-Mapper) frameworku Entity Framework běžící na platformě .NET Core. Tento framework umožňuje práci s databázovými daty pomocí mapování těchto dat na objektové modely (třídy). Pro práci s daty v databázi využívá integrovaný jazyk LINQ, který dokáže generovat SQL dotazy.

Entity Framework Core podporuje dva vývojové přístupy – *Code-First* a *Database-First*. U Code-First přístupu se databáze a její tabulky vygenerují na základě doménových tříd. Database-First přístup funguje opačně, tedy že na základě

databáze se vygenerují doménové třídy. V ukázkové aplikaci je použit přístup *Code-First*.

3.2.1.3 Bootstrap

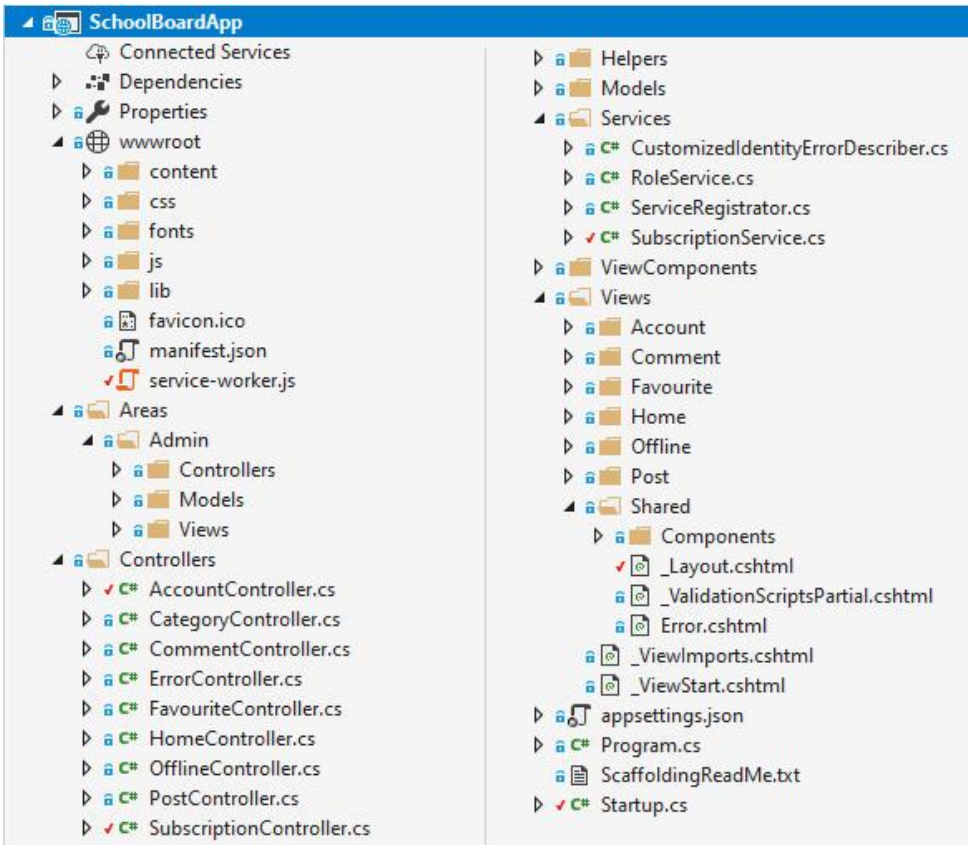
Bootstrap je nástroj určený pro tvorbu webových aplikací pomocí HTML, CSS a JavaScriptu. Obsahuje řadu užitečných komponent jako jsou formuláře, tlačítka, dialogová okna atp., a především disponuje mobile-first mřížkovacím (grid) systémem, díky kterému lze velmi snadno docílit responzivního designu.

3.2.2 Implementace základní webové aplikace

Nejprve byla vytvořena základní webová aplikace, která poté byla upravena na progresivní. Zdrojový kód celé aplikace je rozdělen do dvou .NET projektů. První je DataAccess – knihovna tříd, pomocí které se komunikuje s databází. V této knihovně jsou vytvořeny třídy, které představují jednotlivé tabulky v databázi. Druhý projekt nese název SchoolBoardApp a jedná se o projekt, který se stará o komunikaci s uživatelem. Nachází se zde řadiče, pohledy a pomocné třídy.

