Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v ČB

Pedagogická fakulta

Katedra informatiky

Off-line web v Progressive Web Apps s využitím Service workeru.

Offline website in Progressive Web Apps using Service worker.

Bakalářská práce

Vypracoval: Matěj Vachuta

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2023

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Matěj VACHUTA**
Osobní číslo:	**P20502**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Téma práce:	**Off-line web v Progressive Web Apps s využitím Service workeru.**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování

Cílem bakalářské práce je popsat a zpracovat technologii PWA (Progressive Web Apps) a možnosti využití Service workeru při tvorbě tzv. progresivních webových aplikací. PWA je technologie umožňující tvorbu webové aplikace, která využívá webového prohlížeče a zároveň umožňuje takzvaný cross-platform přístup. Service worker je JavaScriptový asset, který se chová jako proxy server a je tedy nutnou složkou při tvorbě progresivních webových aplikaci. Vytvořená aplikace má pak vlastnosti klasické webové stránky, avšak hlavní výhodou jsou i vlastnosti nativní aplikace. Technologie PWA využívá HTML a CSS k tvorbě vzhledové stránky aplikace a JavaScript pro implementaci Service workeru.
V teoretické části se autor zaměří především na představení samotné technologie PWA, výhody a nevýhody využití PWA, implementaci Service workeru pomocí JavaScriptu, popsání jednotlivých komponent a obecné syntaxe, jež je nutná pro tvorbu aplikace.
V praktické části bude vytvořena progresivní webová aplikace s využitím Service workeru, která bude demonstrovat možnosti, vlastnosti a výhody či nevýhody, které technologie PWA přináší. Součástí praktické části bude i průzkum rozšíření PWA v současné době v poměru k běžným webovým stránkám.

Rozsah pracovní zprávy:	**40**
Rozsah grafických prací:	**interaktivní modely**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. How to build a Progressive Web App | Creative Bloq. CreativeBloq | Art and Design Inspiration [online]. Copyright [cit. 04.04.2019]. Dostupné z: https://www.creativebloq.com/how-to/build-a-progressive-web-app
2. Richard, Sam a Pete LEPAGE. What are Progressive Web Apps?. web.dev [online]. 1 [cit. 2022-03-30]. Dostupné z: https://web.dev/what-are-pwas/
3. Co jsou progresivní webové aplikace (PWA) a jaké mají výhody [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody
4. Co je to vlastně PWA? [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.bootiq.io/progressive-web-application-pwa/
5. Progresivní webové aplikace: Co to je? A jak webu zařídit plné hodnocení PWA v Lighthouse. Vzhůru dolů – webová kodéřina ze všech stran [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.vzhurudolu.cz/prirucka/pwa
6. Gattermayer, Josef. Proč a jak psát progresivní webové aplikace [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.ackee.cz/blog/proc-a-jak-psat-progresivni-webove-aplikace

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **4. dubna 2022**
Termín odevzdání bakalářské práce: **30. dubna 2023**

L.S.

doc. RNDr. Helena Koldová, Ph.D.
děkanka

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 4. dubna 2022

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 15. dubna 2023.

Matěj Vachuta

Abstrakt/Anotace

Cílem bakalářské práce je popsat a zpracovat technologii PWA (Progressive Web Apps) a možnosti využití Service workeru při tvorbě tzv. progresivních webových aplikací. PWA je technologie umožňující tvorbu webové aplikace, která využívá webového prohlížeče a zároveň umožňuje takzvaný cross-platform přístup. Service worker je JavaScriptový asset, který se chová jako proxy server a je tedy nutnou složkou při tvorbě progresivních webových aplikaci. Vytvořená aplikace má pak vlastnosti klasické webové stránky, avšak hlavní výhodou jsou i vlastnosti nativní aplikace. Technologie PWA využívá HTML a CSS k tvorbě vzhledové stránky aplikace a JavaScript pro implementaci Service workeru.

V teoretické části se autor zaměří především na představení samotné technologie PWA, výhody a nevýhody využití PWA, implementaci Service workeru pomocí JavaScriptu, popsání jednotlivých komponent a obecné syntaxe, jež je nutná pro tvorbu aplikace.

V praktické části bude vytvořena progresivní webová aplikace s využitím Service workeru, která bude demonstrovat možnosti, vlastnosti a výhody či nevýhody, které technologie PWA přináší. Součástí praktické části bude i průzkum rozšíření PWA v současné době v poměru k běžným webovým stránkám.

Klíčová slova

PWA(Progressive Web Apps), HTML5, CSS, JavaScript, Service Worker

Abstract

The aim of the bachelor thesis is to describe and elaborate the PWA (Progressive Web Apps) technology and the possibilities of using Service worker in the creation of so-called progressive web applications. PWA is a technology that enables the creation of a web application that uses a web browser and at the same time allows a so-called cross-platform approach. Service worker is a JavaScript asset that acts as a proxy server and is therefore a necessary component in the creation of progressive web applications. The created application then has the properties of a classic web page, but the main advantage is also the properties of a native application. The PWA technology uses HTML and CSS to create the appearance of the application and JavaScript to implement the Service worker.

In the theoretical part, the author will focus mainly on the introduction of the PWA technology itself, advantages and disadvantages of using PWA, implementation of Service worker using JavaScript, description of individual components and general syntax that is necessary for creating an application.

In the practical part, a progressive web application will be created using Service worker to demonstrate the possibilities, features and advantages or disadvantages that PWA technology brings. The practical part will also include an exploration of the prevalence of PWAs today in relation to conventional websites..

Keywords

PWA(Progressive Web Apps) , HTML5, CSS, JavaScript, Service Worker

Obsah

1 Úvod

Bakalářská práce se bude zabývat tvorbou Progressive Web Apps (PWA) s využitím service workeru a klasických scriptovacích jazyků HTML5 a CSS pro tvorbu vzhledové stránky. Práce se taktéž bude zabývat výhodami a nevýhodami PWA.

1.1 Východiska práce

Progressive Web Apps (PWA) je moderní a velmi nová technologie, která by mohla v budoucnu zcela nahradit nativní aplikace a webové stránky. Tato technologie využívá to nejlepší z obou světů. Jednou z jejích předností je například offline fungování, které u webových stránek v současnosti není možné.

PWA je tedy něco mezi nativní aplikací a webovou stránkou. Tato technologie by měla řešit problémy obou "předchůdců". Tím, že PWA vychází z dvou různých světů, přejímá z nich i to nejlepší. Má vlastnosti nativní aplikace, může se tedy stáhnout na jakékoliv zařízení a pomocí prohlížeče, ve kterém se otevírá, řeší i nynější problémy s kompatibilitou operačních systémů, ať už se jedná o počítače či mobilní zařízení. A to není vše, co tato technologie přináší a nabízí.

1.2 Cíle práce

Cílem práce bude představit a otestovat technologii Progressive Web Apps(PWA), která by mohla být budoucností, jež nahradí klasické webové stránky a nativní aplikace. Technologie jako taková zahrnuje spoustu velkých výhod a přebírá vlastnosti obou předchůdců.

PWA využívá k chodu service worker, který umožňuje offline funkcionalitu aplikace. Service worker vytváří na konkrétním zařízení proxy server, jež v prohlížeči uchovává data, která by za normálních okolnostní musela být všechna uložena na nějakém hostingovém serveru nebo přímo v zařízení. Výhodou je, že množství dat uložených v prohlížeči je rapidně menší než množství dat, které by potřebovala nativní aplikace stáhnout na zařízení pro správné fungování.

Základ práce bude seznámení s technologii Progressive Web Apps a prozkoumání jejích možností, výhod a nevýhod a jejich představení a případná demonstrace. Dalším cílem bude i bližší seznámení se service workerem, HTML5 a CSS, které jsou taktéž velice důležitou složkou PWA pro její úspěšné nasazení a funkcionalitu.

Cílem praktické časti bude samotné vytvoření PWA s použitím serviceworkeru a demonstrace možností a vlastností samotné aplikace, případná ukázka výhod a nevýhod na konkrétních příkladech.

1.3 Metody práce

V úvodu představím pojmy PWA a Service worker, zmíním výhody a nevýhody PWA, příklady užití PWA a vlastnosti samotné aplikace. Dále porovnám PWA s Webovými stránkami a nativními aplikacemi. Následně vytvořím jako konkrétní příklad aplikaci, na které budu demonstrovat její vlastnosti, výhody a nevýhody. V závěru uvedu zjištěné poznatky, které budou zahrnovat vlastnosti, výhody a nevýhody, jež jsem sám byl schopen zjistit a otestovat.

2 Technologie užité k tvorbě PWA

2.1 HTML

HTML (HyperText Markup Language) je nejzákladnějším prvkem, který slouží ke tvorbě webových stránek. Pomocí HTML tvoříme strukturu webu. Pro tvorbu webů se využívá především v kombinaci s dalšími technologiemi, které se starají o další aspekty. CSS se stará o stylizaci a funkční část zastupuje například JavaScript. Z názvu technologie dostaneme dva pojmy, Hypertext a Markup.[1]

„HYPERTEXT“ zastupuje propojení jednotlivých stránek s jinými. Jednat se může o interní spojení, tedy spojení které je v rámci jednoho webu, nebo o spojení externí, kdy web odkazuje na jiný web v rámci internetu.[1]

„MARKUP“ znázorňuje využití značek, které slouží k tomu, aby prohlížeč věděl, jakým způsobem jednotlivé texty a obrázky interpretovat. HTML označení obsahuje speciální elementy, které jsou tvořeny pomocí tagů.[1]

Každá informace má přesně definovaný význam, který vyjadřuje konkrétní prvek (element). Prvek definuje počáteční a koncová značka (tag). Počáteční značka může obsahovat atributy, které jsou tvořený příslušným názvem a hodnotou.[2]

2.1.1 HTML5

Vývoj specifikace začal v roce 2004 pod pracovní skupinou WHATWG (Web Hypertext Application Technology Working Group), tedy zcela mimo W3C. Skupina byla založena lidmi z Apple, Mozilla Foundation a Opera Software, kteří chtěli rozvíjet HTML a zároveň se chtěli vyhnout technologii XHTML 2.[3]

Specifikace HTML5 je velmi podstatným rozšířením jazyka pro tvorbu internetových stránek. Do jejího vydání totiž nebylo možné například uploadovat na server větší soubor, než server dovoloval. Do HTML5 bylo přidáno několik

nových API.[4]

- FILE API

  Umožňuje jak nahrávat neomezené množství souborů najednou, tak například i rozřezání velkého souboru na straně klienta a postupný upload těchto částí na server, kde se serverový script postará opět o jeho složení.[4]

- FULSCREEN API

  Toto rozhraní odstraňuje nutnost používat Flash při nutnosti přepnout dokument přes celou obrazovku. To se využívá například při přehrávání videí nebo při prohlížení fotografií na sociálních sítích.[4]

- GEOLOCATION API

  Je schopno požádat o sdělení vaší globální pozice. Ta je vypočítávána ze všech možných známých parametrů, na nichž pochopitelně závisí její přesnost.[4]

```
<!DOCTYPE html>
<head>
    <title>Example</title>
</head>
<body>
     <h1>Hello world!</h1>
</body>
</html>
```

Příklad 1: ukázka HTML

2.2 CSS

CSS (Cascading Style Sheets) je jazyk, který slouží k popsání prezentace webových stránek, včetně barev, rozložení a fontů písma. Umožňuje nám adaptaci prezentací pro různá zařízení, jako jsou například počítače, tablety, mobilní

zařízení nebo tiskárny. CSS není přímo spojené s HTML a je možné jej využít s jakýmkoliv jazykem, který je založený na XML.[5]

2.2.1 CSS3

Významnou změnou v CSS3 oproti CSS2 je zavedení modulů. Výhodou této funkce je bezesporu, že umožňuje rychlejší a snadnější dokončení a přijetí specifikace, protože segmenty jsou dokončovány a přijímány po částech. To také umožňuje prohlížeči podporovat segmenty specifikace.[6]

Mezi klíčové moduly CSS3 patří:

- Box model
- Text effects
- Selectors
- Background and borders
- Animations
- UI
- Multiple column layout
- 2D/3D transformation

Mezi výhody CSS3 patří schopnost zajistit konzistentní a přesné pozicování navigovatelných prvků. Dále snadné přizpůsobení webové stránky, protože jej lze provést pouhou změnou modulárního souboru. Nesporná výhoda je i zobrazování online videí bez použití modulů třetích stran.[6]

```css
body{
    text-align: center;
    background-color: red;
    margin: 0;
}

h1{
    font-size: 20px;
}
```

Příklad 2: ukázka CSS

2.3 JavaScript

JavaScript (JS) je skriptovací jazyk učený pro tvorbu moderních dynamických webů. Byl představen na konci 20. století, konkrétně v 90. letech, jako reakce na klasické statické internetové stránky a díky svým schopnostem a funkcím otevřel prostor pro interaktivní webové aplikace i příjemnější uživatelské rozhraní doplněné animacemi a samozřejmě také 2D i 3D grafikou. JavaScript je především určen ke zhotovování klientských částí aplikací, ale s příchodem Node.js a podobných technologií však dokáže plnohodnotně figurovat i na straně serveru a obecně backendu.[7]

Programy psané v JavaScriptu se nazývají skripty a zapisují se buď přímo do HTML kódu nebo jako samostatně fungující soubory, které se následně do HTML importují. Díky jeho lehčí syntaxi a funkčnosti si základy JavaScriptu mohou osvojit i úplní začátečníci. V minulosti se JS potýkal především s problémy ohledně nedostatečné kompatibility prohlížečů, v důsledku čehož se stránky uživatelům nezobrazovaly tak jak by měly.[7]

Dnes je ovšem součástí drtivé většiny všech webů, přičemž jeho popularita neustále strmě roste a to i díky takzvaným javascriptovým frameworkům, jako je například React, Vue nebo Angular. S postupem času JavaScript dokonce rozšířil své pole působnosti i na vývoj mobilních aplikací, kde prostřednictvím

technologií, jako je React Native, umožňuje současnou tvorbu aplikací pro více platforem – obvykle Android a iOS.[7]

JavaScript je velmi často zaměňován s Javou. Java je ovšem samostatný programovací jazyk. Má s JavaScriptem pouze lehce podobnou syntaxi.[8] Rychlost je první předností JavaScriptu. Mezi jeho další výhody patří plná integrace s HTML a CSS, rychlejší interakce a dynamické načítání stránky. [7]

Mezi další výhody JavaScriptu patří nepochybně jeho velká rozšířenost, široká nabídka frameworků, jež velmi usnadňují tvorbu opakovaných částí webů i webových aplikací a databáze API, která usnadňuje do aplikací integrovat funkcionalitu nástrojů třetích stran. [7]

První nevýhodou je potenciální možnost, že váš kód bude zneužit, protože kód je volně dostupný v prohlížeči.[7]

Jako další nevýhodu můžeme brát riziko nekompatibility nebo zákazu JS v prohlížeči, což má za důsledek špatné zobrazení stránky.[7][8]

```
function HelloWorld() {
    console.log("Hello world");
}

//tlacitko ktere po kliknuti zavola HelloWorld => vypise Hello world do console
btn.addEventListener("click",()=>{
    HelloWorld();
})
```

Příklad 3: ukázka JS

2.4 JSON

JSON = JavaScript Object Notation neboli JavaScriptový zápis objektů, je formát sloužící pro výměnu dat, který se během posledních několika let zařadil mezi nejdůležitější formáty na webu. Navrhl jej Douglas Crockford.[9]

