Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Využití PWA přístupu ve tvorbě softwarových aplikací

Vlastimil Lisák

© 2023 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Vlastimil Lisák

Informatika

Název práce

Využití PWA přístupu ve tvorbě softwarových aplikací

Název anglicky

Software application development using PWA approach

Cíle práce

Cílem práce je objasnit principy progresivních webových aplikací (PWA) a na citacích i vlastním příkladu potvrdit, že PWA přístup je vhodný v případech cross-platformových aplikací s uživatelským rozhraním. Na vlastní příkladu se ukáže jednoduchost vývoje PWA a prokáží nízké náklady na údržbu a demonstruje problematika tvorby PWA.

Metodika

První část práce bude teoretická s analýzou a průzkumem možností současných technologií z dokumentací, knih a odborných článků. Druhá část práce bude praktická, která na základě na základě poznatků z teoretické části pomocí příkladu popíše tvorbu plně funkční PWA včetně posouzení nákladů na vývoj a údržbu.

Doporučený rozsah práce

40 – 60 stran

Klíčová slova

PWA (progressive web application); front-end technology; off-line application; HTML; CSS; JavaScript; efektivita vývoje IS

Doporučené zdroje informací

Javascript okamžitě (2014), A. Pehlivanian, D. Nguyen, ISBN: 978-80-251-4163-2
The PWA Book (2019), P. Karwatka, A. Kwiecień, F. Rakowski, K. Grzybowska, https://www.divante.com/pwabook
Vzhůru do CSS3 (2017), M. Michálek, ISBN: 978-80-260-8440-2
Vzhůru do (responzivního) webdesignu (2017), M. Michálek, ISBN: 978-80-88253-00-6

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 05. 03. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Využití PWA přístupu ve tvorbě softwarových aplikací" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15. 3. 2023 _____

Poděkování

Rád bych touto cestou poděkoval docentovi Ing. Vojtěchu Merunkovi PhD. za vynikající rady ohledně psaní Bakalářské práce.

Využití PWA přístupu ve tvorbě softwarových aplikací

Abstrakt

V této práci je popsán problém náročnosti vývoje nativních multiplatformních aplikací. Jako teoretické řešení je uvedena alternativa vyvíjení progresivních webových aplikací, které mají ty nativní ve většině případech nahradit. Dále je zde popsán vývoj ukázkové aplikace osobního asistenta jako příklad progresivní webové aplikace.

Teoretická část se zabývá popisem problematiky multiplatformních aplikací, jejich údržby a popsáním technologií, které jsou zapotřebí k vytvoření progresivní webové aplikace. Nejdříve popisuje různé možnosti k vytvoření nativní aplikace pro více zařízení, dále se zabývá náročností tohoto přístupu. Dále je popsán PWA přístup k vytvoření multiplatformních aplikací a výhody tohoto přístupu. Následuje popis klasických webových aplikací a technologií umožňujících vytvoření PWA.

Praktická část obsahuje postup vytvoření PWA. Nejdříve byl proveden návrh UIX, slovy popsaný a navrhnutý ve vektorové grafice. Dále je popsán samotný vývoj kódu, byly zde ukázány důležité funkční části kódu. Závěrem je popsáno zhodnocení vyvinuté aplikace, její rychlost, testování a návrhy na vylepšení.

Klíčová slova: PWA – Progresivní webová aplikace; front-end technologie; off-line aplikace; HTML; CSS; JavaScript

PWA approach used in the software application development.

Abstract

In this thesis, there are described problematics and difficulties of native cross-platform applications. The theoretical solution is the development of progressive web applications, that should replace the native applications in most cases. The development of a personal assistant demo application is described here as an example of a progressive web application.

The theoretical part deals with the description of multiplatform applications and their maintenance, describing the technologies that are needed to create a progressive web application. It first describes the various options for creating a native application for multiple devices, then discusses the cost and complexity of this approach. Next, the PWA approach to creating multiplatform applications and the advantages of this approach are described. The following is a description of the classic web applications and the technologies enabling the creation of PWA.

The practical part contains the procedure of creating PWA, first, the UIX design was performed, described in words, and designed in vector graphics. Furthermore, the development of the code itself is described, and important functional parts of the code are shown here. Finally, the evaluation of the developed application, its speed, testing, and suggestions for improvement are described.

Keywords: PWA – progressive web application; front-end technology; off-line application; HTML; CSS; JavaScript

Obsah

1 Úvod

Tato práce představuje koncept PWA – progresivních webových aplikací jako levnější a snadnější alternativu oproti multiplatformním nativním aplikacím.

V dnešní době jsou lidé zvyklí mít co nejrychlejší přístup k informacím a aplikacím, které jim usnadňují život, a to co nejpohodlnějším způsobem, ideálně s přístupem ze všech jimi vlastněných zařízení.

Vývoj nativních multiplatformních aplikací je velice komplexní záležitost, která potřebuje spoustu času schopných programátorů, díky tomu vývoj a následná údržba zůstává velmi nákladná. Tato skutečnost majiteli komerční aplikace snižuje ROI – return on investment.

Jak tedy vyřešit problém optimalizace již zmíněného ROI? Autor této práce věří, že řešením je, pro mnoho případů, využití přístupu PWA – progresivní webové aplikace. Tento přístup zákazníkovi či uživateli umožňuje, aby si aplikaci s vysokou rychlostí načetl na jakémkoliv místě, a to buď pomocí Wi-fi, pevného připojení na internet či mobilních dat, do jakéhokoliv zařízení, které je uživatelem aktuálně používáno na webovém prohlížeči. Pokud bude s aplikací spokojen, může si ji jednoduše ihned v prohlížeči stáhnout a nainstalovat.

Autor se domnívá, že nejen zvýšený počet stažení a používání vyvinuté komerční aplikace zvýší ROI, ale především tomu napomůže drastické snížení nákladů na vývoj PWA, které na rozdíl od vytváření nativních verzí aplikací pro různá zařízení efektivně snižují velikost celkového napsaného kódu. Tím pádem šetří i čas programátorům, a také snižují náročnost údržby takových aplikací, jelikož stačí pouze jeden zdrojový kód, který bude fungovat na všech koncových zařízeních.

2 Cíl práce a metodika

2.1 Cíl práce

Tato práce má za cíl objasnit principy progresivních webových aplikací (PWA) a potvrdit, že má smysl využít PWA ve většině případech použití desktopových i mobilních aplikací s uživatelským rozhraním.

Dílčí cíle práce jsou:

- Prozkoumat současné možnosti a technologie.
- Vytvořit plně funkční PWA.
- Zhodnotit nákladnost a technickou náročnost vývoje a údržby PWA oproti nativnímu vývoji softwaru.