Jak již bylo zmíněno, knihovna DataAccess slouží ke komunikaci s databází. Jsou zde vytvořeny mapovací třídy, podle kterých následně Entity Framework Core vygeneruje tabulky v databázi a při dotazování vrácená data mapuje zpět na tyto třídy, takže pak lze pracovat s daty jako s objekty. Tato knihovna také využívá *ASP.NET Core Identity*, což je systém, který poskytuje služby pro správu uživatelských účtů a rolí. Dále se v knihovně DataAccess nachází třída *AppDbContext* dědící ze třídy *IdentityDbContext,* ve které se nastavují vazby mezi tabulkami a chování databáze. V poslední řadě se v této knihovně nachází migrační třídy generované Entity Frameworkem, které slouží k pohybu mezi jednotlivými verzemi databáze. Pro vytvoření databáze a pohybu mezi verzemi lze využít nástroj *PMC (Package-Manager-Console)*. Příkazem `add-migration „Název migrace“` (text v uvozovkách se nahradí zvoleným názvem a odeberou se uvozovky) Entity Framework vygeneruje migrační třídu, ze které je později vygenerován SQL kód. Příkaz `UpdateDatabase` pak vygeneruje SQL kód z naposledy vytvořené migrace a vytvoří nebo updatuje databázi.

Struktura projektu SchoolBoardApp zobrazená na obrázku 10 se stará o celý chod webové aplikace. Adresář *„wwwroot“* slouží jako složka pro uložení statických souborů webu. Jsou zde uloženy soubory s kaskádovými styly, potřebné javascriptové knihovny, fonty nebo například obrázky. K rozdělení webové aplikace do více částí slouží takzvané *„oblasti (Areas)“*. Do nich poté lze nastavit například přístupová práva tak, jako je tomu i v tomto případě. Oblast *„Admin“*, která představuje onen redakční systém, je určena pouze pro uživatelské role Admin a Editor. Dále se v projektu nachází složka s řadiči. Řadičů je v projektu několik, každý z nich obsluhuje požadavky na jiná data či pohledy.



Obrázek 10: Struktura projektu, Zdroj: vlastní tvorba

Pohledy jsou uloženy ve složce Views a jsou rozděleny do podsložek podle názvů řadičů, které tyto pohledy vrací. Kód pohledů je psán značkovací syntaxí Razor s pomocí tzv. ASP NET Core Tag Helpers. Jedná se o kombinaci HTML kódu a C# kódu. C# kód se do HTML vnořuje pomocí značky zavináč a lze jím do HTML například propisovat hodnoty různých proměnných. ASP NET Core Tag Helpers jsou speciální značky, které pomáhají vykreslovat HTML značky na straně serveru.

Ukázka na obrázku 11 zobrazuje část pohledu, který se stará o výpis příspěvků. Hned na prvním řádku je pomocí znaménka „@“ vnořen do pohledu foreach cyklus, ve kterém se pro každý příspěvek (post) vygeneruje vepsané HTML. Odkaz směřující na detailní pohled příspěvku je vytvořen HTML tagem `<a></a>`, ale místo parametru `href=""` jsou zde použity ASP NET Core Tag helpers „`asp-action, asp-controller, asp-area a asp-route-id`“. V nich se definuje, jaký se má zavolat řadič, jaká se v něm zavolá akce, ve které oblasti se řadič nachází a pomocí asp-route-id se předává parametr dané akce – v tomto případě tedy identifikátor příspěvku. Na základě těchto definic je pak uvnitř tagu `<a>` výsledného HTML kódu vygenerován parametr `href="/post/detail/9"`.

```
@foreach (var post in Model)
{
    var imageUrl = string.IsNullOrEmpty(post.Image) ? "/content/defaults/no-image.jpg" : $"/{AppConstants.Image_Folder}/{post.Image}";

    <div class="row py-3 py-md-2 border-top">
        <div class="col-md-3 px-md-0">
            <img src="@imageUrl" class="img-fluid" alt="@post.Title" />
        </div>
        <div class="col-md-9 my-auto">
            <h2 class="h4 pt-3 pt-md-0">
                <a asp-action="Detail" asp-controller="Post" asp-area="" asp-route-id="@post.Id" class="text-dark">@post.Title</a>
            </h2>
        </div>
        <div class="col-12 px-md-0 pt-2 text-muted category-span">
            <span class="px-2">
                <i class="fas fa-calendar-alt mr-2"></i>@post.CreationDate.ToString("dd.MM.") @post.CreationDate.ToShortTimeString()
            </span>
            <span class="px-2 border-left border-right">
                <i class="fas fa-tags mr-2"></i>@(post.Category != null ? post.Category.Title : "Nezařazeno")
            </span>
            <span class="px-2"><i class="fas fa-user mr-2"></i>@post.Author.FullName</span>

        </div>
    </div>
}
```