JSON je založený na syntaxi jazyka JavaScript. Přestože se velmi po-

dobá objektové syntaxi JavaScriptu, lze jej používat nezávisle na JavaScriptu. Mnoho programovacích prostředí má možnost JSON číst (analyzovat) a generovat.[10]

JSON se objevil ve chvíli, kdy se na webu pro výměnu dat používal především formát XML. Ten však podle komunity srocené okolo JS trpěl různými nedostatky, například práce s ním byla dosti složitá, jelikož bylo nutné používat těžkopádný DOM [9]

JSON nemůže při zápisu celých dokumentů konkurovat formátu XML, nebyl k tomu navržen. Jeho hlavní síla spočívá především v zápisu krátkých strukturovaných dat, které si vyměňují webové aplikace, a proto konkurenční boj vyhrál JSON. [9]

Zápis JSON je platným zápisem jazyka JavaScript. To je jedna z jeho výhod. Jsme tedy schopní již ze samotného pohledu na zápis v JSON pochopit, jak s ním budeme v programu pracovat. [9]

```
[
  {"brand": "Ford", "typeName": "Bronco"},
  {"brand": "Opel", "typeName": "Astra"},
  {"brand": "Toyota", "typeName": "Yaris"}
]
```

Příklad 4: ukázka JSON

3 Jednotlivé komponenty PWA

3.1 index.html

Webové stránky jsou vytvořeny v adresářích na webovém serveru. Pro webové stránky je nutností každou stránku uložit jako samostatný soubor. Například stránka "O nás"může být uložena jako about.html a stránka "Kontakt"jako contact.html.[11]

Někdy se stane, že někdo navštíví webové stránky, aniž by v adrese URL uvedl některý z těchto konkrétních souborů(www.stranka.cz/”onas.html”). I když v požadavku na adresu URL odeslaném na server není uvedena žádná stránka, webový server musí pro tento požadavek vždy nějakou stránku doručit, aby měl prohlížeč co zobrazit. Soubor, který bude doručen, je defaultní stránka pro daný adresář. V podstatě se dá říct a obecně platí, že pokud není požadován žádný soubor, server přesně ví, který soubor má v defaultním nastavení zobrazit. Na většině webových serverů se defaultní stránka v adresáři jmenuje ” index.html “.[11]

3.1.1 Členění HTML souboru

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Cleneni html</title>
  </head>
  <body>
    <h1>Ahoj<h1>
    <p>Lorem ipsum<>
  </body>
</html>
```

Příklad 5: členění HTML

<!DOCTYPE html> - Značka je pro HTML5 povinná a měla by být vždy na prvním místě dokumentu HTML. Pomáhá prohlížeči zjistit, jakou verzi jazyka HTML používáte. Prohlížeč jej rozpozná i při psaní malých nebo velkých písmen, ale je doporučeno ,aby byla značka zapsána přesně jako <!DOCTYPE html>.[12][13]

<html lang=”en”></html> - Může se zdát, že vkládat značku <html> hned za <!DOCTYPE html> je zbytečné , ale tato značka slouží k jinému účelu. Možná jste si všimli, že <!DOCTYPE html> neobsahuje uzavírací značku. Je to proto, že prohlížeč nepotřebuje žádné další informace. Značka <html> je však párová a uzavírací značku obsahuje. <html> obsahuje veškerý kód stránky. Poslední značkou na vaší stránce by dozajisté měla být </html>. Značka html může mít také různé atributy, které prohlížeči sdělují více informací o vašem HTML. Například atribut "lang"sděluje prohlížeči, v jakém jazyce je váš obsah, což take může vyhledávači pomoct nasměrovat uživatele na stránky v jejich jazyce. [12][13]

<head></head> - Hlavička dokumentu HTML obsahuje mnoho informací nejen pro prohlížeč, ale i pro vyhledávače. Obsah hlavičky tvoří klíčová slova, popisy, nadpis a meta značky. Vše v této části, kromě značky title, není pro uživatele viditelné.[12]

<meta charset ="utf-8» - V jazyce HTML5 jsou metaznačky považovány za prázdné prvky, což znamená, že nemohou obsahovat žádný obsah, takže není nutné používat uzavírací značku. Metaznačky pomáhají vyhledávačům najít váš web a poskytují informace o vaší webové stránce. Pravděpodobně nejdůležitější metaznačka, kterou je třeba tady uvést, je znaková sada a měla by být vždy na prvním řádku v prvku head.Tato metaznačka má atribut "charset". Téměř vždy použijete v atribut "utf-8", což je označení pro Unicode neboli "univerzální abeceda". Jedná se o sadu znaků, která pokrývá téměř všechny

systémy psaní na světě. Atribut "utf-8"umožňuje, aby se všechny vaše speciální znaky, jako jsou diakritické znaménka, uvozovky nebo dokonce pomlčky, zobrazovaly správně.[12]

<meta name="viewport"content="width=device-width» - Tento prvek zajišťuje vykreslení stránky na šířku zařízení a zabraňuje mobilním prohlížečům vykreslovat stránky širší než je šířka zařízení.[13]

<title></title> - Do značky title se vkládá název stránky. Technicky vzato je title jediným prvkem, který musí být v hlavičce. Text, který vložíte do značky title, se obvykle zobrazuje v titulkovém řádku prohlížeče a je výchozím názvem pro záložky. Je to také první způsob, jak vyhledávače katalogizují stránky. [12]

<body></body> - Poslední značka, která je pro dokument HTML nezbytná, je značka body. Sem přidáte veškerý možný obsah, který uživatel uvidí. Například záhlaví, navigace a odstavce. [12][13]

3.2 style.css

Soubor style.css nám popisuje, jak budou HTML elementy zobrazovány v prohlížeči. Ve style.css budeme k elementům z HTML přistupovat pomocí různých selektorů a následně těmto elementům můžeme nastavova různá zarovnání, barvu, velikost textu, šířku, výšku a tak dále. V tomto souboru taktéž můžeme nastavoval rozložení (layout) stránky a elementů.[14]

Syntaxe je velmi prostá k elementum přistupujeme pomocí selektorů. Následně pro daný element deklarujeme vlastnosti s určitými hodnotami.[14]