2.2 Metodika

Práce se skládá z teoretické a praktické části. V teoretické části se budou analyzovat aktuální technologie a vzor vývoje multiplatformních aplikací. V praktické části bude vytvořena PWA osobního asistenta. Aplikace bude technicky implementována pomocí HTML, CSS a JavaScriptu.

V teoretické části budou analyzované současné technologie z dokumentací a odborných článků a z odborné literatury na téma progresivních webových aplikací a webových front-end i back-end technologií obecně. Bude představeno paradigma při vývoji multiplatformních aplikací, popsány veškeré problémy s vývojem a náklady na tento vývoj.

Na základě poznatků z teoretické části bude v praktické části vytvořena plně funkční PWA sloužící jako osobní asistent, která bude funkční i off-line po nainstalování aplikace, využívající 5 MB velkého lokálního úložiště poskytnutého prohlížečem na koncovém zařízení. Tento asistent bude umět vytvářet úkoly, o kterých bude mít uživatel v aplikaci přehled, stejně tak jako možnost zapisovat si své poznámky. Aplikace bude plně upravitelná uživatelem z hlediska vzhledu – tmavý režim, barvy atp.

Aplikace bude tzv. single-page. To znamená, že veškerý obsah nutný k fungování této aplikace bude pouze na jediné stránce – souboru index.htm. Při přepínání mezi nastavením, úkoly a poznámkami tudíž nebude docházet k žádnému přesměrování, ale na stránce se bude dynamicky měnit viditelný obsah, se kterým uživatel interaguje, stejně jako v nativních off-line aplikacích.

Technická implementace bude provedena v jazyce HTML, CSS a skriptovacím jazyce JavaScript, v této aplikaci nebude potřeba back-end. Veškerá data bude mít uživatel v již zmíněném lokálním úložišti poskytnutým webovým prohlížečem na svém zařízení.

Závěrem bude syntéza výsledků teoretické a praktické části, dále zde bude posouzen průběh vývoje aplikace, a také budou navrženy změny pro vylepšení aplikace.

3 Teoretická východiska

3.1 Problémy při vývoji nativní multiplatformní aplikace

3.1.1 Velikost „codebase“

Když společnost potřebuje vyvinout aplikaci, která bude svým uživatelům umožňovat přístup z různých zařízení – myšleno z různých platforem operačních systémů (jmenovitě u osobních počítačů například MacOS, Linuxové distribuce, Windows; u mobilních zařízeních iOS, Android), musí vytvořit velmi složitý a hlavně obsáhlý „codebase“ – kolekci zdrojového kódu, která je zapotřebí k vytvoření této multiplatformní aplikace.

Příklady „codebase“ známých nativních aplikací:

- Facebook (veškeré aplikace) – 54 GB (Aboukhadije, F., 2014)
- Google (veškeré aplikace) – 86 TB (Potvin, R. and Levenberg, J., 2016)
- Uber (veškeré aplikace) – 25 milionů řádků kódu (Parwal, A. and Karuppaiya, K., 2021)

3.1.2 Rozmanitost „codebase“

Pokud se jedná o aplikaci, která bude nejen fungovat na více typech zařízení, jako jsou telefon, tablet a osobní počítač, ale zároveň bude schopná fungovat na více operačních systémech daných zařízení (např. Linuxové distribuce, iOS, Android a Windows), musí se zvolit více programovacích jazyků, které zajistí chod aplikace v jednotlivých operačních systémech.

Na demonstraci komplikací, které jsou spojené s front-endem, je uvedena nativní aplikace, u které je požadována funkčnost jen na chytrých mobilních telefonech. Statistika (StatCounter Global Stats, 2022) uvádí, že 99,27 % vlastníků chytrých mobilních zařízení využívá dva operační systémy – iOS a Android.

Nejdříve je nutné najmout do týmu minimálně dva různé programátory, kteří budou schopni programovat v programovacích jazycích pro tyto operační systémy (pokud jeden zvládá jazyky pro oba operační systémy, není nutné shánět druhého). Pro operační systém iOS je

na výběr ze dvou programovacích jazyků – Objective-C (popřípadě analogicky rozšířené Objective-C++) nebo Swift, speciálně určený pro vývoj (nejen) na platformě iOS. Oba jazyky jsou objektově orientované a umožňují vyvinout verzi aplikace pro iOS. Druhá verze bude pro uživatele operačního systému Android, programátoři mají na výběr opět ze dvou široce používaných možností, mohou zvolit programovací jazyk Java nebo Kotlin (Luetic, M., 2021). Kotlin je od roku 2019 preferovaný společností Google vlastnící operační systém Android. Google uvedl, že vývoj na Android bude čím dál tím více především přes Kotlin, takže programovat aplikaci na Android v dnešní době dává mnohem větší smysl právě v tomto jazyce (Roomi, M., 2021).

Při vývoji této aplikace jsou tedy hned na začátku na výběr 4 možné kombinace, jen co se týče výběru programovacího jazyka. Další problém je nutnost dvou verzí této aplikace, každá pro daný operační systém. Pokud bude po programátorech požadováno například vyvinutí nové funkce, musí oba psát dva různé podprogramy vykonávající tu stejnou činnost, v ideálním případě je pak i otestovat a až následně je možné je vydat v aktualizačním balíčku.

3.1.3 Umístění do obchodu s aplikacemi

Obchod s aplikacemi (anglicky „App store“) je známý koncept především díky proslavení značkou Apple, která v roce 2008 založila svůj obchod „Apple App Store“. Ten obsahoval aplikace pro mobilní zařízení iPhone.

Obchod s aplikacemi je digitální místo, kam mohou vývojáři publikovat své aplikace a nabídnout je tak široké veřejnosti. Aplikace mohou být uvedeny zadarmo či za poplatek. Původně tento koncept vzniknul již na začátku devadesátých let obchodem „Electronic App Wrapper“, který je považován za první software tohoto typu (Carey, R., 2015).

Při vývoji nativní multiplatformní aplikace, konkrétně v kroku uveřejnění aplikace na trh, se bez těchto obchodů dá velmi těžko obejít v rámci přístupu uživatelů na mobilních zařízeních k takové aplikaci.

Autor zde navazuje na příklad z předchozích kapitol, ve kterém byl zatím vysvětlen dosavadní postup při vývoji nativní multiplatformní aplikace, zvolení programovacích jazyků v prvním problému a vývoj verzí pro jednotlivé platformy (myšleno OS) v problému druhém. Třetí problém se týká uveřejnění nativní multiplatformní aplikace na obchod s aplikacemi pro danou platformu mobilního zařízení.