Obrázek 11: Ukázka pohledu *GetPosts.cshtml*, Zdroj: vlastní tvorba

3.2.3 Implementace manifestu

Po vytvoření funkční klasické webové aplikace je třeba aplikaci postupně transformovat na PWA. Prvním a zároveň nejjednodušším krokem je vytvoření webového aplikačního manifestu. Ten se nachází v kořenové složce *„wwwrooot“* a jsou v něm definované vlastnosti popsané v teoretické části této práce. Pro vytvoření manifestu lze využít na internetu několik online generátorů tohoto souboru nebo nástroj PWABuilder. Definice tohoto souboru je nicméně natolik jednoduchá, že byl manifest vytvořen ručně, bez použití generátorů. Vytvořený soubor je pak potřeba nalinkovat do hlavičky v souboru *„_Layout.cshtml“*, která představuje základní šablonu pro vzhled celé webové aplikace.

3.2.4 Implementace skriptu Service Worker

Druhým krokem k transformaci webové aplikace na PWA je vytvoření skriptu Service Worker. V tomto skriptu budou k různým typům requestů zvoleny různé cachovací strategie, aby bylo docíleno vhodného chování aplikace i bez připojení k internetu.

Aby mohl být Service Worker u klienta používán, je nutné nejprve zjistit, zda klientský prohlížeč podporuje službu Service Worker. V případě že ano, zavolá se metoda, která Service Workera zaregistruje. Kód s touto kontrolou a registrací se nachází v souboru „*site.js*“, který je vložen do stránky „*_Layout.cshtml*“. Ta tvoří šablonu všech stránek webové aplikace a díky tomu je možné provést tuto kontrolu a registraci z jakékoliv stránky, kterou uživatel navštíví.

Skript služby Service Worker je pojmenován jako service-worker.js a je uložen v adresáři wwwroot. Tento adresář ve struktuře ASP.NET Core projektu představuje kořenový adresář webové aplikace, takže je tím docíleno toho, že bude služba dostupná pro veškeré stránky i oblasti, které se v projektu nachází.

Během instalace služby jsou do paměti cache nahrány soubory, které mají být dostupné bez připojení k internetu. Soubory, které se mají do cache nahrát, jsou definovány v poli *urlsToCache* prostřednictvím relativních URL adres k těmto souborům. Během instalace služba projde tyto adresy, obdrží odpovědi s potřebnými soubory a uloží je do paměti. Po instalaci služby následuje ještě její aktivace, během které je smazána stará paměť cache.

Dalším krokem v implementaci kódu služby Service Worker je vytvoření tzv. „*eventListeneru*“ pro zachycování síťových požadavků. V ukázkové aplikaci je použito několik strategií, které zpracovávají různé typy požadavků. Pro každý zachycený request se pomocí podmínek s regulárními výrazy určuje, zda má být obsah servírován nejprve z cache nebo naopak, zda má být obdržená odpověď z internetu uložena do cache, apod. Ukázka jedné ze strategií je zobrazena na obrázku 12.

```
else if (/\/post\/detail/.test(requestUrl.pathname)) {
    event.respondWith(fetch(event.request)
        .then((res) => {
            if (res.status != 404) {
                return caches.open(CACHE_NAME)
                    .then((cache) => {
                        cache.put(event.request.url, res.clone());
                        return res;
                    });
            }
            else {return res;}
        })
        .catch((err) => {
            return caches.match(event.request)
                .then((resp) => {
                    if (resp == undefined) {
                        return caches.match(offlineUrl);
                    }
                    return resp;
                });
        }));
}
```