```
/*
h1 = selektor
font-size, color = vlasnosti
20px, red = hodnoty
*/
h1{
    font-size: 20px;
    color: red;
}
```

Příklad 6: style.css příklad

3.3 serviceworker.js

Service workery v podstatě fungují jako proxy servery, které jsou umístěny mezi webovými aplikacemi, prohlížečem a sítí (pokud je k dispozici). Jejich úkolem je mimo jiné také umožnit vytvoření efektivního offline prostředí, zachytit síťové požadavky a následně provést příslušnou akci na základě toho, zda je síť dostupná nebo ne a popřípadě aktualizovat prostředky umístěné na serveru.[15]

Service workery jsou taktéž vylepšením pro existující webové stránky. To znamená, že pokud skupina uživatelů prohlížečů, které nepodporují service workery, navštíví webové stránky které service workery používají, nedojde k žádnému narušení základní funkčnosti stránky.[16]

Service worker má podobu JavaScriptového souboru, který může řídit webové stránky, k nimž je přiřazen a zároveň zachycovat a upravovat požadavky na navigaci a ukládaní zdrojů do mezipaměti.[15]

Než service worker převezme kontrolu nad vaší stránkou, musí být nejprve zaregistrován pro vaši PWA. To znamená, že když uživatel poprvé přijde na vaši PWA, síťové požadavky budou nejprve odeslány přímo na váš server, protože service worker ještě nemá kontrolu nad vašimi stránkami.[17]

Po zkontrolování kompatibility vašeho prohlížeče a Service Workeru, může

následně vaše PWA zaregistrovat Service Worker. Po načtení Service Worker vytvoří spojení mezi vaší PWA a sítí.[16]

3.4 manifest.json

Manifest webové aplikace je soubor typu JSON, který informuje prohlížeč o vaší progresivní webové aplikaci. Zejména o tom, jak se má chovat po instalaci na počítači nebo mobilním zařízení uživatele. Typicky soubor manifest obsahuje název aplikace, ikony, které by měla aplikace používat, a adresu URL, která by se měla otevřít po spuštění aplikace.[18]

3.4.1 Klíčové vlastnosti

- **short_name a name**

  Musí se zadat buď vlastnost short_name nebo name, nejlépe však obě. Pokud jsou uvedeny obě, short_name se použije na domovské obrazovce uživatele, v launcheru nebo na jiných místech, kde může být omezený prostor, přičemž name se použije především při instalaci samotné PWA.[18][22]

- **Icons**

  Je zde také možnost definovat sadu ikon, které má prohlížeč používat na domovské obrazovce, ve spouštěči aplikací a přepínači úloh.[18][22] ikony jsou pole objektů, které tvoří obrázky. Každý objekt musí obsahovat src, sizes (velikost) a typ obrázku.[18][22]

- **id**

  Vlastnost id představuje identitu PWA pro prohlížeč. Pokud prohlížeč uvidí manifest, který nemá identitu shodnou s již nainstalovanou PWA, bude jej považovat za novou PWA. Pokud však uvidí manifest s identitou, která odpovídá již nainstalované PWA, bude s ní zacházet jako

s nainstalovanou PWA.[20][22] Vlastnost id umožňuje explicitně definovat identifikátor používaný pro vaši aplikaci. Přidání vlastnosti id do manifestu odstraňuje závislost na start_url nebo umístění manifestu a umožňuje jeho budoucí aktualizaci.[18][22]

- **start_url**

  Člen start_url je řetězec, který představuje počáteční URL webové aplikace - preferovanou URL, která by se měla načíst při spuštění webové aplikace uživatelem[19][22]

- **background_color**

  Člen background_color definuje zástupnou barvu pozadí, která se zobrazí na stránce aplikace před načtením jejího souboru stylů (css). Tuto hodnotu používá uživatelský agent k vykreslení barvy pozadí zástupce za předpokladu, že je manifest k dispozici před načtením souboru stylů.[22] Vlastnost background_color se používá na úvodní obrazovce při prvním spuštění aplikace na mobilním telefonu. [18][22]

- **display**

  Display je řetězec, který určuje vývojáři preferovaný režim zobrazení webové stránky. Režim zobrazení mění, jak velká část uživatelského rozhraní prohlížeče se uživateli zobrazí.[22]

  Můžete si přizpůsobit uživatelské rozhraní prohlížeče, které se zobrazí při spuštění aplikace. Například je možné skrýt adresní řádek a prvky uživatelského rozhraní prohlížeče. Hry lze dokonce nastavit tak, aby se spouštěly přes celou obrazovku.[18][22]

- **scope**

  Scope je řetězec, který definuje navigační rozsah aplikačního kontextu této webové aplikace. Omezuje, které webové stránky lze zobrazit, když je manifest aplikován. Pokud uživatel přejde mimo tento rozsah, vrátí se na běžnou webovou stránku uvnitř karty nebo okna prohlížeče.[21][22]

- **theme_color**

  Funkce theme_color nastavuje barvu panelu nástrojů a může se projevit v náhledu aplikace.[18][22]

- **description**

  Vlastnost description popisuje účel vaší aplikace[18][22]

4 PWA

Co je to PWA? Jednoduše řečeno, PWA je webová stránka se všemi výhodami aplikace. PWA poskytují rychlejší, spolehlivější a poutavější verzi vašich webových stránek nebo e-shopu.[23]

Podrobněji můžeme říct, že Progresivní webové aplikace jsou aplikace, které jsou vytvořené pomocí moderních API. PWA přináší větší spolehlivost, více možností, lepší instalovatelnost. Největší výhodou je možnost instalovat PWA jak na PC tak na ostatních zařízeních jak jsou mobilní telefony a tablet, přičemž je jedno jaký operační systém zařízení má. Toto všechno je možné za pomocí jedné kódové základny, není tedy nutno přepisovat aplikace do různých jazyků pro různé OS.[24]

PWA umí velkou většinu věcí, které umí nativní aplikace, jako například pracovat offline, v případě potřeby přistupovat k fotoaparátu a mikrofonu, GPS a mnoho dalšího.[39]23

Důvodů, proč používat progresivní webovou aplikaci je tu více, ale zde jsou některé z hlavních možností, které poskytuje:

- **Rychlost**

  PWA jsou velmi rychlé a poskytují plynulý zážitek.[24]

- **Integrované UX**

  PWA se chovají velmi podobně jako nativní aplikace, proto je pro uživatele velmi snadné s nimi pracovat. Po instalaci se nacházejí stejně jako nativní aplikace na domovské obrazovce zařízení, posílají push notifikace jako nativní aplikace. Mohou přistupovat k fotoaparátu, poloze zařízení a mnohem vice.[24]

- **Spolehlivost**

  I když vypadne internet, pomocí Service workeru je PWA schopna fungovat dále bez výpadku.[24]

Základní výhody PWA

Podobné nativním aplikacím

Po instalaci na zařízení fungují PWA stejně jako ostatní aplikace.

- PWA mají vlastní ikony, které lze přidat na domovskou obrazovku nebo hlavní panel zařízení.[25]
- PWA se mohou spouštět automaticky při otevření přidruženého typu souboru.[25]
- PWA se mohou spouštět po přihlášení uživatele.[25]

Pokročilé možnosti

PWA mají přístup k pokročilým funkcím.

- PWA mají možnost pokračovat v práci, když je zařízení offline.[25]
- PWA podporují push notifikace (oznámení).[25]
- PWA mohou provádět pravidelné aktualizace i když není aplikace právě spuštěna.[25]
- PWA jsou schopny přistupovat k hardwarovým funkcím.[25]

Výhody související s webem

PWA mohou běžet ve webových prohlížečích stejně jako webové stránky což jim přináší tyto výhody:

- PWA mohou být indexovány vyhledávači.[25]
- PWA lze sdílet a spouštět ze standardního webového odkazu jako klasické webové stránky.[25]
- PWA jsou pro uživatele bezpečné, protože používají zabezpečené koncové body HTTPS a další uživatelská ochranná opatření.[25]

- PWA se přizpůsobují velikosti nebo orientaci obrazovky uživatele.[25]
- PWA mohou využívat pokročilé webové API, například WebBluetooth, WebUSB, WebPayment, WebAuthn nebo WebAssembly.[25]

Nižší náklady na vývoj

PWA mají mnohem nižší náklady na vývoj napříč platformami než kompilované aplikace, které vyžadují specifickou a samostatnou kódovou základnu pro každou platformu.[25]

V případě PWA můžete použít jedinou kódovou základnu, která je sdílená mezi webovou stránkou, mobilní aplikací a aplikací pro stolní počítače (napříč operačními systémy).[25]

Základní princip fungování PWA

Progresivní webové aplikace (PWA) jsou vytvářeny za pomoci HTML, CSS a JavaScriptu. Scripty a kódy jsou umístěny na webových serverech a jsou spouštěny ve webových prohlížečích. Lze je používat buď přímo ve webovém prohlížeči jako webové stránky, nebo je zde možnost nainstalovat PWA do zařízení pomocí funkce instalace aplikace v podpůrném prohlížeči.[25][26]

4.1 PWA vs. webové aplikace

V posledních několika letech prudce vzrostl počet uživatelů mobilních zařízení a s tím vzrostla i potřeba lepšího webového prostředí pro mobilní telefony. Uživatelé dnes využívají telefony k nejrůznějším činnostem. Například nakupování, objednávání jídla atd., tím pádem je logické jim tyto činnosti co nejvíce zjednodušit.[27][29][30]

Běžné webové stránky však tento zážitek nemohou poskytnout, a to z mnoha důvodů, od pomalého načítání až po nepřehledné uživatelské rozhraní. Zde přichází myšlenka webových aplikací, které nabízejí příjemnější uživatelský zážitek bez ohledu na používané zařízení a prohlížeč. [27][29][30]

Co je to vlastně webová aplikace ?

Zjednodušeně řečeno se dá říci, že webová aplikace je webová stránka vytvořená tak, aby její obsah byl přizpůsobený pro všechny obrazovky bez ohledu na zařízení, na kterém se zobrazuje. Webová aplikace se vytváří pomocí front-endového technologického balíčku, který zahrnuje především HTML, CSS, JavaScript a také back-endové technologie, jako jsou například Ruby, PHP, Python atd. Webová aplikace funguje prostřednictvím webového prohlížeče. Webové aplikace mohou využívat funkce zařízení, na kterých fungují, přičemž ale záleží na webových prohlížečích. To znamená, že tyto funkce zařízení mohou fungovat v prohlížeči Chrome, ale ne v prohlížeči Mozilla Firefox nebo jiném prohlížeči atd.[27][30]

Základní typy webových aplikací

- Static web apps
- Dynamic web apps
- Single-page apps
- Multi-page apps
- Ecommerce web apps

4.1.1 Porovnání základních aspektů

Instalace

Jedním z velkých rozdílů mezi PWA a webovou aplikací je instalace. PWA lze nainstalovat do zařízení, ať už se jedná o počítač nebo mobilní telefon. Webová aplikace je však navržena tak, aby běžela uvnitř webového prohlížeče, a nelze ji tedy do zařízení nainstalovat. Proto je PWA pro uživatele s ohledem na tento aspekt výhodnější.[27][30]

Snadný přístup

Webovou aplikaci nelze nainstalovat, proto se k ní přistupuje pouze prostřednictvím webového prohlížeče, naopak PWA jde instalovat. Po instalaci mohou uživatelé k PWA snadno přistupovat z domovské obrazovky, kde se zobrazí ikona aplikace stejně jako u nativních aplikací. PWA lze také sdílet prostřednictvím odkazu a umožnit tak ostatním, aby si aplikaci nainstalovali. Zmenšuje se tím tak počet kroků instalace.[27][28][30]

Rychlejší UX

Progresivní webové aplikace mohou ukládat do mezipaměti data, která uživatelům poskytují obrázky, texty a další podobný obsah před úplným načtením celé aplikace. Zkracuje se tím tak doba čekání uživatelů, což zvyšuje míru udržení pozornosti uživatelů a zapojení uživatelů. Nabízejí tedy rychlejší UX než webové aplikace.[27][28][30]

Větší zapojení uživatelů

Jednou z hlavních výhod PWA je, že mohou využívat push notifikace a mnoho dalších funkcí zařízení, které umožňují zvýšit zapojení uživatelů. Pomocí push notifkací může aplikace například posílat nabídky na produkty firmy.[27][30]

Offline fungování

Díky mezipaměti může PWA ukládat data v zařízení uživatelů, což uživatelům umožňuje přístup k obsahu i bez připojení k internetu. Stručně řečeno se dá říci, že PWA mohou fungovat offline. Tato offline funkčnost u běžných webových aplikací není k dispozici. [27][30]

4.2 PWA vs. nativní aplikace

Co je to nativní aplikace?

Termín nativní aplikace hovoří sám za sebe, znamená to že je nativně vyvinutá pro konkrétní platformu, iOS nebo Android, a zajišťuje vynikající přizpůsobitelnost a skvělé UX. Ať už si vyberete vývoj aplikace pro Android nebo iOS, vaše aplikace bude mít neopakovatelnou kódovou základnu.[31]

Nativní mobilní aplikace používáte již od počátku éry chytrých telefonů. Nativní mobilní aplikace se instaluje z obchodů s aplikacemi v systémech iOS nebo Android. Jsou vytvořeny tak, aby využívaly hardwarové možnosti konkrétního mobilního zařízení a poskytovaly poutavý uživatelský zážitek.[32]

Protože nativní aplikace jsou vyvíjeny pro konkrétní operační systém, slibují vyšší výkon a efektivnější využití možností hardwaru. Nativní aplikace pro platformy iOS a Android se vyvíjejí samostatně, což znamená větší investice z hlediska peněz, času a úsilí.[32]

4.2.1 Porovnání základních aspektů

Instalace

Klíčovým rozdílem mezi PWA a nativními aplikacemi je způsob, jakým k nim koncový uživatel přistupuje. Nativní aplikace se vyhledávají a instalují prostřednictvím obchodu s aplikacemi, jako je Google Play nebo Apple App Store pro iOS. PWA se instalují prostřednictvím prohlížeče, kdy po navštívení stránky pomocí URL máme možnost přidat si PWA na domovskou obrazovku.[33]

Jedním z hlavních problémů, na který mnoho lidí při instalaci aplikace myslí, je, kolik paměti zabírá. Progresivní webové aplikace nevyžadují žádnou instalaci. Při pohledu do prohlížeče si návštěvníci mohou aplikaci snadno přidat do záložek a následně na domovskou obrazovku. PWA se zobrazí na domovské obrazovce, v jejich adresáři aplikací a bude také odesílat oznámení. Kromě toho progresivní aplikace nezabírají ve srovnání s plnými aplikacemi tolik místa.[34]

Cross-Platform

Hlavní rozdíl mezi progresivní webovou aplikací a nativní aplikací spočívá v tom, že se PWA přizpůsobí různým operačním systémům a velikostem obrazovky. Na rozdíl od nativních aplikací můžete PWA otevřít v systému iOS, Android, Windows nebo v jakémkoli operačním systému - nabídnou skvělý uživatelský zážitek bez ohledu na zařízení.[31]

Offline fungování

PWA dokáží do určité míry fungovat Offline. Zajištěno je to tak, že určité funkce aplikace se uloží do mezipaměti. Problémem však nastává, pakliže by uživatel chtěl využívat veškeré funkce PWA offline, jelikož to není možné. PWA bude mít vžy určitou část funkcionality nedostupnou bez připojení k internetu. Na druhé straně nativní aplikace v offline fungování vynikají, protože jsou celé uloženy na uživatelském zařízení. To znamená, že vešekerá funkcionalita je dostupná i bez internetu.[32][33]

Výkon

Progresivní webové aplikace se načítají rychleji, ale běží v prohlížeči třetí strany. To znamená, že vždy bude existovat pravděpodobnost latence.[32] Naproti tomu nativní aplikace se po instalaci bezproblémově integruje jako součást smartphonu. Dokáže tím pádem i lépe využívat vlastnosti hardwaru a zařízení než PWA. Nativní mobilní aplikace jsou také výkonnější a dosahují vysokého skóre v oblasti výkonu díky kódu který je optimalizovaný a zaměřený na konkrétní platformu.[32]

Protože nativní aplikace musí být vyvinuty pro konkrétní zařízení, na kterém je používáte, mohou snadno využívat nástroje specifické pro danou platformu a plně využívat všechny funkce, které poskytuje daný operační systém. Když máte PWA, nemůžete tyto funkce plně využít, to znamená že nativní aplikace mají právě z tohoto důvodu tendenci fungovat celkově lépe. Nativní apli-

kace jsou sice dražší, ale díky jejich výhodám se vyplatí, pokud se chcete plně věnovat tomu, aby výkon vaší aplikace byl na každém zařízení co nejlepší.[34]

Updaty

Při používání PWA uživatel ani nepostřehne, že je aplikace updatována. Je to kvůli tomu, že aplikace si při spuštění automaticky stáhne nejnovější data sama. U nativních aplikací má update na starost povětšinou obchod play nebo app store, ale může se stát že bude aplikace vyžadovat manuální update(dnes velmi zřídka). Dá se tedy říci, že v tomto aspektu jsou na tom PWA a nativní aplikace dosti podobně.[32][33]

Zabezpečení

Nativní aplikace mohou být bezpečným řešením jak pro vlastníka aplikace, tak pro uživatele. V nativní aplikaci je snazší použít vícefaktorové ověřování než v PWA, což je užitečné, pokud má aplikace funkci přihlášení. Vícefaktorové ověřování přidává nativním aplikacím velkou vrstvu zabezpečení. Obecně se dá tedy říct, že nativní aplikace můžou být lépe zabezpečeny než PWA, ale není to nic tak markantního.[33]

5 Praktická část

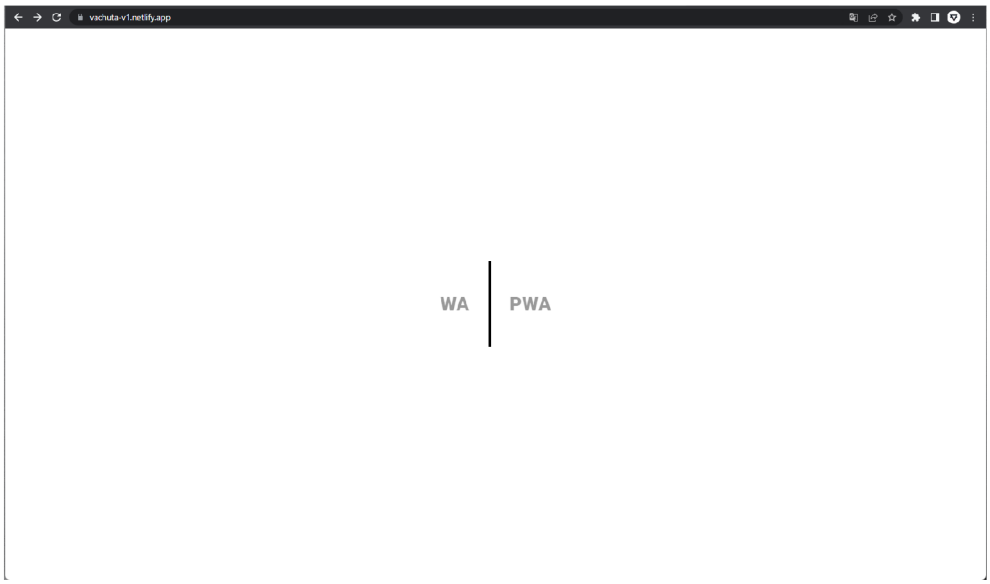
V praktické části bakalářské práce jsem vytvořil WA(webovou aplikaci) a PWA (Progreseivní webovou aplikaci), abych mohl demonstrovat jednotlivé rozdíly těchto technologií. Pro porovnání jsem vytvořil aplikaci, která slouží k počítání obsahů a obvodů 2d útvarů, počítání povrchů a objemů 3d těles a k převodu jednotek délky společně s převodem jednotek obsahu. Pro naviagci mezi těmito dvěmi aplikacemi jsem vytvořil i velmi jednoduché menu.

5.1 Rozdělení a představení komponent

Projekt jsem rozdělil na 3 hlavní komponenty, které tedy dohromady tvoří jeden funkční celek.

Menu

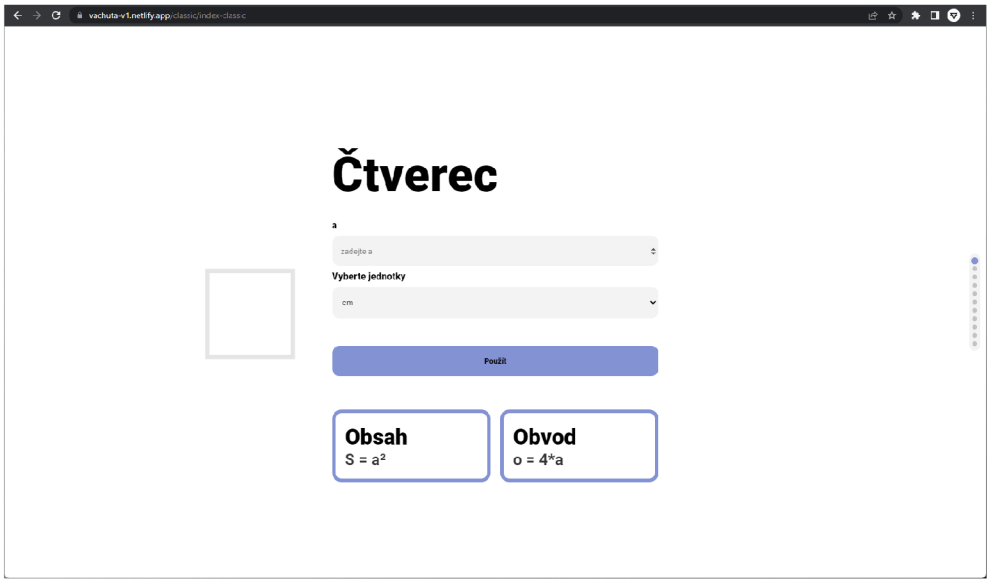
Menu jsem vytvořil kvůli možnosti jednoduché navigace mezi dvěmi aplikacemi.



Obrázek 1: Menu

WA

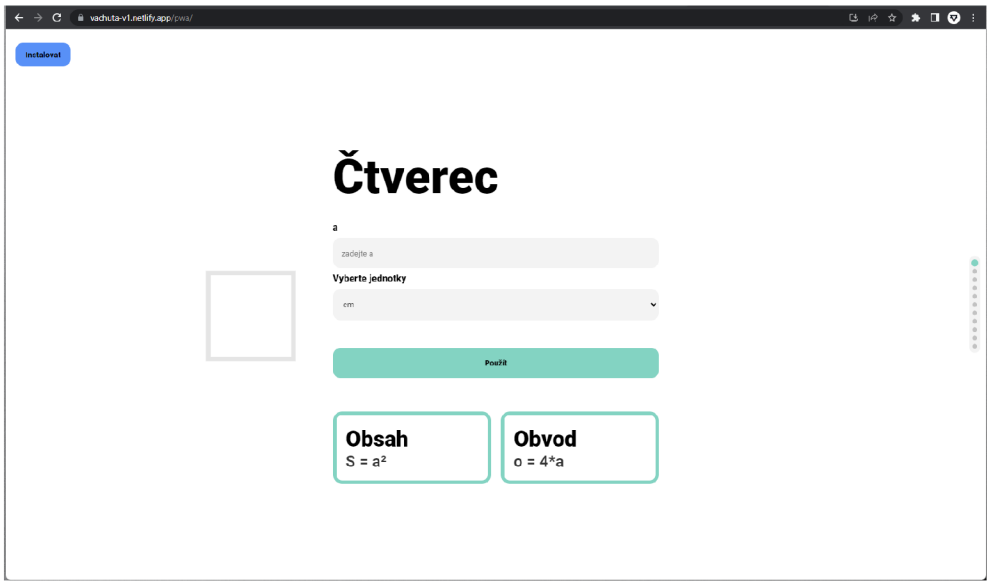
Webovou aplikaci jsem vytvořil za účelem porovnání již zmíněných technologii (WA vs. PWA). WA jsem odlišil od PWA barvou, aby od sebe aplikace byly dobře rozeznatelné.



Obrázek 2: Webová aplikace(klasická)

PWA

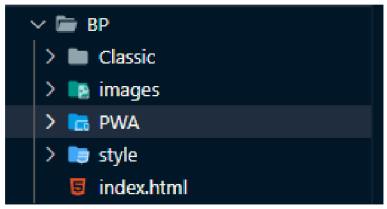
PWA jsem vytvořil, jako stěžejní prvek celého projektu. UI jsem zvolil stejné, avšak použil jsem opět jiné sekundární barvy pro lepší rozlišení samotných aplikací.



Obrázek 3: Progresivní Webová aplikace

5.2 Rozdělení adresáře projektu

V adresáři BP jsem vytvořil pro efektivnější funkčnost jednotlivé složky. Složku pro WA, složku pro PWA, složku s obrázky a jednotnou složku pro styly.



Obrázek 4: Adresář projektu

5.3 Tvorba jednotlivých komponent

5.3.1 Menu

HTML

Obrázek 5: HTML - Menu

V hlavičce (head) menu jsem použil základní meta tagy, které zajišťují základní responzivitu a správné znakové kódování. Následně jsem přidal link na styly pro tuto stránku. Další viditelné linky v hlavičce, jsou linky od google fonts. Tyto linky jsem přidal, kvůli importování fontu, který používám v každé aplikaci projektu. Tyto linky zjednodušeně importují vše potřebné pro to, aby font na mojí stránce fungoval. Do hlavičky jsem přidal taktéž název samotné stránky.

V těle (body) jsem si vytvořil div s třídou cont (zkráceně container), do kterého jsem vložil dva hyperlinky a div s třidou line. Přidané hyperlinky slouží k navigaci na WA a PWA, proto jejich href zahrnuje cesty k html souborům těchto dvou aplikací. Div (.line) slouží jako grafický prvek - čára.