Je vyvíjena nativní multiplatformní aplikace pro Android a iOS, obě verze jsou již hotové a zbývá vyřešit již zmiňovaný vstup na obchody s aplikacemi pro tyto platformy. K dispozici je „Google Play" pro operační systém Android a „Apple App Store" pro platformu iOS. Co se týče ekonomického aspektu u placených aplikací, obě firmy mají stejná pravidla. 70 % získá vlastník aplikace a 30 % obchod (Zdraveska, S., 2022). U Google Play musí vývojář počítat, že zaplatí 25 USD za zveřejnění aplikace. Výhodou ovšem zůstává, že Google Play má jednodušší a rychlejší proces schvalování aplikace, vývojáři tedy mohou být kreativnější a nemusí trávit tolik času upravováním aplikace ke správným „měřítkům" obchodu. To s sebou nese i nevýhodu v tom, že pokud jsou v aplikaci nějaké chyby, Apple App Store poskytne lepší zpětnou vazbu (Viswanathan, P., 2020).

Pro oba operační systémy existuje i možnost obejít nutnost zveřejnění aplikace uživatelům dané platformy, ale to s sebou nese značné nevýhody. U Androidu je tato možnost více benevolentní (jedná se o open-source platformu). Systém Android ve výchozím nastavení sice nepovoluje možnost instalovat aplikační balíčky (.apk), nicméně pokud si uživatel stáhne tento soubor s aplikací demonstrovanou na příkladu, tak mu je i přes výchozí nastavení operačního systému poskytnuta po otevření souboru možnost povolit instalaci aplikací z neznámých zdrojů – mimo Google Play. Po povolení se instalace spustí a aplikace bude fungovat, i když nepochází z obchodu Google Play (Hindy, J., 2021).

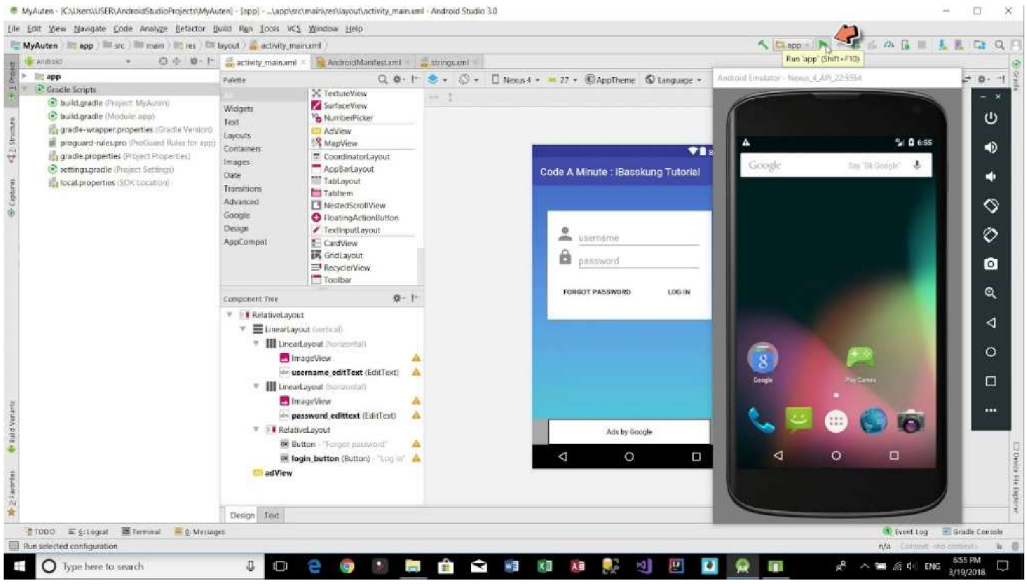
Existují i obchody pro platformy na osobní počítače. Pro Windows 8 a vyšší je k dispozici předinstalovaný Microsoft Store (Microsoft Apps, n.d.). Na počítačích s MacOS se od verze OS X 10.6.6 nachází Mac App Store (Stahování na iTunes (Mac App Store), n.d.).

3.1.4 Konkluze

V předchozích kapitolách se čtenář seznámil s problémy, kterým se musí čelit při vývoji nativní aplikace pro více platforem. V této kapitole budou všechny kroky shrnuty a zobrazeny v diagramu (viz obrázek 2).

Největší oblastí diagramu je *codebase* aplikace, to jest veškerý kód, který je potřeba pro její chod. *Codebase* se dělí na dvě části, front-end a back-end. Front-end je kód, který je určen pro koncová zařízení, na kterých bude aplikace spuštěna. Může to být Windows či MacOS počítač, nebo jakékoliv mobilní zařízení s operačním systémem Android či iOS. Front-endový kód má klíčovou charakteristiku, spustí ho pouze koncové zařízení.

Při vývoji programátoři běžně využívají takzvaný „virtuální stroj“. To znamená, že počítačem, kterým píší kód pro danou platformu, nasimulují operační systém dané platformy. Tento přístup umožní kód otestovat na počítači programátora, aniž by ho musel přenášet a spouštět na externím zařízení. Tyto virtuální stroje jsou často implementovány v takzvaných integrovaných vývojářských prostředích (angl. zkratka IDE – Integrated development environment) pro danou platformu. Například pro vývoj pro platformu Android je možné v IDE Android Studio využít funkci zvanou Android Virtual Device (česky Android virtuální zařízení), která umožnuje nasimulovat mobilní zařízení s operačním systémem Android.



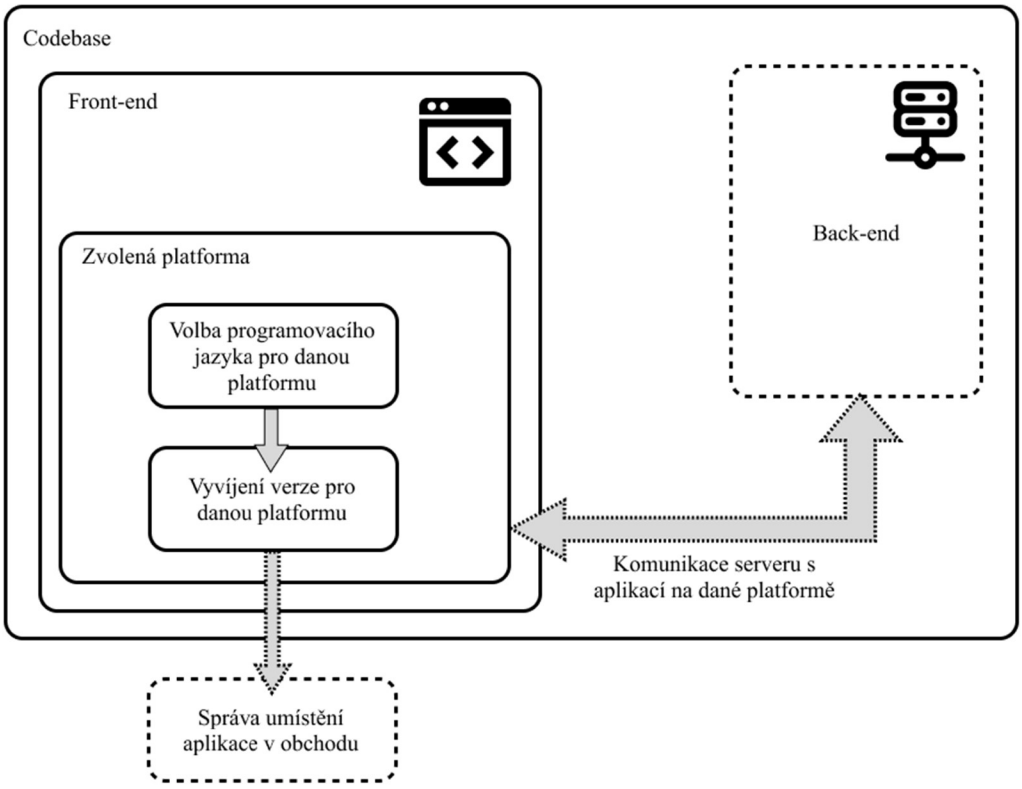
Obrázek 1: Ukázka AVD (zdroj: https://i.ytimg.com/vi/YKIgYMMHSLc/maxresdefault.jpg)