Obrázek 12: Zpracování requestu na detail příspěvku, Zdroj: vlastní tvorba

V tomto kódu je v podmínce nejprve regulárním výrazem zjištěno, zda URL adresa webového požadavku obsahuje řetězec znaků „/post/detail“, což je část URL adresy, přes kterou se lze dostat k detailnímu pohledu příspěvku na webu. Pokud je podmínka splněna, webový požadavek se odešle na aplikační server, který buď vrátí nebo nevrátí odpověď. Pokud je odpověď vrácena, je ještě nutné zkontrolovat, zda nemá stavový kód roven 404, tedy že požadovaný dokument nebyl nalezen. Tato podmínka je zde z toho důvodu, že webová aplikace používá vlastní chybové stránky, které není nutné ukládat do paměti cache. Pokud tedy odpověď na request nemá stavový kód roven 404, je odpověď v podobě html dokumentu detailu daného příspěvku uložena do paměti cache a zároveň vrácena prohlížeči. Pokud by nebylo možné získat odpověď ze serveru, pokusí se service worker získat odpověď z paměti cache. Pokud i zde odpověď nenajde, je prohlížeči vrácen dokument s offline stránkou. Tímto způsobem lze získávat vždy aktuální detail příspěvku v případě připojení k internetu a v opačném případě získat alespoň jeho starší verzi.

Na stejném principu jsou obsluhovány i požadavky na seznam oblíbených příspěvků uživatele. Díky tomu si uživatel i bez připojení k internetu může procházet své oblíbené příspěvky.

3.2.5 Implementace push notifikací

V základních požadavcích na ukázkovou webovou aplikaci je uvedeno, že budou v aplikaci implementovány push notifikace. Možnost zasílání notifikací bude zpřístupněna pouze přihlášeným uživatelům aplikace. Ti si budou moci notifikace povolit z jakékoliv stránky webu. Implementace push notifikací do webové aplikace je v případě této aplikace řešena ve třech fázích, které budou v této kapitole popsány.

3.2.5.1 Vytvoření odběru notifikací a zpracování odběru

Tlačítko, které uživateli umožní získávat notifikace, je umístěno v souboru *_Layhout.cshtml*, který tvoří šablonu webu. Díky tomu je možné přihlásit se k odběru notifikací odkudkoliv z webu. Následně je potřeba zjistit, zda prohlížeč, přes který uživatel webovou stránku navštěvuje, push notifikace podporuje a jestli si již uživatel zasílání notifikací dříve nepovolil. Tato kontrola je prováděna pomocí JavaScriptu v souboru *site.js*, který je připojen k šabloně webu. Stisknutím tlačítka k odběru notifikací se získají informace o nové registraci odběru notifikací, které jsou následně AJAXem odeslány na aplikační server a uloženy do databáze. Po uložení údajů o odběru je tlačítko schováno.

3.2.5.2 Rozesílání notifikací

Notifikace jsou uživatelům zasílány vždy po vytvoření nového příspěvku. K tomu řadič PostController, ve kterém vytvoření příspěvku probíhá, využívá metodu *SendNotificationsAsync* patřící třídě *SubscriptionService* a třídu *PushNotificationData*, ve které se definují data, která budou později v notifikaci zobrazena.

Metoda *SendNotificationsAsync*, která jako parametr přijímá instanci třídy *PushNotificationData*, je určena k zaslání oznámení všem zaregistrovaným odběratelům. V těle této metody je tedy získán seznam všech záznamů