```html
<body>
  <div class="cont">
    <a href="Classic/index-classic.html">WA</a>
    <div class="line"></div>
    <a href="./PWA/index.html">PWA</a>
  </div>
</body>
```

Příklad 7: menu HTML - body

CSS

BP > style > style-main.css > ...

```css
* {
  margin: 0;
  padding: 0;
}

body {
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  font-family: "Roboto", sans-serif;
}

.cont {
  min-height: 20vh;
  min-width: 40%;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: center;
  gap: 2em;
}

.line {
  width: 4px;
  height: 9em;
  background-color: black;
}
.cont a {
  color: black;
  text-decoration: none;
  opacity: 0.4;
  font-size: 2em;
  font-weight: 900;
  transition: 0.4s ease-in-out;
}
.cont a:hover {
  opacity: 1;
  transform: scale(1.02);
}
```

Obrázek 6: CSS - Menu

```
* {
    margin: 0;
    padding: 0;
}
```

Příklad 8: menu css-reset

Pomocí selectoru * jsem vybral všechny prvky html a pomocí nastavení hodnot margin a padding na 0 jsem docílil toho, aby všechny nadpisy, divy atd. neměly žádný margin ani padding.

```
body {
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  font-family: "Roboto", sans-serif;
}
```

Příklad 9: menu css-body

Následně byla potřeba nastavit výška těla (body), zarovnání elementů v těle a nastavení fontu pro stránku. Proto jsem tělu nastavil výšku 100vh (viewport height), která zajišťuje, aby na každém zařízení tělo (body) dosáhlo 100% výšky daného zařízení. Pro zarovnání elementů uvnitř těla (body) jsem použil display flex, to znamená že tělo využívá flex box rozložení. Jelikož jsem se rozhodl použít flex box, mohu zarovnat elementy na střed jak vertikálně tak horizontálně a to velmi snadno. K vertikálnímu zarovnání se v tomto případě používá align-items a k horizontálnímu zarovnání justify-content. U obou zarovnání jsem použil zarovnání na střed, tedy center. Poslední věc, kterou jsem provedl, bylo naastavení fontu Roboto pro všechny prvnky v těle (body). Nastavení fontu se provádí pomocí font-family.

```
.cont {
  min-height: 20vh;
  min-width: 40%;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: center;
  gap: 2em;
}
```

Příklad 10: menu css-cont

Pro div s třídou cont zde nastavuji minimální výšku. Tedy jakmile je výška divu rovna nastavené hodnotě, nemůže být menší, ale může být libovolně větší. Jako další věc jsem nastavil minimální šířku divu tzn., že minimální šířka je 40%. Zarovnání elementů uvnitř divu bylo opět dosaženo za pomoci flex boxu, kdy jsem vybral středové zarovnání. Poslední co jsem nastavil, byl mezera (gap) mezi elementy, gap lze použít díky flex boxu.

```
.line {
  width: 4px;
  height: 9em;
  background-color: black;
}
```

Příklad 11: menu css-line

V této části CSS nastavuji výšku, šířku a barvu čáry. Pro nastavení šířky a výšky používám klasické width a height. Jako barva pozadí je zde zvolena černá, čehož jsem dosáhl pomocí background-color. Ke zvolení takovéto výšky a šířky jsem se rozhodl po vyzkoušení několika variant.

```
.cont a {
  color: black;
  text-decoration: none;
  opacity: 0.4;
  font-size: 2em;
  font-weight: 900;
  transition: 0.4s ease-in-out;
}
```

Příklad 12: menu css-cont a

Hyperlinky, které jsou součástí divu (.cont), musí být taktéž patřičně nastylovány. Proto jsem pro větší estetičnost zvolil černou barvu, následně jsem odstranil dekoraci hyperlinku pomocí text-decoration. Tím jsem tedy odstranil podtržení textu. Pro tyto hyperlinky je zde použita i zvýšená průhlednost (opacity), pro kterou jsem se rozhodl kvůli dalším krokům v další část stylování. Taktéž jsem zvolil určitou velikost a váhu fontu. A jako poslední jsem připravil přechod pro animaci.

```
.cont a:hover {
  opacity: 1;
  transform: scale(1.02);
}
```

Příklad 13: menu css-cont a

Jako poslední v řadě, jsem nastavil animaci pro hyperlinky. Animace se spustí při najetí myší na hyperlinky toto obstarává :hover selector. Jakmile myš najede na link, průhlednost se změní z 0.4 na 1 a text se zvětší o 0.02. Tím jsem vytvořil jednoduchou malou animaci.

5.3.2 PWA

HTML

```html
<head>
  <!-- Responsive -->
  <meta charset="utf-8" />
  <meta
    name="viewport"
    content="width=device-width,
      initial-scale=1"
  />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />

  <!-- Title -->
  <title>PWA</title>

  <!-- Meta Tags required for Progressive Web App -->
  <meta name="apple-mobile-web-app-status-bar" content="#aa7700" />
  <meta name="theme-color" content="black" />

  <!-- Manifest File Link -->
  <link rel="manifest" href="manifest.json" />
  <link rel="stylesheet" href="../style/style.css" />
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
  <link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/swiper@8/swiper-bundle.min.css"
  />

  <link
    href="https://fonts.googleapis.com/css2?family=Bebas+Neue&family=Oswald:wght@300;400;500;700&display=s
    rel="stylesheet"
  />

  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
  <link
    href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700;900&display=swap"
    rel="stylesheet"
  />
</head>
```

Obrázek 7: HTML - hlavička (head)

```
<!-- Responsive -->
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,initial-scale
   =1"/>
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<!-- Title -->
<title>PWA</title>
<!-- Meta Tags required for Progressive Web App -->
<meta name="apple-mobile-web-app-status-bar" content="#aa7700"
   />
<meta name="theme-color" content="black" />
```

Příklad 14: PWA HTML - head

V této části hlavičky (head) jsem nejprve přidal úplně nejzákladnější meta tagy. Tedy tag pro správné formátování a kódování písma, tagy pro základní responzivitu obsahu. Následně jsem přidal i title se jménem stránky. Jako poslední meta tagy jsem musel přidat požadované tagy pro PWA. Tag, který přidává na apple zařízení status bar s určitou barvou a meta tag pro určení theme barvy.

```
<!-- Manifest File link -->
<link rel="manifest" href="manifest.json" />
<link rel="stylesheet" href="../style/style.css" />
<link rel="preconnect" href="https://fonts.googleapis.com" />
<link rel="preconnect" href="https://fonts.gstatic.com"
   crossorigin />
<link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/swiper@8/swiper-bundle.
   min.css"
/>
<link
    href="https://fonts.googleapis.com/css2?family=Bebas+Neue&
   family=Oswald:wght@300;400;500;700&display=swap"
```

```
12      rel="stylesheet"
13  />
14  <link rel="preconnect" href="https://fonts.googleapis.com" />
15  <link rel="preconnect" href="https://fonts.gstatic.com"
        crossorigin />
16  <link
17      href="https://fonts.googleapis.com/css2?family=
        Roboto:wght@400;700;900&display=swap"
18      rel="stylesheet"
19  />
```

Příklad 15: PWA HTML - head

V druhé části hlavičky (head) jsem přidal několik nutných linků, které jsou potřebné pro správný chod a vzhled aplikace. První link odkazuje na manifest.json, ve kterém se nachází základní nutné informace pro chod PWA. Druhý link, který jsem přidal, je link propojující HTML a CSS, tedy styly. Jako poslední velmi důležitý link považuji ten, který obsahuje CDN pro swiper.js knihovnu. Bez této knihovny by aplikace nemohla fungovat.

```
<div class="ios">···
</div>

<div class="android"></div>
<section class="installSection">···
</section>

<div class="swiper">
  <!-- Additional required wrapper -->
  <div class="swiper-wrapper">
    <!-- Slides -->
    <!--Square-------------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Rectangle----------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Triangle-----------------------------------------------------------
    <div class="swiper-slide">···
    </div>
    <!--Circle-------------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Cube---------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Block--------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Sphere-------------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Cone---------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Cylinder------------------------------------------------------------>
    <div class="swiper-slide">···
    </div>
    <!--Converter of Lenghts----------------------------------------------------------->
    <div class="swiper-slide">···
    </div>
    <!--Converter of volume------------------------------------------------------------>
    <div class="swiper-slide">···
    </div>
  </div>
</div>
```

Obrázek 8: HTML - základní struktura obsahu

Obrázek ukazuje základní strukturu těla (body), kdy hlavní kontejner je div s třídou "swiper". Tento kontejner zajišťuje funkčnost právě swiper.js knihovny, kterou jsem pro tuto aplikaci použil. V divu "swiper"se nachází další div, který funguje jako obal pro samotné slidy, které tvoří aplikaci. Slidy jsem tvořil na základě dokumentace swiper.js, kdy pro vytvoření slidu je nutno vytvořit div s třídou "swiper-slide". Vytvořil jsem tedy 11 slidů. Každý slide

představuje jednu miniaplikaci na výpočet ať už obsahu a obvodu, povrchu a objemu nebo převodu jednotek. Pro větši přehlednost, jsem si jednotlivé slidy patřičně okomentoval.

```
<div class="swiper-pagination"></div>
<script src="detection.js"></script>
<script src="main.js" defer></script>
<script src="https://cdn.jsdelivr.net/npm/swiper@8/swiper-bundle.min.js"></script>
<script>
  const swiper = new Swiper(".swiper", {
    // Optional parameters
    direction: "vertical",
    mousewheel: true,

    // If we need pagination
    pagination: {
      el: ".swiper-pagination",
      clickable: true,
    },
    effect: "coverflow",
    coverflowEffect: {
      rotate: 80,
      slideShadows: false,
    },
    // Navigation arrows
    navigation: {
      nextEl: ".swiper-button-next",
      prevEl: ".swiper-button-prev",
    },

    // And if we need scrollbar
  });

  window.addEventListener("load", () => {
    registerSW();
  });

  // Register the Service Worker
  async function registerSW() {
    if ("serviceWorker" in navigator) {
      try {
        await navigator.serviceWorker.register("serviceworker.js");
      } catch (e) {
        console.log("SW registration failed");
      }
    }
  }
</script>
```

Obrázek 9: HTML - základní struktura obsahu

Na tomto obrázku je vidět poslední část HTML. V této části jsem hlavně propojoval HTML s JS a obstarával základní logiku aplikace.

```
<div class="ios">
      <svg class="close-btn">
       ....
      </svg>
      <ol>
        <h4>Nainstalování aplikace</h4>
        <li>
          <span>
            Klikněte na
            <svg class="share-ios">
            ....
            </svg>
          </span>
        </li>
        <li>
          <span>
            Přidat na plochu
            <svg class="addTo">
            ....
            </svg>
          </span>
        </li>
        <li>Přidat</li>
      </ol>
</div>
```

Příklad 16: PWA HTML - ios

V této části HTML jsem vytvořil div s třídou "ios". Tento div slouží jako návod pro instalaci PWA na ios. Uvnitř jsem přidal svg ikony. Konkrétně jsem přidal tyto ikony: Ikona zavírání, ikona sdílet (share), ikona přidat (add).Ikonám jsem taktéž přidělil třídy, aby jsem je mohl později v css nastylovat.

Základní strukturu divu tvoří ol, tedy číslovaný seznam (ordered list). Tento seznam jsem využil, protože jeho vlastnosti ideálně sedí na číslovaný návod instalace. V seznamu jsem přidal itemy pomocí li (list item) tagů. Jelikož jsem

přidal do návodu i ikony, musel jsem přidat do li také span, kvůli pozdější možnosti středového zarovnání textu s ikonou.

```
<section class="installSection">
    <button class="install">Instalovat</button>
</section>
```

Příklad 17: PWA HTML - instalovat

Zde jsem vytvořil sekci s třídou "installSection". Tato sekce slouží jako kontejner pro tlačítko (button), které má po kliknutí nabídnout možnost instalace PWA na počítač. Samotnému tlačítku jsem taktéž přidělil třídu a to "install". Pomocí této třídy jsem později schopný propojit funkčně tlačítko s JS.

```
<div class="swiper">
    <div class="swiper-wrapper">
        .......
    </div>
</div>
<div class="swiper-pagination"></div>
```

Příklad 18: PWA HTML - swiper

Protože jsem se pro svou aplikaci rozhodl použít swiper.js knihovnu, která mi umožňuje pojmout aplikaci jako swipeable, musel jsem tedy přidat základní strukturu.

Základem je div se specifickou třídou "swiper"v němž je kontejner starající se o slidy. Tomuto kontejneru jsem opět musel přidělit specificky danou třídu a to "swiper-wrapper". Tímto jsem tedy připrvil HTML pro práci se samotnou swiper.js knihovnou. Jako poslední jsem ze swipper.js použil možnost stránkování (pagination), proto jsem musel přidat div s třídou "swiper-pagination". Tento div nám z knihovny naimportuje navigační tečky (bullety).

```
<div class="swiper-slide">
    ...
</div>
```

Příklad 19: PWA HTML - swiper-slide

V samotném kontejneru "swiper-wrapper"jsem dle dokumentace musel přidat jednotlivé slidy. Toho jsem docílil pomocí přidávání divů s konkrétní třídou "swiper-slide". Tento div je následně základem pro všechny moje micro aplikace, jelikož jsou všechny brány jako jednotlivé slidy.

```
<div class="swiper">
    <div class="container">
        <div class="subcontainer">
            <h1>nazev-objektu</h1>
            <form action="" class="nazev-objektu">
                ...
            </form>
            <div class="exportContainer">
                ...
            </div>
        </div>
        <img src=".." alt="" />
    </div>
</div>
```

Příklad 20: PWA HTML - slide

Pro každý slide jsem vytvořil základní HTML strukturu, do které jsem podle potřeby pro daný slide přidával konkrétní elementy.

První jsem vytvořil div s třídou "container", který funguje jako kontejner. Tento div má význam především pro layout samotného slidu. Pro ten stejný účel jsem vytvořil i div s třídou "subcontainer". Oba dva divy spolu budou později pracovat, prostřednicvtím CSS, jako elementy flex-box layoutu. Div s třídou "subcontainer"obsahuje několik elementů. Jako první jsem přidal nadpis

h1, který se vyskytuje v každém slidu PWA.

Velmi důležitou roli však hraje element form. Form jsem přidal, protože díky němu budu moci v pozdější fázi pracovat se zadanými hodnotami. Pro samotné získaní hodnot ve formu jsem použil inputy, které jsou k tomuto účelu určené.

Po formu jsem následně přidal div s třídou"exportContainer", který obsahuje textové placeholdery h2 a h3, které budu později využívané JavaScriptem pro export výsledků.

Posledním elementem slidu je img, který jsem přidal pro vylepšení estetiky aplikace pomocí obrázků.