Pro každou platformu musí vývojáři programovat samostatnou verzi této aplikace. Všechny verze by v ideálním případě měly být sjednoceny, aby nedocházelo k obrání uživatelů jedné platformy o funkce, které jsou dostupné na jiných platformách. Tento požadavek může být mnohdy velmi náročné splnit, jelikož může docházet k problémům v hlediscích bezpečnosti, příliš velkého využití procesoru zařízení s daným operačním systémem, nebo příliš velké komplikovanosti zdrojového kódu.

Po ukončení iniciálního vývoje aplikace či přidání nových funkcí pro všechny platformy v nových verzích je možné se přesunout k publikaci do obchodů s aplikacemi na dané platformě, popsaných v kapitole 3.1.3. Aplikaci samozřejmě není nutné v obchodech publikovat, proto je v diagramu „Správa umístění aplikace v obchodu“ označena přerušovanou čarou, ovšem touto možností se tvůrce obírá o možnost dosažení všech uživatelů, kteří instalují aplikace pouze z těchto obchodů a nedůvěřují neznámým zdrojům.

Back-end je v diagramu také označen přerušovanou čarou, jelikož k existenci aplikace není v každém případě nutný. Pokud uživatel nepotřebuje jakákoliv data zpracovávat či z jakéhokoliv jiného důvodu nahrávat a přijímat přes internet na server, není existence back-endu nutná. Například pokud by byla vyvíjena aplikace pro luštění křížovek a všechny křížovky by byly součástí aplikace, přičemž by uživatel s dalšími verzemi získával i nové křížovky a zároveň by si mohl pro sebe vytvářet vlastní křížovky na aktuálně používaném

zařízení, existence serveru pro komunikaci s aplikací není nutná. Pokud by ale aplikace měla mít funkce, jako jsou založení účtu, vytváření křížovek nejen pro sebe, ale i pro ostatní uživatele aplikace, ukládání všech vyřešených nebo aktuálně řešených křížovek napříč všemi používanými zařízeními, musel by mít tvůrce server s databází, kam se všechna tato data budou ukládat, aby vše bylo možné. Aby mohl uživatel kdykoliv používat jiné zařízení, musí být umožněna synchronizace, díky které se veškerá data nahrají na server a následně se po změně zařízení tyto informace stáhnou do aktuálně používaného zařízení, kde může uživatel pokračovat v luštění.



Obrázek 2: Teoretický diagram (zdroj: vlastní tvorba)

3.2 Vývoj aplikace PWA přístupem

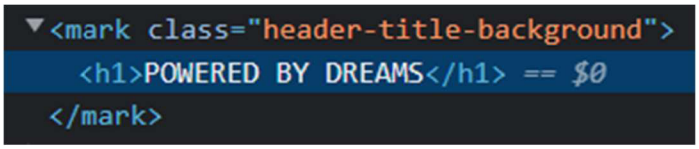
Mimo back-endu lze aplikaci vyvinout PWA technologiemi určenými primárně pro web, které se postupně navrství. Těmito technologiemi jsou značkovací jazyk *HTML*, který tvoří základ, stylovací jazyk *CSS* a programovací jazyk *JS*.

Značkovací jazyk *HTML* zobrazuje veškerý obsah, který se v aplikaci bude vykreslovat, ale také další důležité věci, které jsou probrány v následující kapitole. Další vrstvou je stylovací jazyk *CSS* neboli *kaskádové styly*, které slouží k popsání způsobu zobrazení obsahu v aplikaci. CSS umožňuje velmi široké spektrum možností, jak obsah zobrazit. Poskytuje řešení v případech, kdy je potřeba větší písmo, změna barvy obsahu, jeho velikosti atd. Poslední vrstvou potřebnou pro front-end aplikaci je programovací jazyk *JS*, zkratka pro Java Script. Jedná se o skriptovací jazyk, který umožňuje samotnou funkčnost takové aplikace. Bez jeho použití zůstává jen nastylovaný obsah, se kterým uživatel nemá možnost interagovat, což by ve zmíněném případě křížovkářské aplikace znamenalo, že uživatel uvidí veškerý text v aplikaci a pole křížovky, ale nebude ji moci luštit. JS umožňuje i další esenciální věci pro chod aplikace, například ukládání dat do lokálního úložiště nebo funkčnost aplikace off-line, vše popsáno v kapitole o JS (Michálek, M., 2017).

Obrázek 3: Vrstvení (zdroj: Michálek, M., 2017, Vzhůru do responzivního webdesignu, s. 56)

3.2.1 HTML – Hyper-text Markup Language

HTML je značkovací jazyk, kterým se definuje obsah na webových stránkách, potažmo v progresivních webových aplikacích. Obsah se vymezuje v takzvaných elementech, které se značí začínajícím a končícím tagem v hranatých závorkách.



Obrázek 4: Příklad obsahu definovanými tagy (zdroj: vlastní tvorba)

HTML dostane webový prohlížeč od serveru, prohlížeč potom vygeneruje ve svém okně jeho obsah, který je popřípadě upravený CSS souborem (Introduction to HTML, 2022). Pokud je třeba na webové doméně umístit na server HTML dokument, který se načte po navštívení této domény (domovská stránka), pojmenuje se index. Toto pojmenování zajistí, že se při navštívení domény takové aplikace uživatelem bez zadaného specifického souboru v URL adresáři prohlížeče zobrazí právě tento dokument – stejně, jako kdyby ho zadal absolutní cestou. Pokud by například adresa HTML souboru aplikace byla aplikace.cz/index.htm, k načtení tohoto souboru uživateli stačí zadat pouze aplikace.cz, jelikož webový server v kořenovém adresáři vrátí prohlížeči uživatele právě index.htm (Kyrnin, 2020).