s registrovanými odběry notifikací. Cyklem *foreach* se všem odběrům v seznamu, za pomoci knihovny WebPush, rozešlou notifikace. Na obrázku 13 je zobrazena výsledná implementace rozesílání notifikací s pomocí knihovny WebPush.

```
1 reference
public async Task SendNotificationsAsync(PushNotificationData pushData)
{
    var subscriptions = context.Subscriptions.ToList();
    if (subscriptions.Any())
    {
        var vapidDetails = new VapidDetails("mailto:admin@skolnikuryr.cz", vapidPublicKey, vapidPrivateKey);
        string payload = ConvertPushDataToPayload(pushData);
        List<AppSubscription> failedSubscriptions = new List<AppSubscription>();
        foreach (var sub in subscriptions)
        {
            try
            {
                var pushSubscription = new PushSubscription(sub.PushEndPoint, sub.PushP256DH, sub.PushAuth);
                var webPushClient = new WebPushClient();
                webPushClient.SendNotification(pushSubscription, payload, vapidDetails);
            }
            catch (WebPushException ex)
            {
                failedSubscriptions.Add(context.Subscriptions
                    .FirstOrDefault(it => it.PushEndPoint == ex.PushSubscription.Endpoint));
                continue;
            }
        }
        if (failedSubscriptions.Any())
        {
            context.RemoveRange(failedSubscriptions);
            context.SaveChanges();
        }
    }
}
```

Obrázek 13: Implementace rozesílání notifikací, Zdroj: vlastní tvorba

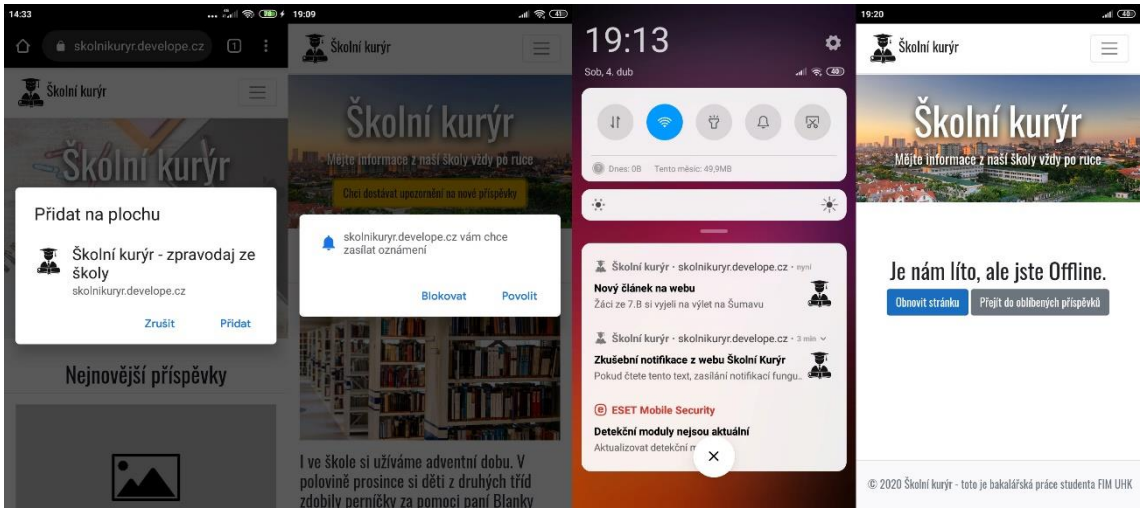
3.2.5.3 Přijímání a zpracování notifikací

Zachycování příchozích notifikací je nutné implementovat ve skriptu Service Worker. Zde se odchytávají *PushEventy*, které obsahují data, jež byla původně definována ve třídě *PushNotificationData* před odesláním notifikace. Z těchto dat je v metodě *showNotification* vytvořena notifikace, která je následně zobrazena v zařízení uživatele. Událost, která nastane po kliknutí na notifikaci, zachycuje opět Service Worker. Tentokrát odchytává událost typu *NotificationEvent*, přes níž má přístup k notifikaci, na kterou uživatel kliknul, a k jejím parametrům. V těchto parametrech se nachází i URL adresa detailu nově vytvořeného příspěvku, na kterou je následně uživatel přesměrován.

3.2.6 Testování výsledné aplikace

Ukázková aplikace byla průběžně testována nástrojem Lighthouse. Na základě výsledků jeho testů byly v aplikaci doplněny nebo opraveny některé prvky. V kategorii PWA bylo například nutné do hlavičky šablony stránky vložit odkaz na ikonu pro zařízení operačních systémů iOS, aby bylo možné přidat aplikaci na plochu i pro tato zařízení.