```
<div class="swiper-slide">
    <div class="container">
        <div class="subcontainer">
            <h1>čtverec</h1>
            <form action="" class="square">
            <label for="">a</label>
                <input
                    min="1"
                    name="a"
                    type="number"
                    step="0.01"
                    placeholder="zadejte a"
                />
                <label for="">Vyberte jednotky</label>
                <select name="unitSquare" id="">
                    <option value="cm">cm</option>
                    <option value="m">m</option>
                </select>
                <button>Použít</button>
            </form>
            <div class="exportContainer">
                <div class="s">
                    <h2>Obsah</h2>
                    <h3 class="squareArea">S = a&sup2</h3>
                </div>
                <div class="o">
                    <h2>Obvod</h2>
                    <h3 class="squareCircum">o = 4*a</h3>
                </div>
            </div>
            <img src="../images/square.svg" alt="" />
        </div>
    </div>
</div>
```

Příklad 21: PWA HTML - slide (čtverec)

Tento HTML kod jsem vytvořil pro slide, který bude sloužit k výpočtu obsahu a obvodu čtverce.

Protože s jednotlivými formy budu pracovat v JS musel jsem každému přidělit třídu. Pro tento form, jelikož se týka čtverce, jsem zvolil třídu "square"(anglicky čtverec). Jelikož potřebuji pracovat s daty co zadává uživatel, využil jsem nejjednodušší cestu a to inputy. Inputy jsem přidal do každého formu a do všech slidů. Každý input jsem také opatřil jménem pomocí label, aby uživatel viděl dobře co a kam zadat. V tomto konkrétním případě jsem použil pouze jeden input. Následně jsem přidal výběr jednotek, kdyby chtěl uživatel vidět výsledek s metry nebo centimetry. Výběr jednotek jsem vytvořil pomocí selectu, kterému jsem dal jméno "unitSquare", abych v JS měl přístup k vybrané jednotce. Do formu jsem taktéž přidal tlačítko, které bude po kliknutí odesílat data do JS a startovat výpočty v JS.

Po formu jsem do divu "subcontainer"implementoval další div s třídou "exportContainer". Tento div jsem použil jako prostředek ke kontrole umístění exportovaných výsledků z JS. Div obsahuje další dva divy. Div s třídou "s"a div s třídou "o". Tyto divy jsem opět použil jako prostředek pro layout exportovaných výsledků. Každý z dvou divů "s"a "o"obsahuje h2 nadpisy. První div obsahuje nadpis obsah a text s výsledkem a druhý div obsahuje nadpis obvod a text s výsledkem. Výsledky jsou zobrazeny pomocí h3 nadpisu a mají přiděleny třídy, aby jsem později v JS mohl s těmito DOM elementy správně a dobře pracovat.

Poslední věcí co jsem přidal, je img. Element img jsem použil u všech slidů pro zobrazení konkrétního obrazce nebo tělesa, které se na samotném slidu nachází. V tomto konkrétním případě se jedná o obrázek čtverce. Jelikož v aplikaci se obrázky vyskytují pouze jako obsah slidu, nemusel jsem pro pozdější stylování přidělovat každému obrázku třídu.

Tímto způsobem jsem vytvořil všechny ostatní slidy, které slouží k výpočtům pro 2D obrazce a 3D tělesa. Rozdíly jsou pouze v počtech inputů a minoritních změnách layoutu.

```
<input
    min="1"
    name="a"
    type="number"
    step="0.01"
    placeholder="zadejte a"
/>
```

Příklad 22: PWA HTML - input

Každý input v aplikaci jsem nakonfiguroval tak, aby vyhovoval mým potřebám. Inputy jsou nakonfigurovány tak, aby nejmenší možné číslo bylo 1. Taktéž jsem je nastavil jako inputy typu číslo (number). Velikost kroku čísla jsem nastavil na 0.01, kdyby někdo chtěl používat šipky v inputech. V neposlední řadě mají inputy placeholdery a své jméno.

```
<script src="detection.js"></script>
<script src="main.js" defer></script>
<script src="https://cdn.jsdelivr.net/npm/swiper@8/
   swiper-bundle.min.js"></script>
```

Příklad 23: PWA HTML - scripts

Pomocí těchto scriptů, jsem propojil index.html s main.js a detection.js. Poslední script importuje swiper.js knihovnu pomocí cdn.

```
<script>
      const swiper = new Swiper(".swiper", {
        // Optional parameters
        direction: "vertical",
        mousewheel: true,

        // If we need pagination
        pagination: {
          el: ".swiper-pagination",
          clickable: true,
        },
        effect: "coverflow",
        coverflowEffect: {
          rotate: 80,
          slideShadows: false,
        },
      });
</script>
```

Příklad 24: PWA HTML - scripts

V této části scriptu (který je součásti HTML) jsem nastavil swiper.js pro moje potřeby. V první řadě jsem vytvořil objekt swiper, který se odkazuje na třídu "swiper". Tento objekt má několik hodnot které jsem mu nastavil. Nejprve jsem nastavil direction na "vertical"to znamená, že moje aplikace bude swipovatelná vertikálně, tedy odshora dolů. Následně jsem nastavil hodnotu mousewheel na true, čímž jsem povolil na počítači swipovat pomocí scrollování na myši.

Objekt pagination zajišťuje logiku stránkování. V mém případě jsem objektu u proměnné clickable nastavil hodnotu na true, to znamená že navigační kuličky (bullety) na boku aplikace budou fungovat na kliknutí. Druhá proměnná objektu pagination je el. Tato hodnota odkazuje na třídu "swiper-pagination".

Dalším objektem který jsem zde na základě dokumentace vytvořil je cover-

flowEffect. Tento objekt slouží jako animace při swipování mezi slidy. Obsahuje dvě proměnné. Proměnnou rotate jsem nastavil na 80, což mi zajistí natočení slidu na 80 stupňů při swipování. Druhou proměnnou slideShadows jsem nastavil na "false", protože nechci aby jednotlivé slidy při swipování měli stín.

```
<script>
      window.addEventListener("load", () => {
        registerSW();
      });

      // Register the Service Worker
      async function registerSW() {
        if ("serviceWorker" in navigator) {
          try {
            await navigator.serviceWorker.register("
   serviceworker.js");
          } catch (e) {
            console.log("SW registration failed");
          }
        }
      }
</script>
```

Příklad 25: PWA HTML - registrace SW

V poslední části scriptu uvnitř HTML jsem musel přidat script, který slouží k registraci serviceworkeru. Tento script je pro chod PWA klíčový.

Asynchronní funknce s názvem "registerSW"má za úkol zjistit zda-li je serviceworker v navigátoru. Pokud je serviceworker v navigátoru, funkce zkusí registrovat serviceworker.js, jestliže pokus o registraci selže do konzole se vypíše "SW registration failed". Celou funkci spouštím pomocí "window.addEventListener"to znamená, že jakmile se načte okno s aplikací, listener zavolá funkci a funkce se automaticky spustí a pokusí se registrovat serviceworker.

JavaScript

serviceworker.js

```
let staticCacheName = "pwa";

self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll([
        "/",
        "index.html",
        "main.js",
        "detection.js",
        "../style/style.css",
      ]);
    })
  );
});

self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

  event.respondWith(
    caches.match(event.request).then(function (response) {
      return response || fetch(event.request);
    })
  );
});
```

Příklad 26: PWA JS - serviceworker

Zde jsem vytvořil stěžejní bod mé aplikace a to serviceworker, bez kterého by PWA nemohla fungovat.

Na základě podkladů jsem vytvořil proměnnou s názvem staticChache-Name a nastavil její hodnotu na "pwa". V další části scriptu je vidět, že SW naslouchá sám sobě a jakmile se prohlížeč snaží nainstalovat novou verzi PWA,

otevře cache, zjistí jestli již nějaká cache existuje a uloží do ní nejnovější data (index.html...) a to pomocí "return cache.addAll()".

Druhou část scriptu tvoří opět addEventListener, který jsem opět přidal na základě dokumentací a podkladů, tentokrát však obstarává kontrolu dat. Pokud se data nezměnili nebo nechybí, nebude potřeba stahovat žádný nový obsah ze serveru, pokud však funkce nenajde požadovaná data, stáhne si je ze serveru aby vše bylo funkční a nejnovější.

detection.js

```
//checks if app is running in browser tab or standalone app
window.addEventListener("DOMContentLoaded", () => {
  let displayMode = "browser tab";
  if (window.matchMedia("(display-mode: standalone)").matches) {
    displayMode = "standalone";
  }
  // Log launch display mode to analytics
  console.log("DISPLAY_MODE_LAUNCH:", displayMode);
  // if display mode is browser tab it shows install button, if not button is hidden
  if (displayMode == "browser tab") {
    //Check wheter u use IOS or android
    window.addEventListener("load", (e) => {
      var userAgent = navigator.userAgent || navigator.vendor || window.opera;

      // Windows Phone must come first because its UA also contains "Android"
      if (/windows phone/i.test(userAgent)) {
        console.log("Windows Phone");
        document.querySelector(".installSection").style.display = "none";
      }
      if (/android/i.test(userAgent)) {
        console.log("Android");
        document.querySelector(".installSection").style.display = "none";
      }
      if (/iPad|iPhone|iPod/.test(userAgent) && !window.MSStream) {
        console.log("iOS");
        document.querySelector(".installSection").style.display = "none";
        iosDiv.style.display = "block";
      }
      console.log("UNKNOWN");
    });

    // manual install button
    let deferredPrompt;
    window.addEventListener("beforeinstallprompt", (e) => {
      deferredPrompt = e;
    });
    const installApp = document.querySelector(".install");
    installApp.addEventListener("click", async () => {
      if (deferredPrompt !== null) {
        deferredPrompt.prompt();
        const { outcome } = await deferredPrompt.userChoice;
        if (outcome === "accepted") {
          deferredPrompt = null;
        }
      }
    });
    window.addEventListener("appinstalled", async function (e) {
      installApp.style.display = "none";
    });
  } else {
    document.querySelector(".installSection").style.display = "none";
  }
});
let closeBtn = document.querySelector(".close-btn");
let iosDiv = document.querySelector(".ios");
closeBtn.addEventListener("click", () => {
  iosDiv.style.display = "none";
});
```

Obrázek 10: JS - detection.js

Tento kód jsem vytvořil, protože jsem potřeboval poznat různé OS a jestli je PWA otevřená v tabu prohlížeče nebo jestli je otevřena v okně aplikace. Taktéž jsem do scriptu zakomponoval kód, který umožňuje na počítači instalovat jednoduše aplikaci pomocí kliknutí na tlačítko instalovat. Tlačítko se však

zároveň skryje, když je aplikace otevřená na mobilním zařízení nebo v okně aplikace.

```
window.addEventListener("DOMContentLoaded", () => {
    ....
});
```

Příklad 27: PWA JS - detection.js

Metodu "window.addEventListener"jsem přidal, protože jsem potřeboval zajistit to, aby byla hned po načtení všech DOM elementů spuštěna "arrow"funkce. Tato metoda je tedy spouštěčem 90% kódu, který se v detection.js nachází.

```
let displayMode = "browser tab";
if (window.matchMedia("(display-mode: standalone)").matches) {
    displayMode = "standalone";
}
console.log("DISPLAY_MODE_LAUNCH:", displayMode);
if (displayMode == "browser tab") {
....
}else {
    document.querySelector(".installSection").style.display = "
    none";
}
```

Příklad 28: PWA JS - detection.js

Do "window.addEventListener"jsem vložil nejprve proměnnou, kterou jsem nastavil na "browser tab". Následně jsem přidal dvě podmínky. První if podmínka kontroluje, jestli je PWA aplikace otevřena jako samostatná nainstalovaná apliklace. Pokud ano, nastaví proměnnou displayMode na "standalone". Druhá if podmínka kontroluje, jestli je aplikace otevřena v tabu prohlížeče. Jestliže je, s proměnnou se nic neděje, protože je již nastavená na "browser tab"a spustí se kód uvnitř podmínky. Avšak pokud druhá if podmínka neprojde, pomocí else jsem nastavil neviditelnost instalovací sekce, která pro iOS

zobrazuje návod k instalaci. Kromě podmínek jsem přidal i jednoduchý výpis do konzole. Tento log vypíše o jaký displayMode se jedná.

```
window.addEventListener("load", (e) => {
      let userAgent = navigator.userAgent || navigator.vendor
   || window.opera;
      // Windows Phone must come first because its UA also
   contains "Android"
      if (/windows phone/i.test(userAgent)) {
        console.log("Windows Phone");
        document.querySelector(".installSection").style.display
    = "none";
      }
      if (/android/i.test(userAgent)) {
        console.log("Android");
        document.querySelector(".installSection").style.display
    = "none";
      }
      if (/iPad|iPhone|iPod/.test(userAgent) && !window.
   MSStream) {
        console.log("iOS");
        document.querySelector(".installSection").style.display
    = "none";
        iosDiv.style.display = "block";
      }
      console.log("UNKNOWN");
});
```

Příklad 29: PWA JS - detection.js

Do druhé podmínky, kdy "displayMode = browser tab", jsem vložil tento kód. Jedná se opět o window.addEventListener metodu. Metoda spustí kód uvnitř po tom, co se načte celá stránka včetně stylů, scriptů a tak dále. Kód uvnitř arrow funkce slouží k rozpoznání operačního systému zařízení, na kterém je

PWA spuštěna. Rozpoznává tedy tři operační systémy. Telefony s operačním systémem windows, telefony s operačním systémem android a iPhone s iOS. Pro všechny telefony jsem pomocí querySelectoru schoval tlačítko instalace, jelikož instalační tlačítko na telefonech nefunguje. A u iOS jsem zviditelnil předpřipravenou sekci, která obsahuje instalační návod (pouze pro iOS).

```
let deferredPrompt;
window.addEventListener("beforeinstallprompt", (e) => {
     deferredPrompt = e;
});
const installApp = document.querySelector(".install");
installApp.addEventListener("click", async () => {
    if (deferredPrompt !== null) {
        deferredPrompt.prompt();
        const { outcome } = await deferredPrompt.userChoice;
        if (outcome === "accepted") {
          deferredPrompt = null;
        }
      }
});
```

Příklad 30: PWA JS - detection.js

Tento kód je poslední část druhé if podmínky a přidal jsem ho, protože umožňuje instalaci PWA pomocí vlastního tlačítka, které jsem si přidal. První metoda s arrow funkcí, kterou jsem přidal, ukládá event do proměnné deferredPrompt, aby jsem jej mohl použít níže. Dále jsem vytvořil proměnnou installApp, do níž je uložen DOM element. V mém případě je DOM elementem tlačítko, které jsem si přidal v HTML. Pomocí "installApp.addEventListener"jsem zajistil to, aby se po kliknutí na tlačítko ukázalo instalční okénko. Okénko obsahuje tlačítko instalovat a zrušit, pokud se uživatel rozhodne kliknout na instalovat tak kód, který jsem přidal zajistí, aby se okénko při dalším kliknutí neukázalo. Toto okénko by se normálně dalo otevřít pouze po kliknutí na built-in tlačítko, které se nachází ve vyhledávacím poli.