V HTML dokumentu by měly být následující věci, a to i přesto, že prohlížeč dokáže zobrazit i neúplný dokument:

- <!DOCTYPE html> na začátku dokumentu značí, že se jedná o HTML dokument, jelikož prohlížeče mohou zobrazovat i jiné typy dokumentů;
- <html> je kořenový element HTML stránky;
- <head> je element obsahující základní informace o dokumentu, například jaké je jeho kódování, titulek, deskripce, autor a mnoho dalšího;
- <body> je element definující tělo dokumentum zde se nachází všechny viditelné věci, které se uživateli zobrazí, ale i jiné elementy (Introduction to HTML, 2022).

3.2.1.1 Vstupy

Pro jakékoliv vyplňování formulářů a ostatní zadávání informací na webové stránce je třeba použít vstupy, ty mají název input a existují různé typy – textové, číselné, vstup pro vyhledávání, nebo například vstup typu datetime-local. Po kliknutí na tento element má uživatel možnost zadat datum textově či aktivovat jeho výchozí výběr s časem prohlížeče, ty se vizuálně odlišují napříč prohlížeči a OS (HTML input type="datetime-local", n.d.).

3.2.1.2 Viewport

Pro jakoukoliv webovou aplikaci, která je vizuálně přístupná mobilním zařízením s různými rozměry obrazovky, je důležitý takzvaný meta tag *viewport*.

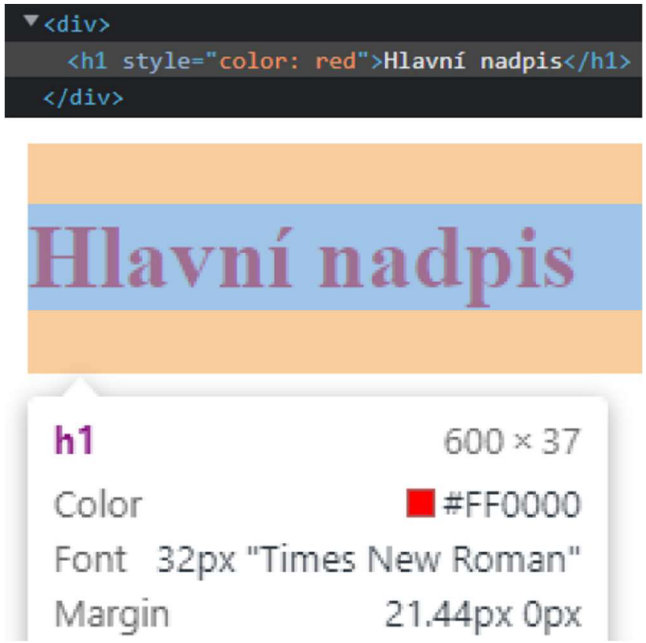
> *„Zjednodušeně (ale lidsky) řečeno slouží k informování prohlížeče o tom, zda a jak je web připraven pro mobilní zařízení“* (Michálek, 2017b, s. 102)

3.2.2 CSS

CSS je v angličtině zkratka pro Cascading Style Sheets, volně přeloženo do češtiny jako kaskádové styly. Je to speciální typ zápisu, který webovému prohlížeči definuje vzhled HTML dokumentu na obrazovce zařízení.

3.2.2.1 Aplikování stylů na dokument

HTML lze vizuálně nastylovat třemi způsoby. První – nejméně používaný – je *přímý zápis* v HTML. Nastavit například hlavnímu nadpisu červenou barvu je možné přímo v elementu nadpisu zápisem style=“požadovaná vlastnost“ v začínajícím tagu.



Obrázek 5: Přímý zápis CSS vlastnosti v elementu (zdroj: vlastní tvorba)

Tento způsob je vhodné použít pouze v případě, že jen jeden element ve webové aplikaci má specifický vzhled. Pokud by se vyskytovalo více elementů s tímto specifickým vzhledem, je vhodnější *nadefinovat v CSS souboru* třídu, kterou je následně nutné zapsat všem těmto elementům. Tím nabydou specifického vzhledu, čímž se sníží redundance kódu, a zároveň to poskytne možnost změnit kód pouze na jednom místě, i když se vzhled následně změní u všech specifikovaných elementů s danou třídou.

Dalším způsobem je zapsat vzhled HTML přímo v HTML souboru. Dá se tak učinit *zápisem elementu* <style> *v hlavičce dokumentu*, ve kterém je následně popsán vzhled daného dokumentu. Tento způsob je vhodný použít v případě, že se vzhled zapsaný v elementu <style> vztahuje pouze na daný dokument, nikoliv napříč více dokumenty, v takovém případě je – kvůli nižší redundanci – vhodnější využít možnosti připojení CSS souboru do hlavičky dokumentu, a tím nastylovat více dokumentů jedním souborem.

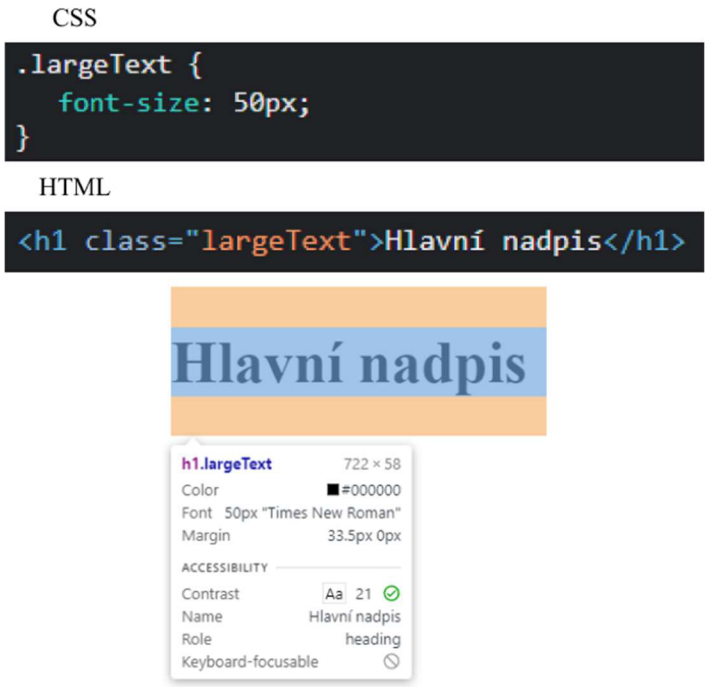
3.2.2.2 Možnosti CSS

V dnešní době umožňuje CSS (verze CSS3) vše od definování barev, odsazení textu, fontů, výšky textu, prostoru mezi obsahem zvaný margin a padding, vytvoření responzivního layoutu, až po pokročilé definování zobrazení stránky na různých zařízeních či složitou grafiku, která je nyní možná pouze pomocí CSS (přechody, průhlednost, kulatost rohů atd.),

tudíž není třeba tolik grafiky tvořit jako obrázky vyexportované z Photoshopu (Michálek, M., 2017, Vzhůru do CSS3).