Jelikož po celou dobu vývoje byla aplikace spouštěna lokálně v počítači, na kterém probíhal vývoj, nemohly být dostatečně otestovány funkcionality, které v aplikaci byly implementovány. Proto musela být webová aplikace publikována pod veřejně dostupnou adresou, odkud k ní bylo možné přistupovat z různých zařízení a testovat tak její funkčnost. K takto publikované aplikaci se přistupovalo především z mobilních zařízení a byly zde testovány funkce jako možnost nainstalovat webovou aplikaci do zařízení, přijímání notifikací či zobrazení vlastní offline stránky v případě, že zařízení nebylo připojeno k internetu. Obrázek 14 zobrazuje zachycené screeny z procesu testování.



Obrázek 14: Screenshoty zachycené během testování aplikace na mobilním zařízení, Zdroj: vlastní tvorba

4 Shrnutí výsledků

V předchozích kapitolách této bakalářské práce byla představena nová metodika vývoje webových aplikací a s ní spjaté technologie. Tyto technologie byly popsány a většina z nich byla použita i při tvorbě referenční aplikace. Po prozkoumání těchto technologií a během práce na ukázkové aplikaci bylo zjištěno, že vývoj progresivní webové aplikace se nijak výrazně neliší od vývoje běžné webové aplikace a že transformovat stávající webovou aplikaci na tu progresivní lze relativně snadno. Je však potřeba brát v úvahu nekompatibilitu především starších verzí prohlížečů s těmito novými technologiemi (především skript Service Worker, rozhraní Push API, apod).

Samotný vývoj ukázkové progresivní webové aplikace probíhal stejně jako u běžné webové aplikace. Nejprve byla vytvořena základní webová aplikace, která byla postupným zaváděním nových technologií spjatými s vývojem PWA převedena na aplikaci progresivní. Výsledná webová aplikace splňovala všechna kritéria pro její označení moderními internetovými prohlížeči jako progresivní webová aplikace. Díky tomu prohlížeče samy vybídnou uživatele, aby si webovou aplikaci nainstaloval do svého zařízení. Pokud internetový prohlížeč progresivní webovou aplikaci nepozná, funguje tato aplikace i nadále, uživatel však ztrácí benefity, které progresivní webové aplikace nabízejí a může aplikaci využívat pouze v prohlížeči. Konkrétní implementací ve skriptu Service Worker je možné ukázkovou aplikaci částečně využívat i bez připojení k internetu. Uživatelům s novějšími prohlížeči je také umožněno přihlásit se k odběru notifikací, takže jim již neunikne žádný nově přidaný příspěvek.

Problém při implementaci Service Workera nastal ve chvíli, kdy bylo nutné rozlišit zobrazení offline stránky pro nepřihlášeného a přihlášeného uživatele. Přihlášeným uživatelům bylo totiž nutné navíc zobrazit odkaz na sbírku jejich oblíbených článků, které si v offline módu mohou prohlédnout. Tento problém byl nakonec vyřešen tak, že Service Worker zachytí požadavky na přihlášení i odhlášení uživatele a po obdržení odpovědi ze serveru navíc odešle požadavek na vrácení offline stránky, ve které je zjišťováno, zda je uživatel přihlášený či nikoliv. Tuto stránku (s tlačítkem nebo bez něj) pak pouze uloží do cache.

5 Závěry a doporučení

Dle zadaných požadavků byla vytvořena ukázková progresivní webová aplikace. Ta byla nejprve vytvořena jako běžná webová aplikace, do které byly postupně implementovány vlastnosti progresivních webových aplikací.

Stěžejními technologiemi pro tvorbu progresivní webové aplikace je webový aplikační manifest, ve kterém se definuje chování aplikace po jejím nainstalování na plochu mobilního zařízení, a služba Service Worker. S využitím Service Worker lze využívat nová webová API podporovaná novějšími prohlížeči, obsluhovat push notifikace a s využitím různých cachovacích strategií zajistit webové aplikaci vlastní offline funkcionalitu.