```
let closeBtn = document.querySelector(".close-btn");
let iosDiv = document.querySelector(".ios");
closeBtn.addEventListener("click", () => {
  iosDiv.style.display = "none";
});
```

Příklad 31: PWA JS - detection.js

V poslední části jsem zajistil, aby se instalační návod pro iOS po kliknutí na křížek zavřel. Docílil jsem toho tak, že jsem si vytvořil dvě proměnné, do kterých jsem načetl mé dva DOM elementy a pomocí arrow funkce jsem nastavil, aby po kliknutí na křížek zmizel celý div ".iOS".

main.js

```
// square dom var ----------------------------------------------------------------
let squareArea = document.querySelector(".squareArea");
let squareCircum = document.querySelector(".squareCircum");
let squareForm = document.querySelector(".square");

// rectangle dom var -------------------------------------------------------------
let rectArea = document.querySelector(".rectangleArea");
let rectCircum = document.querySelector(".rectangleCircum");
let rectForm = document.querySelector(".rectangle");

// triangle dom var --------------------------------------------------------------
let triangleArea = document.querySelector(".triagnleArea");
let triangleCircum = document.querySelector(".triagnleCircum");
let triaForm = document.querySelector(".triangle");

// triangle dom var --------------------------------------------------------------
let circleArea = document.querySelector(".circleArea");
let circleCircum = document.querySelector(".circleCircum");
let circleForm = document.querySelector(".circle");

//cube dom var-------------------------------------------------------------------
let cubeArea = document.querySelector(".cubeArea");
let cubeVolume = document.querySelector(".cubeVolume");
let cubeForm = document.querySelector(".cube");

//block dom var-----------------------------------------------------------------
let blockArea = document.querySelector(".blockArea");
let blockVolume = document.querySelector(".blockVolume");
let blockForm = document.querySelector(".block");

//sphere dom var ----------------------------------------------------------------
let sphereArea = document.querySelector(".sphereArea");
let sphereVolume = document.querySelector(".sphereVolume");
let sphereForm = document.querySelector(".sphere");

//cone dom var ------------------------------------------------------------------
let coneArea = document.querySelector(".coneArea");
let coneVolume = document.querySelector(".coneVolume");
let coneForm = document.querySelector(".cone");

//cylinder dom var --------------------------------------------------------------
let cylinderArea = document.querySelector(".cylinderArea");
let cylinderVolume = document.querySelector(".cylinderVolume");
let cylinderForm = document.querySelector(".cylinder");

// converter dom var -------------------------------------------------------------
let converter = document.querySelector(".converter");

//converter area dom var --------------------------------------------------------
let converterArea = document.querySelector(".converterArea");
let convAreaOut = document.querySelector(".convResultArea");
```

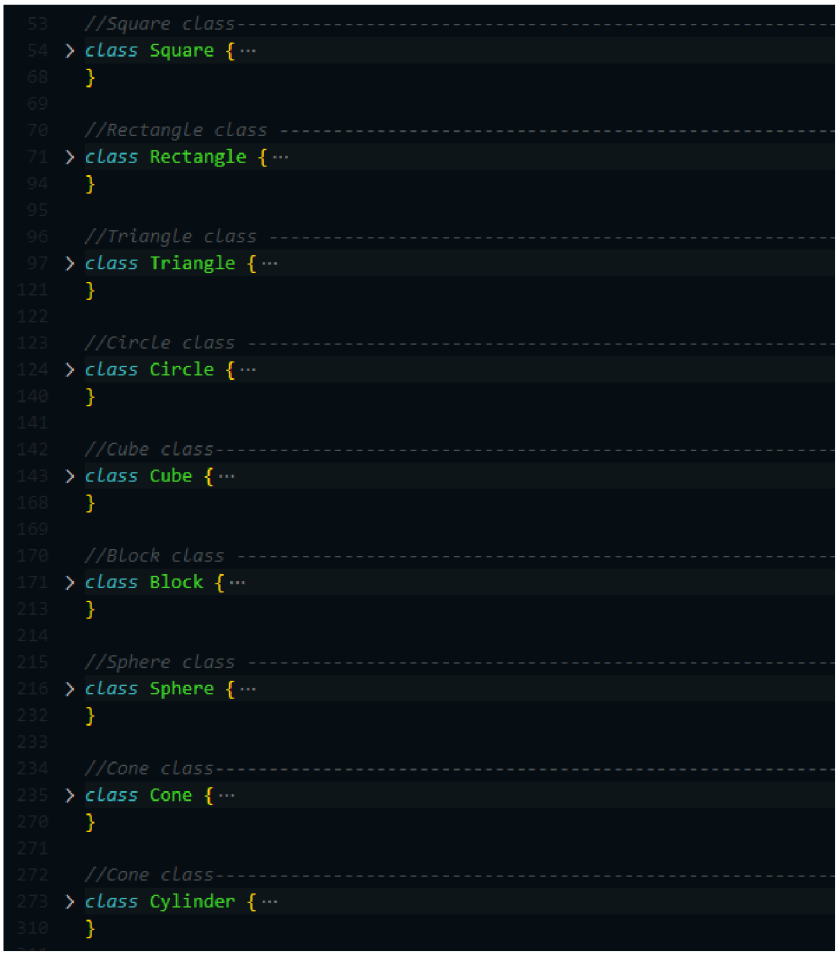
Obrázek 11: JS - main.js (DOM)

V této úplně první části kódu se nachází především proměnné, do nichž jsem si načetl DOM elementy, které budu potřebovat pro vytvoření dobře fungující logiky aplikace.

Vždy jsem si udělal skupiny DOM elementů, které k sobě patří v rámci jednoho formu (formuláře), který mám v HTML. Proměnné co spolu souvisejí,

jsem si pojmenoval tak, že první je vždy klíčové slovo například "square"a k tomu jsem připojil to, co zastupují například Area to znamená že výsledný název bude "squareArea".

Pro získání DOM elementů jsem použil "document.querySelector ("název")"



Obrázek 12: JS - main.js (třídy)

Pro logiku mé aplikace jsem zvolil přístup pomocí tříd. Každá třída obsahuje dvě funkce. U 2D obrazců se jedná o funkce "area"a "circum"v překladu tedy obsah a obvod. U 3D těles se jedná o "area"a "volume"v překladu povrch a objem. Každá funkce má určité parametry se kterými pracuje. Do samotných funkcí jsem přidal patřičné vzorce na výpočet.

Každá třída je pojmenovaná názvem tělesa nebo obrazce pro který je určená. Dá se říci, že jsem zvolil objektový přístup, kdy je jako objekt brána třída, která obsahuje unikátní data.

```
class Square {
  resultArea = 0;
  resultCircum = 0;
  area(a, unit) {
    this.resultArea = a * a;
    squareArea.innerHTML =
      "S = " + a + "&sup2" + " = " + this.resultArea + "" +
   unit + "&sup2";
  }
  circum(a, unit) {
    this.resultCircum = a * 4;
    squareCircum.innerHTML =
      "o = " + "4*" + a + " = " + this.resultCircum + " " +
   unit;
  }
}
```

Příklad 32: PWA JS - main.js - třída

Třídy, které jsem vytvořil, jsou pojmenované podle obrazce/tělesa pro které jsou určeny. Vždy obsahují několik věcí.

Jako první jsem přidal proměnnou result area, kterou jsem nastavil na 0. Tato proměnná slouží k výpočtu pomocí daného vzorce. ve funkci "area"k ní přistupuji pomocí "this", protože pracuji ve třídě. Poté jsem přidal proměnnou "resultCircum". Tato proměnná slouží ke stejným účelům jako proměnná "resultArea". Tedy k výpočtu, tentokrát se však jedná o výpočet obvodu. Opět k ní ve funkci "circum"přistupuji pomocí "this".

Další část kódu, kterou jsem přidal, je určená pro celkovou logiku pro daný obsah, obvod, objem nebo povrch. Jedná se vždy o dvě funkce, přičemž každá funkce má dané parametry, které potřebuje. Parametry slouží v tomto případě

hlavně k výpočtům. V samotné funkci se nachází proměnná, která slouží k výpočtu. Proměnná "unit", která přidá do vzorce jednotky a globální proměnná, jež výsledek výpočtu vezme a uloží ho do DOM elementu, který daný dosazený (ve vzorci) výsledek zobrazí v aplikaci pomocí textu. Takto je tomu v každé funkci.

Tato struktura je užita pro všechny třídy a liší se vždy pouze názvem třídy, názvem globální proměnné, počtem parametrů a užitými vzorci pro výpočet a dosazení. U 3D těles je obvod nahrazený objemem.

```
class Sphere {
  resultArea = 0;
  resultVolume = 0;
  area(r, unit) {
    this.resultArea = 4 * Math.PI * (r * r);
    this.resultArea = Math.round(this.resultArea * 100) / 100;
    sphereArea.innerHTML =
      "S = 4*" + "  " + r + "&sup2 = " + this.resultArea + "" +
    unit + "&sup2";
  }

  volume(r, unit) {
    this.resultVolume = (4 / 3) * Math.PI * (r * r * r);
    this.resultVolume = Math.round(this.resultVolume * 100) /
     100;
    sphereVolume.innerHTML =
      "V= " + "2  " + r + "=" + this.resultVolume + " " + unit;
  }
}
```

Příklad 33: PWA JS - main.js - příklad další třídy

Zde je možné vidět, že třídy se liší hlavně názvem proměnných, počtem parametrů, složitostí výpočtů a tak dále.

```
312   // Square listener function with form data ----------------------------------
313   const sqr = new Square();
314 > squareForm.addEventListener("submit", (e) => {···
323   });
324
325   // Rectangle listener function with form data -------------------------------
326   const rect = new Rectangle();
327 > rectForm.addEventListener("submit", (e) => {···
336   });
337
338   // Triangle listener function with form data --------------------------------
339   const tria = new Triangle();
340 > triaForm.addEventListener("submit", (e) => {···
354   });
355
356   // Circlel istener function with form data ----------------------------------
357   const circle = new Circle();
358 > circleForm.addEventListener("submit", (e) => {···
366   });
367
368   // Cube listener function with form data ------------------------------------
369   const cube = new Cube();
370 > cubeForm.addEventListener("submit", (e) => {···
377   });
378
379   // Block listener function with form data -----------------------------------
380   const block = new Block();
381 > blockForm.addEventListener("submit", (e) => {···
390   });
391
392   // Sphere listener function with form data ----------------------------------
393   const sphere = new Sphere();
394 > sphereForm.addEventListener("submit", (e) => {···
401   });
402
403   // cone listener function with form data ------------------------------------
404   const cone = new Cone();
405 > coneForm.addEventListener("submit", (e) => {···
414   });
415
416   // cylinder listener function with form data --------------------------------
417   const cylinder = new Cylinder();
418 > cylinderForm.addEventListener("submit", (e) => {···
426   });
427
428   // converter listener function
429 > converter.addEventListener("submit", (e) => {···
570   });
571
572 > function convertArea(firstInpt, unitOne, unitTwo) {···
738   }
739
740 > converterArea.addEventListener("submit", (e) => {···
748   });
```

Obrázek 13: JS - main.js

Část kódu, která je zobrazena, slouží k vytváření instancí tříd a následnému volání funkcí, které jsou ve třídách. Přidal jsem taktéž EventListenery s eventem "submit", což mi umožnilo z formů (formulářů) efektivně, rychle a snadno získat data, jež potřebuji k naplnění parametrů volaných funkcí tříd.

Tento kód také obsahuje taktéž dvě funkce, které slouží k převodu jednotek délky a obsahu. Pro tento účel jsem místo tříd zvolil EventListenery s eventem "submit"s vnořenou arrow funkcí, která má kód potřebný k převodu zabudovaný k sobě.

```
const sqr = new Square();
squareForm.addEventListener("submit", (e) => {
  e.preventDefault();
  const data = new FormData(squareForm);
  const a = data.get("a");
  const unit = data.get("unitSquare");

  sqr.area(a, unit);
  sqr.circum(a, unit);
});
```

Příklad 34: PWA JS - main.js

Nejprve jsem vytvořil novou instaci třídy pomocí "new Square()"a uložil jsem jí do konstanty sqr. Následně jsem pomocí globální proměnné "squareForm"a metody "addEventListener"s eventem submit vytvořil základ pro vnořenou arrow funkci, která má parametr "e". Parametr "e"jsem použil v kombinaci s preventDefault, abych zabránil samovolnému načítání stránky.

Uvnitř arrow funkce jsem vytvořil proměnnou "data"a do ní jsem uložil nově vytvořenou instanci formu. Poté jsem vytvořil proměnnou do které jsem pomocí proměnné data s metodou get načetl hodnotu inputu se jménem "a". To samé jsem provedl v proměnné unit.

V poslední části kódu jsem implementoval funkce ze třídy a dosadil do parametrů nově získané data z proměnných, které jsou vytvořeny výše.

Takto postupuji i u ostatních obrazců a těles. Hlavní struktura zůstává stejná, mění se hlavně počet proměnných, které jsem vytvářel dle potřeby, na základě počtu parametrů funkcí tříd.

```
converter.addEventListener("submit", (e) => {
  e.preventDefault();

  const data = new FormData(converter);
  const firstInpt = data.get("firstInpt");
  const unitOne = data.get("unitConverterOne");
  const unitTwo = data.get("unitConverterTwo");

  function convert() {
    let resultOne = 0;
    //mm ----------------------------------------------------------------
    if (unitOne == "mm") {

      if (unitTwo == "mm") {
        resultOne = firstInpt;
        document.querySelector(".convResult").innerHTML = resultOne + "mm";
        console.log(resultOne);
      }
      if (unitTwo == "cm") {
        resultOne = firstInpt / 10;
        document.querySelector(".convResult").innerHTML = resultOne + "cm";
        console.log(resultOne);
      }
      if (unitTwo == "dm") {
        resultOne = firstInpt / 100;
        document.querySelector(".convResult").innerHTML = resultOne + "dm";
        console.log(resultOne);
      }
      if (unitTwo == "m") {
        resultOne = firstInpt / 1000;
        document.querySelector(".convResult").innerHTML = resultOne + "m";
        console.log(resultOne);
      }
      if (unitTwo == "km") {
        resultOne = firstInpt / 1000000;
        document.querySelector(".convResult").innerHTML = resultOne + "Km";
        console.log(resultOne);
      }
    }
```

Obrázek 14: JS - main.js - converter

Zde jsem vytvořil první z convertorů, jedná se o convertor délky. Metoda s vnořenou arrow funkcí je stále stejná, mění se však počet proměnných a hlavně jsem přidal Další vnořenou funkci "covert()", která slouží k převádění jednotek

pomocí jednoduchých podmínek a výpočtů.

Takto vypadá v podstatě i convertor obsahu, který jsem vytvořil. Jediným rozdílem však je, že nepoužívám vnořenou funkci na převod, ale volám si funkci s parametry.

```
576   converterArea.addEventListener("submit", (e) => {
577     e.preventDefault();
578     const data = new FormData(converterArea);
579     const firstInpt = data.get("firstInptArea");
580     const unitOne = data.get("unitConverterAreaOne");
581     const unitTwo = data.get("unitConverterAreaTwo");
582
583     convertArea(firstInpt, unitOne, unitTwo);
584   });
585
586 > function convertArea(firstInpt, unitOne, unitTwo) {···
752   }
```

Obrázek 15: JS - main.js - converter (2)

CSS

```
* {
  padding: 0;
  margin: 0;
  font-family: "Roboto", sans-serif;
}

img {
  display: none;
}

input,
button {
  border-radius: 1em;
}
```

Příklad 35: PWA - CSS

Pomocí selectoru "*"jsem provedl rest stylizace všech prvků, kdy jsem jim odebral defaultni padding a marging. Taktéž jsem přidal nastavení základního fontu pro všechny elementy. Pro mou aplikaci jsem zvolil font Roboto.

Následně jsem zakázal zobrazování obrázků, jelikož používám metodu stylování "mobile first", tedy začínám vždy nejprve s mobilním zařízením a končím desktopem.

Pro input a button jsem nastavil border-radius 1em a použil jsem k tomu obecný selector, abych všem inputům a tlačítkům nastavil vše jednotně.