3.2.2.3 CSS Třídy

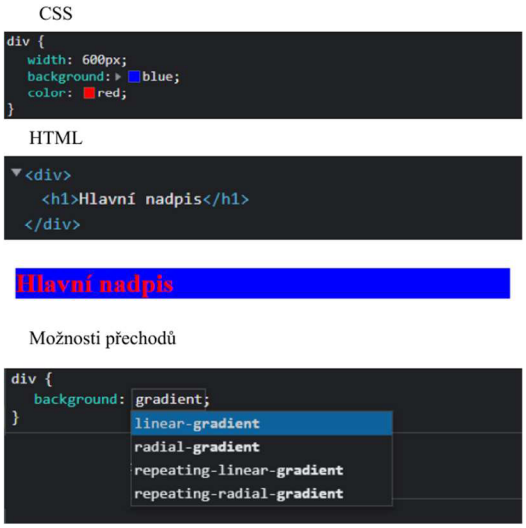
Třídy slouží k aplikování stylů na více elementů s homogenními či různými tagy. Daný element musí mít v HTML napsaný atribut class s určenou třídou. Je ovšem nutné nezapomínat, že různým tagům prohlížeč přidává automaticky předdefinované CSS vlastnosti. Pro odstranění rozdílů mezi prohlížeči automaticky přidanými CSS vlastnostmi lze připojit CSS kód, který tyto rozdíly odstraní (Gallagher, n.d.).



Obrázek 6: CSS třída (zdroj: vlastní tvorba)

3.2.2.4 Barvy v dokumentu

Barvy v celém dokumentu se dají nastavit třemi CSS vlastnostmi – color, background-color a background. Color slouží k nastavení barvy písma a background-color k barvě pozadí. Složitější barevné pozadí nastavuje vlastnost background, kde je dokonce možné nastavit přechody mezi barvami.



Obrázek 7: Barvy v dokumentu (zdroj: vlastní tvorba)

3.2.2.5 Font a vlastnosti textu

Fonty v dokumentu lze aplikovat vlastní, či využít některé z cloudových služeb. V případě vlastních fontů je třeba využít „zaváčového“ pravidla @font-face, a tím deklarovat název rodiny fontu a cestu k souboru (Michálek, M., 2017, Vzhůru do CSS3).

```
@font-face {
  font-family: _nazev_rodiny_;
  src: url(_cesta_k_souboru_s_pismem_)
    format(_format_souboru_);
}
```

Obrázek 8: @font-face pravidlo (zdroj: Michálek, M., 2017, Vzhůru do CSS3, s. 75)

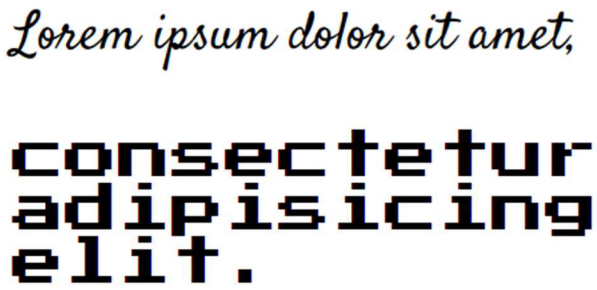
Při použití cloudových služeb (například Google Fonts) se nabízí mnohem jednodušší řešení – použití „zaváčového“ pravidla @import a v url uvedení adresy fontu. V obou případech je třeba v CSS definovat vlastnost font-family na daný HTML element, který je potřeba v daném fontu. Využít se poté dá více fontů v jednom dokumentu.

CSS

```css
@import url('https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Satisfy&display=swap');

.handWrittenText {
  font-family: 'Satisfy';
}
.digitalText {
  font-family: 'Press Start 2P';
}
```

HTML

```html
<p class="handWrittenText">Lorem ipsum dolor sit amet,</p>
<p class="digitalText">consectetur adipisicing elit.</p>
```



Obrázek 9: Použití Google Fonts (zdroj: vlastní tvorba)

Velikost textu se nastavuje pomocí font-size, elementem pro nadpisy je tag <h1> až <h6> a pro paragrafy tag <p>. Prohlížeč automaticky nastavuje velikost nadpisu od stanovené velikosti textu pro rodičovský element body (HTML). Nadpis <h1> je dvojnásobně vysoký až po nadpis <h6>, který je pouze 0.67krát vysoký (v případě prohlížeče Google Chrome).

Velikost řádků se deklaruje přes vlastnost line-height a lze definovat v pixelech a mnoha dalších jednotkách. Vzhledem k povaze textu se doporučuje nastavovat výška řádku dle výšky textu, a to buď zápisem v procentech či číslech. Výška řádku lze tedy zapsat například jako 150 % nebo 1.5.

Zarovnání textu označuje vlastnost text-align. Text lze zarovnat horizontálně na obě strany či střed, popřípadě do bloku zápisem text-align: justify.

Délku textu určuje tzv. inline-size, pro lepší responzivitu na menších zařízeních je vhodnější použít vlastnost max-inline-size, která nastaví maximální délku textu. Pokud je například na

chytrém telefonu šířka displeje 320 CSS pixelů (velikost telefonu S), text zůstane v maximální šíři displeje, přestože vlastnost max-inline-size je nastavena na víc.

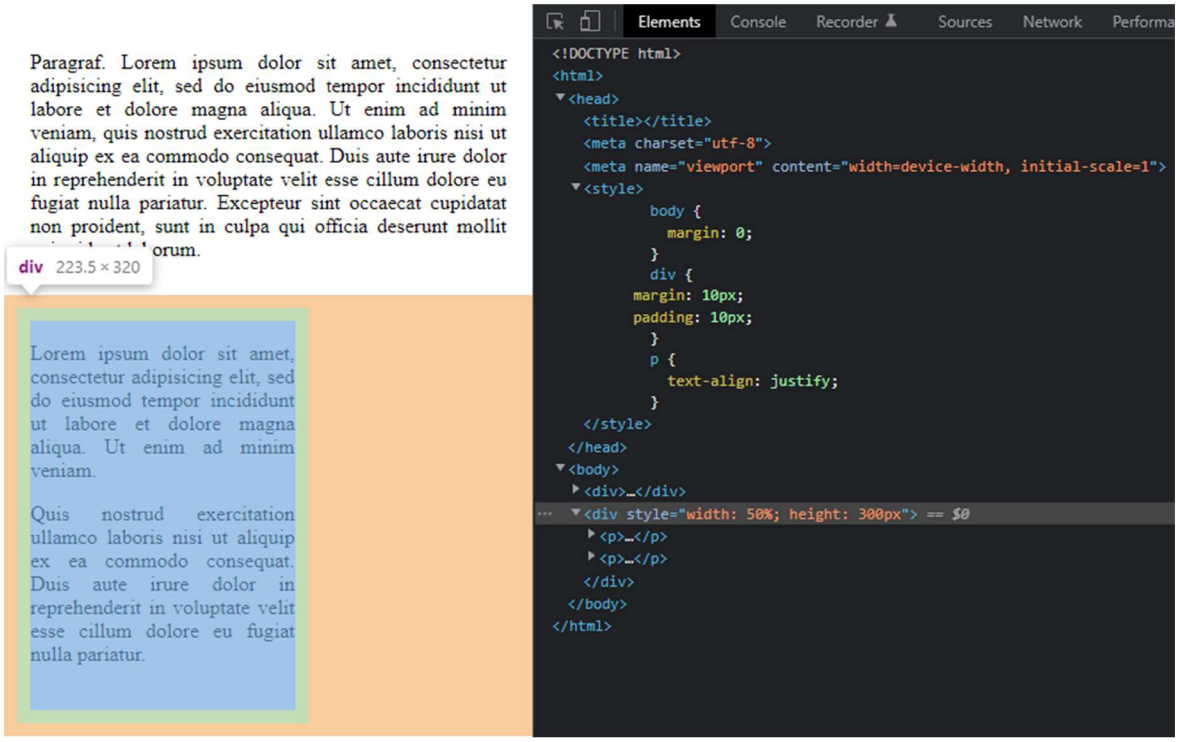
Obrázek 10: Vlastnosti textu – velikost zařízení o šířce tabletu (zdroj: vlastní tvorba)