Webový aplikační manifest byl pro ukázkovou aplikaci vytvořen ručně, ale bylo by možné použít nástroj PWA builder, který po zadání parametrů tento soubor vygeneruje. Navíc po nahrání dostatečně velké ikony aplikace vytvoří i její menší verze, takže by odpadlo i manuální zmenšování ikon pomocí softwarů pro práci s grafikou. Ve skriptu Service Worker by mohla být použita knihovna Workbox, která by pravděpodobně ulehčila a urychlila implementaci cachovacích strategií pro jednotlivé webové požadavky. Pro tvorbu tohoto skriptu však byl použit čistý JavaScript, neboť autor této práce neměl prozatím zkušenosti s jeho implementací a chtěl pochopit, jak jednotlivé části skriptu fungují.

Aplikace byla určena pouze k demonstrování získaných poznatků a nebylo by vhodné aplikaci využít do reálného provozu. Nicméně by mohlo být zajímavé vytvořit plnohodnotné webové stránky pro existující školu s použitím technologií progresivních webových aplikací. Například pomocí push notifikací by mohli být studenti upozorňováni na nadcházející akce pořádané ve škole, na změny úředních hodin studijního oddělení apod. Bez internetového připojení by pak mohly být dostupné například stránky s kontaktními informacemi.

6 Seznam použité literatury

[1] ADERINOKUN, Ire. The Service Worker Lifecycle. *Google developers* [online]. 2016, Jul 19 [cit. 2020-04-06]. Dostupné z: https://bitsofco.de/the-service-worker-lifecycle/

[2] ARCHIBALD, Jake. The Offline Cookbook. *Google Developers* [online]. 2019 [cit. 2020-03-31]. Dostupné z: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook

[3] ARCHIBALD, Jake. The Service Worker Lifecycle. *Google developers* [online]. 2019, 04-08 [cit. 2020-04-06]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle

[4] BAR, Adam. What Web Can Do Today?. *What Web Can Do Today* [online]. 2020, [cit 2020-04-09]. Dostupné z: https://whatwebcando.today/

[5] CACERES, Marcos a kol. Web App Manifest. *World Wide Web Consorcium (W3C)* [online]. Cambridge, 2020, 26 March 2020 [cit. 2020-03-29]. Dostupné z: https://www.w3.org/TR/appmanifest/RUSSEL, Alex. Progressive Web Apps: Escaping Tabs Without Losing Our Soul. *Infrequently Noted* [online]. 2015 [cit. 2019-06-18]. Dostupné z: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

[6] CONTRACTOR, Nishant. Everything you want to know about Service Worker. *DEV Community* [online]. 2020, Feb 28 [cit. 2020-03-30]. Dostupné z: https://dev.to/niscontractor/everything-you-want-to-know-about-service-worker-4c91

[7] GAUNT, Matt. Service Workers: an Introduction. *Web | Google Developers* [online]. 2019, 2019-08-09 [cit. 2020-03-29]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers/

[8] GEDDES, Dave. Service worker mindset. *Web.dev* [online]. 2019, Jun 4, 2019 [cit. 2020-03-29]. Dostupné z: https://web.dev/service-worker-mindset/

[9] JAISWAL, Aayush. Understanding Service Workers and Caching Strategies. *Bits and Pieces* [online]. 2019, Feb 12, 2019 [cit. 2020-03-31]. Dostupné z: https://blog.bitsrc.io/understanding-service-workers-and-caching-strategies-a6c1e1cbde03

[10] MCLACHLAN, Peter. Introducing a Trusted Web Activity for Android. *Chromium Blog* [online]. 2019, February 5 [cit. 2020-04-09]. Dostupné z: https://blog.chromium.org/2019/02/introducing-trusted-web-activity-for.html

[11] MEDLEY, Joseph a kol. Push API. *MDN web docs* [online]. c2005-2020, Dec 13, 2019 [cit. 2020-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Push_API

[12] MEDLEY, Joseph. Web Push Notifications: Timely, Relevant, and Precise. *Google Developers* [online]. 2019, 2019-02-12 [cit. 2020-04-01]. Dostupné z: https://developers.google.com/web/fundamentals/push-notifications