```
h1 {
  margin-top: 20%;
  font-size: 3.4em;
}
h3 {
  opacity: 0.8;
}
```

```
h2 {
  margin-top: 5%;
  font-size: 1.9em;
}
h1,
h2 {
  font-weight: 900;
}
label {
  font-weight: 900;
}
```

Příklad 36: PWA - CSS

Nadpisům 1. úrovně jsem nastavil margin-top na 20% a vlastní velikost. Vše jsem nastavil po zkoušení různých hodnot. Margin-top jsem nastavil, aby nadpisy měli nad sebou více místa.

Pro nadpisy 2. úrovně jsem nastavil margin-top 5% a opět vlastní velikost, v tomto případě je velikost 1.9em. Podobně jako u nadpisů 1. úrovně jsem nastavil margin-top, aby vznikl větší prostor nad samotnými nadpisy.

U nadpisů 3. úrovně jsem chtěl docílit lepší vizuální separace od nadpisů 2. úrovně a proto jsem snížil opacity z 1 na 0.8, což mi lehce zesvětlí samotný nadpis a vytvoří větší hloubku.

Posléze jsem nadpisům 1. a 2. úrovně nastavil tučnost na 900, to znamená velmi tučný a to samé jsem provedl i u label.

```
input {
  border: none;
  background-color: rgba(0, 0, 0, 0.048);
  padding: 1.3em 1em;
}
```

Příklad 37: PWA - CSS (1)

U inputů jsem nastavil ohraničení na "none", to mi zaručí že inputy ne-

budou mít žádné ohraničení. Poté jsem nastavil barvu s danou průhledností, která se mi líbila nejvíce. Jako poslední jsem nastavil inputům padding, který dodá inputům větší prostor a vypadá z mého pohledu nejlépe

```
select {
  border: none;
  background-color: rgba(0, 0, 0, 0.048);
  padding: 1.3em 1em;
}
```

Příklad 38: PWA - CSS (2)

U selectů jsem nastavil to samé jako u inputů, abych docílil stejného vzhledu a aplikace tak vypadala hezky sjednoceně.

```
.swiper {
  width: auto;
  height: 100vh;
}
.installSection {
  z-index: 10000;
  position: absolute;
  margin: 1em 1em;
}
.install {
  background: rgb(88, 144, 247);
  z-index: 1000;
  width: 7em;
  height: 3em;
  border: none;
  font-weight: 900;
}
```

Příklad 39: PWA - CSS (3)

Zde jsem nastavil pro container ".swiper"z knihovny swiper.js automatické

nastavení šířky a výšu 100vh, aby na každém zařízení zabral 100% výšky zařízení.

U ".installSection"jsem nastavil z-index na velké číslo, aby tento div by vždy v úplném popředí. Poté jsem nastavil divu i position absolute, aby jej neovlivňoval žádný jiný element z html a dal jsem mu margin 1em.

Jako poslední v této části kódu, jsem provedl nastavení u tlačítka ".install", kdy jsem mu nastavil určitou barvu a opět z-index na vysokou hodnotu. Následně byla potřeba nastavit výška a šířka tlačítka, kdy jsem zvolil hodnoty 7em a 3em. Poslední v řadě jsem nastavil border na "none"a tučnost písma na 900.

```
.container {
  height: 90vh;
  display: flex;
  padding: 4% 13%;
  gap: 1.5em;
  align-items: center;
  flex-direction: column;
}
.section {
  display: flex;
  flex-direction: column;
}
.subcontainer {
  width: 100%;
  display: flex;
  flex-direction: column;
  gap: 0.75em;
}
```

Příklad 40: PWA - CSS (4)

V této části kódu byla potřeba nastavit u divu ".container"výška společně s displaye flex, aby všechny elementy uvnitř divu bylo možné zarovnat pomocí

flex-box příkazů. Jelikož jsem použil display flex, mohl jsem tím pádem použít i gap a nastavit tak mezi elementy mezeru pro lepší vzhhled a prostorové uspořádání. Poslední jsem přidal divu padding.

Dále jsem divu ".section"provedl nastavení na display flex a směr flexu na sloupec.

Pro ".subcontainer"byla potřeba nastavit šířku na 100%, display na flex a směr flexu na sloupec. Také jsem přidal nastavení gap, pro lepší prostorové rozvržení.

```
form {
  margin-top: 5%;
  display: flex;
  flex-direction: column;
  gap: 1em;
}
form button {
  height: 4em;
  border: none;
  background-color: rgb(131, 211, 194);
  font-weight: 900;
  cursor: pointer;
  font-size: 0.8em;
}
```

Příklad 41: PWA - CSS (5)

Pro formy (formuláře) jsem nastavil margin-top na 5%, kvůli většímu odsazení od nadpisu. Potřeba byla také nastavit display na flex, aby se elementy uvnitř formu daly dobře zarovnat. Posléze jsem přidal i nastavení směru flexu na sloupec. Poslední jsem přidal gap.

V druhé části kódu jsem pomocí selectoru vybral všechny tlačítka, co jsou uvnitř formulářů a nastavil jsem jim určitou výšku, border na none, určitou barvu, velikost fontu a tučnost fontu. Taktéž jsem nastavil, aby při najetí myší na tlačítko byl ukazovací kurzor.

```
.swiper-pagination-bullet-active {
  background-color: rgb(131, 211, 194) !important;
  transform: scale(1.5);
}
.swiper-pagination {
  background-color: rgba(0, 0, 0, 0.048);
  padding: 0.1em 0.3em;
  border-radius: 1em;
}
```

Příklad 42: PWA - CSS (6)

Tato část kódu je zaměřená na přizpůsobení vzhledu swiper.js komponentů. Těmto komponentům jsem nastavil především jiné barvy, velikosti a padding, jelikož defaultní nastavení a barvy mi nevyhovovaly.

```
.selCont {
  display: flex;
  width: auto;
}
.selCont input {
  width: 100%;
}
.selContTwo {
  display: flex;
  margin-top: 5%;
  font-size: 2em;
  align-items: center;
  justify-content: center;
}
```

Příklad 43: PWA - CSS (7)

Div ".selCont"slouží jako kontejner selectů u convertorů (délky a obsahu), proto bylo třeba nastavit display flex pro lepší práci s layoutem selectů. Pro input uvnitř divu jsem pro lepší vzhled nastavil šířku na 100%.

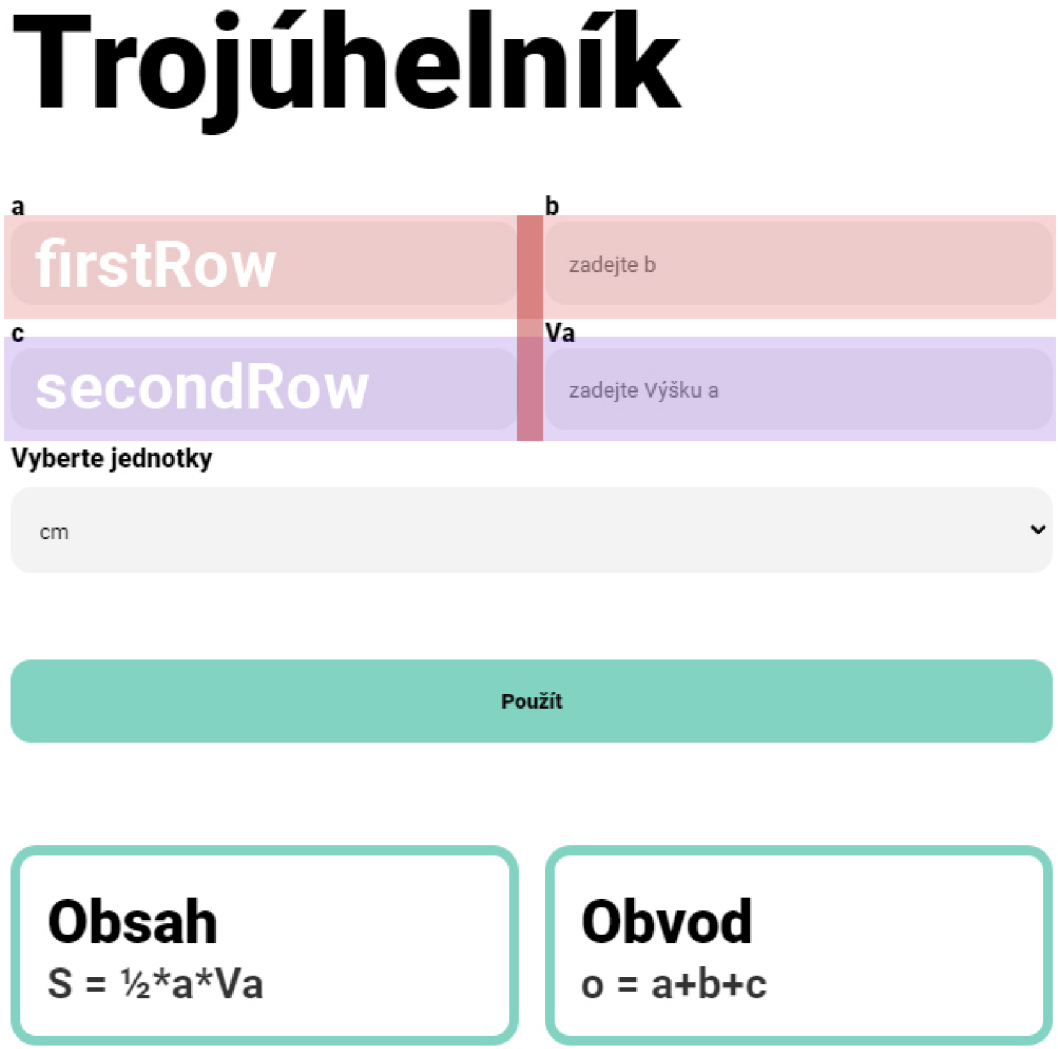
V druhé polovině kódu jsem musel pro div ".selContTwo"nastavit opět display flex. Následovalo nastavení nmargi-top, font-size a nakonec jsem použil flex-box a příkazy k nastavení zarovnání. Tento div slouží jako kontejner pro zobrazení výsledků konvertorů.

```
.firstRow {
  display: grid;
  grid-template-columns: repeat(2, calc(50% - 0.5em));
  gap: 1em;
}
.secondRow {
  display: grid;
  grid-template-columns: repeat(2, calc(50% - 0.5em));
  gap: 1em;
}
.firstRow div {
  display: flex;
  flex-direction: column;
}
.secondRow div {
  display: flex;
  flex-direction: column;
}
```

Příklad 44: PWA - CSS (8)

U divu ".firsRow"a ".secondRow"bylo zapotřebí nastavit display grid, jelikož se pomocí gridu lépe zarovnávají elementy ve více směrech. Díky tomu, že jsem se rozhodl použít grid, mohl jsem tím pádem použít grid-template-columns, u kterého jsem nastavil, aby se dvakrát opakovali sloupce, přičemž každý sloupec má šířku 50% - 0.5em. Od 50% jsem odečítal 0.5em, abych kompenzoval mezeru mezi sloupci 1em.

Pro divy uvnitř ".firstRow"a ".secondRow"jsem nastavil display flex a zarovnání elementů pod sebe pomocí flex-direction column.



Obrázek 16: PWA-CSS- firstRow, secondRow

```
.subgroup {
  display: flex;
  flex-direction: column;
}
```

Příklad 45: PWA - CSS (9)

Tento kód jsem nastavil divu ".subgroup", který je kontejnerem pro input a label v určitých formulářích. Nastavil jsem mu display flex a flex-direction column, aby byl vždy label nad inputem a já mohl pak tyto samotné "podskupiny"opět zarovnávat v jiných kontejnerech.

```
.ios {
  display: none;
  border-radius: 1em 1em 0 0;
  bottom: 0;
  z-index: 10000;
  background-color: rgb(224, 224, 224);
  width: 100%;
  max-height: 50vh;
  position: absolute;
}
.ios ol {
  padding: 11% 18% 15% 18%;
  display: flex;
  flex-direction: column;
  gap: 1.5em;
}
```

Příklad 46: PWA - CSS (10)

Část, která je zde zobrazená, slouží ke stylování divu ".ios". Tento slouží jako návod pro instalaci na ios. Přidal jsem mu dipsplay none, jelikož viditelnost upravuji v JavaScriptu pro konkrétní zařízení a okna. Taktéž jsem zakulatil horní dva rohy divu pomocí border-radius a celý div jsem umístil na spodek obrazovky. Protože chci, aby byl návod vždy v úplném popředí, tak jsem přidal

z-index s velmi vysokým číslem. Následně jsem nastavil divu ještě maximální výšku, kterou může mít a absolutní pozici. Samozřejmě jsem mu nastavil i šířku na 100%.

Uvnitř divu se nachází ol (ordered list), kterému jsem nastavil padding, mezeru mezi elementy a zarovnání na sloupec.

```
.ios h4 {
  font-weight: 900;
  font-size: 1.1em;
}
.ios li {
  margin-left: 8%;
}
.ios span {
  display: flex;
  align-items: center;
  gap: 0.5em;
}
```

Příklad 47: PWA - CSS (11)

Pro div ".ios"jsem udělal ještě několik dalších nastavení týkajících se především nadpisu a li (list itemů). Pro nadpis 4. úrovně jsem vybral tučnost 900 a velikost písma 1.1 em. Samotných list itemům jsem nastavil lehké odsazení od levého okraje divu.

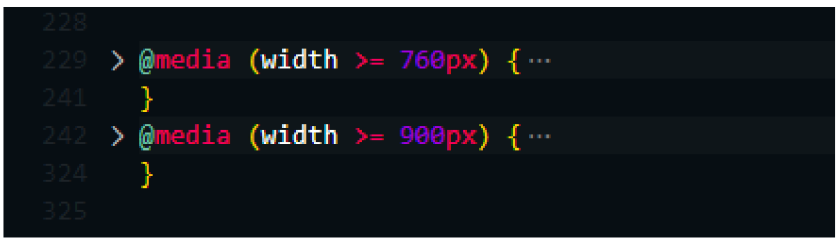
Poslední jsem nastavoval ".ios span", což je v podstatě "pseudo kontejner", který mi posloužil k vertikálnímu zarovnání ikon s textem. Vertikálního zarovnání jsem docílil pomocí nastavení display flex a align-items center. Na závěr jsem přidal mezi text a ikony mezeru pomocí gap.