3.2.2.6 Velikost obsahu a prázdný prostor

Velikost elementu se nastaví vlastnostmi width a height. Width nastaví element na požadovanou šířku, height na výšku. Velikost je možné nastavit opět několika způsoby, například v CSS pixelech nebo procenty. Při zápisu výšky či šířky v procentech vyobrazí prohlížeč element v procentuální velikosti rodičovského elementu, uvnitř kterého se daný element nachází.

Prostor mezi obsahem určují CSS vlastnosti margin a padding. Margin udává velikost prázdného prostoru z vnější strany elementu. Padding nastavuje prázdný prostor mezi okrajem samotného elementu (rodičem) a elementy uvnitř (dětmi).

Do velikosti elementu nastavené vlastnostmi width a height se nepromítá vlastnost padding, element se o deklarovanou velikost paddingu zvětší.



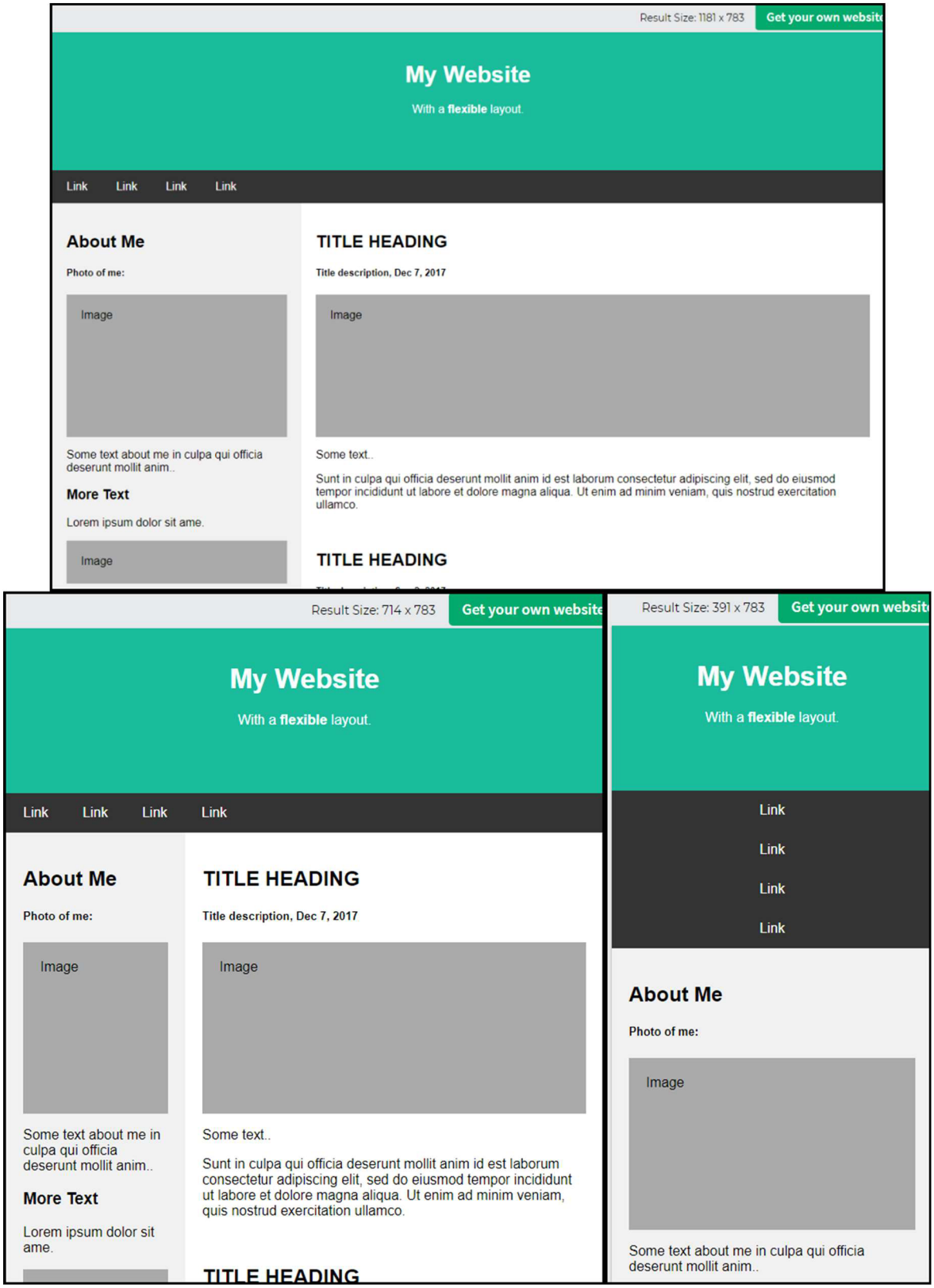
Obrázek 11: Velikost obsahu a prázdný prostor (zdroj: vlastní tvorba)

3.2.2.7 Responzivita

Základem responzivního HTML dokumentu je meta tag viewport, který je popsaný v kapitole 3.2.1.1. Informuje prohlížeč o tom, zda a jak je dokument responzivní. U větších displejů lze například využít šířku celého displeje vytvářením sloupců s obsahem, který je pevně stanoven, následně se počet sloupců mění dle šířky displeje, aby se každý sloupec vešel na stránku. Takové chování je možné díky CSS vlastnosti zvané flexbox. Ta umožňuje vytvářet rozvržení webové stránky, které se flexibilně mění k velikosti displeje zařízení.