[13] MICHÁLEK, Martin. Lighthouse: Nepostradatelná analýza webu od Google. *Vzhůru dolů* [online]. 2018, 16.9.2018, [cit. 2020-04-02]. Dostupné z: https://www.vzhurudolu.cz/prirucka/lighthouse

[14] MILLS, Chris a kol. Notifications API. *MDN web docs* [online]. c2005-2020, Dec 11, 2019 [cit. 2020-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

[15] MILLS, Chris a kol. Progressive web apps (PWAs). *MDN web docs* [online]. 2020, Apr 1 [cit. 2020-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

[16] MILLS, Chris a kol. Using Service Workers. *MDN Web Docs* [online]. Mountain View, Jul 24, 2014 [cit. 2020-03-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

[17] POSNICK, Jeff. Workbox: your high-level service worker toolkit. *Web.dev* [online]. 2018, Nov 5 [cit. 2020-04-02]. Dostupné z: https://web.dev/workbox/

[18] RAWAT, Girish. How Web Caching Improves Internet Performance. *3Pillar Global* [online]. 2018 [cit. 2020-03-31]. Dostupné z: https://www.3pillarglobal.com/insights/how-web-caching-improves-internet-performance

[19] RUSSEL, Alex. Progressive Web Apps: Escaping Tabs Without Losing Our Soul. *Infrequently Noted* [online]. 2015 [cit. 2019-06-18]. Dostupné z: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

[20] RUSSELL, Alex a kol. Service Workers Nightly. *3Pillar Global* [online]. 2020, 24 February 2020 [cit. 2020-03-31]. Dostupné z: https://w3c.github.io/ServiceWorker/#cache-put

[21] SCHOLZ, Florian a kol. FetchEvent. *MDN web docs* [online]. Mountain View, 2014, Nov 26 [cit. 2020-03-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent

[22] THOMSON, Martin a Peter BEVERLOO. Voluntary Application Server Identification for Web Push. *IETF Tools* [online]. 2016, June 29 [cit. 2020-04-04]. Dostupné z: https://tools.ietf.org/html/draft-ietf-webpush-vapid-01

[23] THOMSON, Martin, Elio DAMAGGIO a Brian RAYMOR. *Generic Event Delivery Using HTTP Push* [online]. 2015, 2015-04-15 [cit. 2020-04-01]. Dostupné z: https://tools.ietf.org/html/draft-thomson-webpush-protocol-00

[24] THOMSON, Martin. Message Encryption for Web Push. *IETF Tools* [online]. 2017, September 04 [cit. 2020-04-04]. Dostupné z: https://tools.ietf.org/html/draft-ietf-webpush-encryption-09

7 Seznam použitých obrázků

8 Přílohy

1) Zadání bakalářské práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Máša Stanislav	Podélná 867, Sedlčany	I1600574

TÉMA ČESKY:

Progresivní webové aplikace

TÉMA ANGLICKY:

Progresive web applications

VEDOUCÍ PRÁCE:

Mgr. Daniela Ponce, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit dynamické webové stránky pro středně velkou organizaci s využitím metodiky vývoje progresivních webových aplikací. Student popíše metodiku, porovná PW aplikace s aplikacemi nativními a webovými, vyhledá a popíše dostupné nástroje. Poté navrhne a realizuje vlastní demonstrativní progresivní webovou aplikaci dle svého uvážení.

1. Úvod
2. Principy progresivních webových aplikací
3. Vlastnosti progresivních webových aplikací
4. Dostupné nástroje pro vývoj PWA
5. Vývoj vlastní aplikace
6. Shrnutí výsledků
7. Závěry a doporučení

SEZNAM DOPORUČENÉ LITERATURY:

Mark J., C# 7.1 and .NET Core 2.0 - modern cross-platform development. Birmingham, Packt (2017) ISBN 978-1-78839-807-7

Freeman, Adam. Pro ASP.NET Core MVC. Berkeley, CA: Apress (2016) ISBN 978-1-4842-0397-2

Sheppard, Dennis. Beginning progressive web app development : creating a native app experience on the web. United States: Apress (2017) ISBN 978-1-4842-3090-9

Podpis studenta: Matěj Datum: 18.7.2019

Podpis vedoucího práce: D. Bauce Datum: 18.7.2019