```css
.close-btn {
  position: relative;
  width: 8%;
  margin-top: 3%;
  left: 89%;
}
.share-ios {
  width: 12%;
  text-align: center;
}
.addTo {
  width: 10%;
}
```

Příklad 48: PWA - CSS (12)

Protože byly ikony moc velké, musel jsem jim nastavit správné velikosti. U ".close-btn"jsem nastavil velikost na 8% u ".share-ios"na 12% a u ".addTo"na 10%. Velikosti jsem nastavoval pomocí width, jelikož obrázky většinou zachovávají poměr stran.

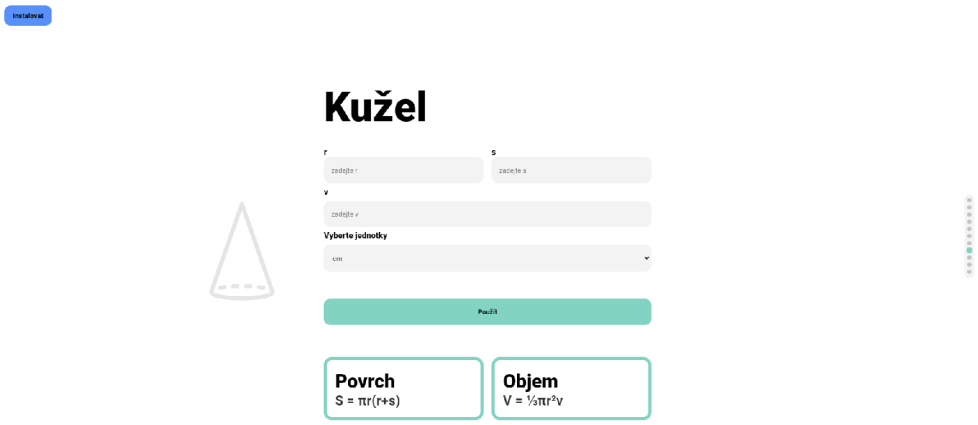
Pro ikonu ".close-btn"jsem ještě musel upravit její umístění, chtěl jsem ať je umístěná v pravém horním rohu a dosáhl jsem toho pomocí position relativ, kdy je pozice relativní kontejneru ve kterém se ikona nachází. Následně jsem mohl použít left a doladil jsem pomocí margin-top.



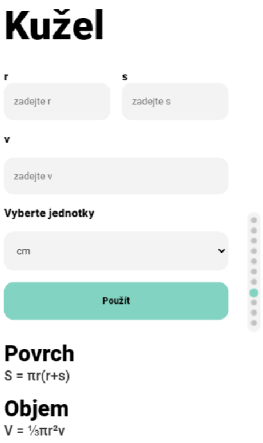
Obrázek 17: PWA-CSS-breakpoints

Ve finální fázi CSS jsem pouze ladil velikosti a odsazení různých elementů na základě velikosti obrazovek. Také jsem pro obrazovky větší 900px přidal

obrázky 2D a 3D útvarů. Layout formulářů však zůstává v podstatě stejný. Obecně jsem ze stránky designu nic neměnil, aby aplikace zůstala jednotná a čistá.



Obrázek 18: PWA-CSS-počítač



Obrázek 19: PWA-CSS-mobil

5.3.3 WA

WA (Web app) jsem vytvořil na stejném principu a základu jako PWA, tedy WA je schopno opět vypočítávat obsahy, obvody, objemy a povrchy jak 2D útvarů tak 3D těles. WA jsem musel vytvořit z co největší části stejně jako PWA, abych mohl porovnat tyto technologie mezi sebou.

HTML

HTML je v podstatě stejné jako u PWA, ale chybí v něm elementy pro instalační tlačítko a taktéž v něm chybí div ".ios", který u PWA slouží jako návod k instalaci na iOS. Tyto elementy jsem ve WA odebral, protože nejsou potřebné.

JavaScript

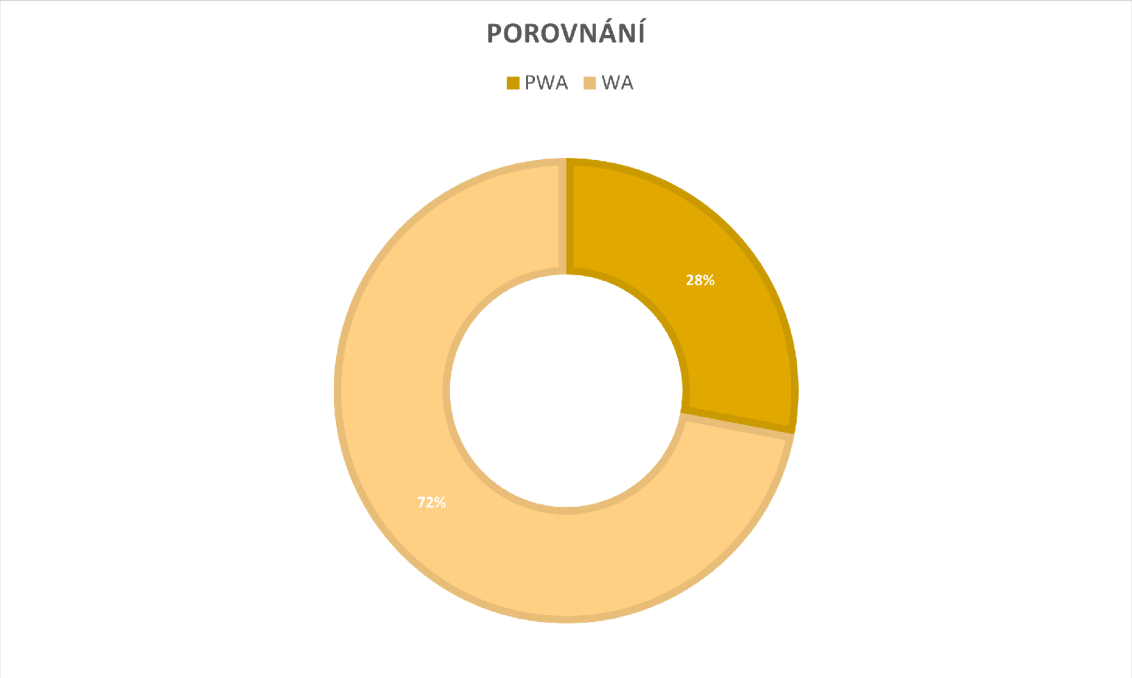
U JS je to však větší rozdíl, samotná logika aplikace je stejná. Tedy main.js je úplně stejný jako u PWA, ale to je poslední věc co mají společnou. V samotném JS WA se nenachází detection.js a serviceworker.js, protože u WA není v mém případě potřeba taktéž rozeznávat zařízení a samozřejmě ani serviceworker není potřebný.

CSS

V CSS je vše úplně stejné, až na barvu tlačítek, která byla zvolena jiná. Nestyluji v něm však div ".ios"s jeho elementy a instalační tlačítko, tím pádem se CSS WA moc neliší od PWA.

5.4 Průzkum

V rámci mého menšího průzkumu jsem zkoumal rozšíření technologie PWA v reálném světě. Navštívil jsem 25 náhodných webů a zjistil jsem, zda se jedná o klasickou WA nebo jestli se jedná o technologii PWA.



Obrázek 20: Graf rozšířenosti

Z 25 stránek, které jsem zvolil naprosto náhodně, jich bylo 7 PWA a 18 WA. Může se tedy zdát, že technologie není příliš rozšířená. Pravdou však je, že na to jak je "mladá"tato technologie, tak 7/25 je naprosto úžasný výsledek.

Taktéž jsem zjistil, že PWA používají především obchody a komerční společnosti. Což není zas tak velké překvapení, jelikož PWA je levnější na vývoj než nativní aplikace.

	PWA	WA
YouTube.com	x	
seznam.cz		x
slevomat.cz	x	
heureka.cz		x
mall.cz	x	
alza.cz		x
czc.cz		x
wikipedia.cz		x
zdrojak.cz		x
codecademy.com	x	
datart.cz		x
aboutyou.cz	x	
zalando.cz		x
fooshop.cz	x	
sizeer.cz		x
facebook.com		x
idnes.cz		x
aktualne.cz		x
novinky.cz		x
tescoma.cz		x
csfd.cz		x
imdb.com		x
twitch.tv		x
unsplash.com	x	
stackoverflow.com		x

Obrázek 21: Tabulka k průzkumu

6 Závěr

Bakalářská práce se zabývala technologií PWA (Progressive Web App) s využitím serviceworkeru, která slouží k tvorbě progresivních webových aplikací.

V teoretické části jsem se soustředil na představení a popsání samotné technologie PWA a serviceworkeru. Popsal jsem postupně všechny důležité prvky, které PWA musí obsahovat. Následně jsem rozebral samotné komponenty PWA a na konec jsem taktéž porovnal PWA s nativními aplikacemi a webovými aplikacemi.

V praktické části jsem vytvořil 2 aplikace PWA a WA, abych je mezi sebou mohl dobře porovnat. Aplikace jako takové slouží k výpočtům týkajících se především 2D obrazců (čtverec, trojúhelník atd.) a 3D těles (krychle, kvádr). Obě aplikace byly vytvořeny tak, aby měly ten samý účel a vzhled, liší se od sebe hlavně samotnou funkčností a technologickými možnostmi. Výsledkem je webová stránka s jednoduchým menu, kde je možné otevřít jak PWA tak WA.

Na konci praktické části jsem udělal i menší průzkum rozšířenosti technologie PWA, kdy jsem pomocí náhodného navštěvování webových stránek zjistil, že z 25 webů jich 7 používá technologii PWA.

Technologie PWA je velmi nová a perspektivní. Postupně se začíná více a více rozšiřovat jak v komerční sféře tak i ve sféře zábavy. Nechci předem predikovat, jak tato technologie dopadne, ale myslím si že její možnosti jsou ohromné a velmi převyšují klasické WA. Už jen fakt, že giganti typu Google začínají používat tuto technologii o něčem svědčí.

Po mé zkušenosti s PWA a WA musím říci, že PWA je v mnoha ohledech lepší než klasická webová stránka. Ať už se jedná o rychlost, možnost instalace na plochu nebo o offline funkčnost. Tato technologie je mnohem perspektivnější a troufám si říci že i lepší než WA a obecně klasické webové stránky.

Seznam použité literatury a zdrojů

[1] HTML: HyperText Markup Language | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[2] Značkovací jazyk HTML | Web. Tvorba webu – HTML, CSS, JS | Web [online]. Dostupné z: https://web.vavyskov.cz/znackovaci-jazyk.html

[3] Webdesignérův průvodce po HTML5 - díl nultý - Zdroják. Zdroják - o tvorbě webových stránek a aplikací [online]. Dostupné z: https://zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-dil-nulty/

[4] HTML5: co přináší a proč se o něj zajímat - Root.cz. Root.cz - informace nejen ze světa Linuxu [online]. Copyright © 1997 [cit. 25.03.2023]. Dostupné z: https://www.root.cz/clanky/html5-co-prinasi-a-proc-se-o-nej-zajimat/

[5] HTML & CSS - W3C. World Wide Web Consortium (W3C) [online]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss.html

[6] What is CSS3? | Uses & Need | Features and Advantages of CSS3. EDUCBA | Best Online Training & Video Courses Certification [online]. Copyright © 2023 [cit. 25.03.2023]. Dostupné z: https://www.educba.com/what-is-css3

[7] WEB & MOBILE DEVELOPMENT AGENCY | Rascasone [online]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-javascript-pro-zacatecniky

[8] Javascript - úvod. Jak psát web, návod na html stránky [online]. Dostupné z: https://www.jakpsatweb.cz/javascript/javascript-uvod.html

[9] JSON : jednotný formát pro výměnu dat - Zdroják. Zdroják - o tvorbě webových stránek a aplikací [online]. Dostupné z: https://zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/

[10] Working with JSON - Learn web development | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

[11] Basics of the Index.html Default Webpage. ThoughtCo.com is the World's Largest Education Resource [online]. Dostupné z: https://www.thoughtco.com/index-html-page-3466505

[12] [online]. Dostupné z: https://www.pluralsight.com/blog/creative-professional/whats-stuff-top-html5-document

[13] HTML basics - Learn web development | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[14] CSS Tutorial. W3Schools Online Web Tutorials [online]. Dostupné z: https://www.w3schools.com/css/

[15] Service Worker API - Web APIs | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

[16] Service worker overview - Chrome Developers. Chrome Developers [online]. Dostupné z: https://developer.chrome.com/docs/workbox/service-worker-overview/

[17] Service workers. web.dev [online]. Dostupné z: https://web.dev/learn/pwa/service-workers/

[18] Add a web app manifest. web.dev [online]. Dostupné z: https://web.dev/add-manifest/

[19] manifest.json - Mozilla | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

[20] Uniquely identifying PWAs with the web app manifest id property - Chrome Developers. Chrome Developers [online]. Dostupné z: https://developer.chrome.com/blog/pwa-manifest-id/

[21] scope - Web app manifests | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Manifest/scope

[22] Web app manifests | MDN. [online]. Copyright ©1998 [cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Manifest

[23] Complete Guide To PWA: Definition, Technology, Real Examples. Magento Agency | Certified Magento Experts | Scandiweb [online]. Copyright © 2022 scandiweb. All Rights Reserved. [cit. 25.03.2023]. Dostupné z: https://scandiweb.com/blog/learn-all-about-progressive-web-apps/

[24] Progressive Web Apps. web.dev [online]. Dostupné z: https://web.dev/learn/pwa/progressive-web-apps/

[25] Progressive Web Apps 101: the What, Why and How. [online]. Dostupné z: https://www.freecodecamp.org/news/progressive-web-apps-101-the-what-why-and-how-4aa5e9065ac2/

[26] Overview of Progressive Web Apps (PWAs) - Microsoft Edge Development | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 25.03.2023]. Dostupné z: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/

[27] Web App vs Progressive Web App: Why are they different from each other?. Agile Web Development Services | CSSChopper [online]. Copyright © CSSChopper 2023 [cit. 25.03.2023]. Dostupné z: https://www.csschopper.com/blog/web-app-vs-progressive-web-app/

[28] Progressive web app structure - Progressive web apps (PWAs) | MDN. [online]. Copyright © js13kGames 2012

[cit. 25.03.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/App_structure

[29] Progressive Web Apps Vs Responsive Websites: How to Choose. Designial- UX Research & Testing | Product & UX Strategy | Talent [online]. Copyright © Copyright 2023 [cit. 25.03.2023]. Dostupné z: https://designial.com/blogs/progressive-web-apps-vs-responsive-websites-how-to-choose

[30] Difference between Progressive Web Apps and Regular Web Apps. [online]. Dostupné z: https://www.websitepulse.com/blog/progressive-web-app-different-from-regular-web-app

[31] PWA VS Native App: What To Choose In 2023?. SolveIt | Full-Cycle Software Development Company [online]. Copyright © 2016 [cit. 25.03.2023]. Dostupné z: https://solveit.dev/blog/progressive-web-app-vs-native-app

[32] PWA vs Native App – Differences and Similarities for Business, Performance and User Experience | ASPER BROTHERS. Software Development Company - ASPER BROTHERS [online]. Dostupné z: https://asperbrothers.com/blog/pwa-vs-native-app/

[33] How Do Progressive Web Apps Really Compare to Native Apps?. Convert Your Site to Native Mobile Apps in Days with MobiLoud [online]. Dostupné z: https://www.mobiloud.com/blog/progressive-web-apps-vs-native-apps

[34] PWA vs Native: Progressive Web Apps vs Native Apps - Teknicks. Teknicks: Product-Led Growth (PLG) Marketing Agency [online]. Copyright ©2023 Teknicks [cit. 25.03.2023]. Dostupné z: https://www.teknicks.com/blog/pwa-progressive-web-app-vs-native-apps-which-is-better/

[35] CO JSOU PROGRESIVNÍ WEBOVÉ APLIKACE (PWA) A JAKÉ MAJÍ VÝHODY [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody

[36] CO JE TO VLASTNĚ PWA? [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.bootiq.io/progressive-web-application-pwa/

[37] Progresivní webové aplikace: Co to je? A jak webu zařídit plné hodnocení PWA v Lighthouse. Vzhůru dolů – webová kodéřina ze všech stran [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.vzhurudolu.cz/prirucka/pwa

[38] RICHARD, Sam a Pete LEPAGE. What are Progressive Web Apps?. web.dev [online]. 1 [cit. 2022-03-30]. Dostupné z: https://web.dev/what-are-pwas/

[39] GATTERMAYER, Josef. Proč a jak psát progresivní webové aplikace [online]. 1 [cit. 2022-03-30]. Dostupné z: https://www.ackee.cz/blog/proc-a-jak-psat-progresivni-webove-aplikace

Seznam příkladů

Seznam obrázků

A Příloha

Web https://vachutabp.netlify.app/