> *„Flex v češtině znamená pružný, přizpůsobivý. Flexboxy jsou tedy pružné elementy layoutu. Jednou z hlavních předností flexboxu je totiž schopnost vyplňovat zbylý prostor bez nutnosti přepočítávání Javascriptem.“* (Michálek, M., 2017a, s. 132)

K flexboxu je možné specifikovat i takzvané *break-pointy*, ty umožňují měnit či přidávat CSS vlastnosti na základě překročení bodu velikosti viewportu – viditelné oblasti stránky na uživatelově zařízení. Tato velikost se může u zařízení měnit i během prohlížení stránky, a to když si uživatel například zmenší okno prohlížeče nebo progresivní webové aplikace se zobrazovaným obsahem.



Obrázek 12: Ukázka responzivního layoutu (zdroj: sestaveno z webu w3schools.com)

3.2.2.8 CSS proměnné (custom properties)

CSS proměnné jsou důležitou vlastností pro vizuální přizpůsobení aplikace uživatelem. Stejně jako v programovacích jazycích umožní CSS uložit proměnnou v různých částech dokumentu, která se následně může pomocí JavaScriptu měnit. V jiných případech ji lze využít k rychlejším úpravám CSS kódu, jelikož proměnná může být využitá na více místech a není nutno ji potom přepisovat. Například jednu barvu lze použít na pozadí tlačítek, barvu linků a na dalších místech; po přepsání globální proměnné se změny projeví na všech místech, kde je použita, celý kód to při správném použití vytváří organizovanější. Pro specifikování globální CSS proměnné je třeba deklarovat proměnnou pseudo-třídě `:root` (Using CSS custom properties (variables) - CSS Cascading Style Sheets | MDN, n.d.).

3.2.2.9 Usnadnění přístupu osobám se zrakovým postižením

Pro osoby se zhoršeným zrakem existuje esenciální nastavení webového prohlížeče na zvětšení obsahu webu. Pokud je CSS napsaný způsobem, že veškeré vlastnosti důležité pro čitelnost a velikost obsahu jsou deklarované v pružných jednotkách, konkrétně velikosti textu, `margin` a `padding` obsahu, prohlížeč tento obsah zvětší dle uživatelského nastavení prohlížeče.

> *„Pozor, nebavíme se o „zoomování“, ale zvětšení písma pro všechny weby. Taková věc existuje v prohlížečích nebo operačních systémech. A ano, lidé to používají. Asi jednou také budeme. Dělají to lidé s horším zrakem nebo třeba jen méně kvalitními displeji“* (Michálek, M., Vzhůru do (responzivního) webdesignu, str. 76).

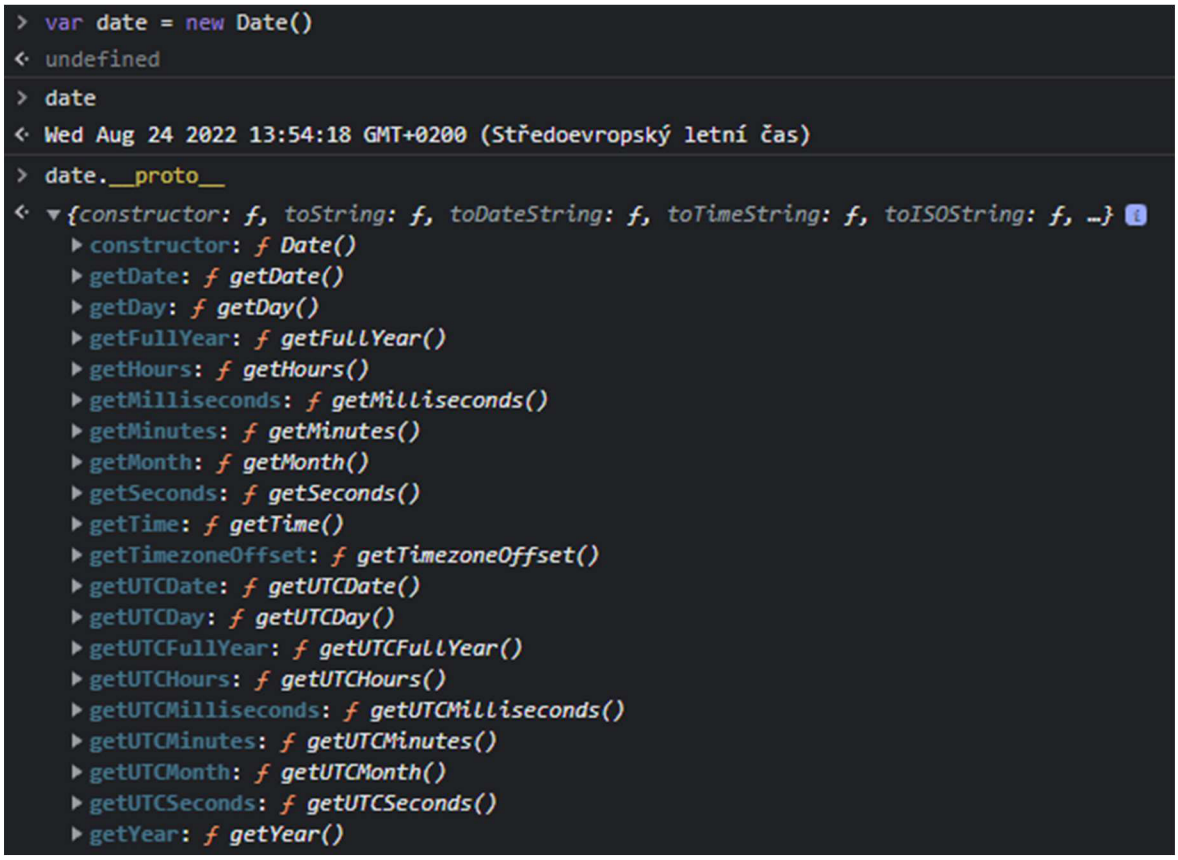
3.2.3 JavaScript

Javascript je *multi-paradigmatický*, *událostmi řízený* a *dynamicky typovaný* jazyk používaný zejména pro front-end vývoj webových stránek a aplikací.

3.2.3.1 Multi-paradigmatický jazyk

Javascript umožňuje využít celou škálu *paradigmat*, ta zahrnuje *objektově orientované*, *funkcionální*, *procedurální* a *prototypově orientované* programování (del Vale, 2020).

Objektově orientované paradigma umožňuje vytvářet třídy, které slouží jako šablony pro vytváření objektů. Při vývoji v JS se dá využít mnoho předdefinovaných tříd, například třídu Date, která umožňuje vytvořit objekt s datem a časem se specifickým datovým pásmem. Třída Date má specifikované i metody, které u objektu zjistí například název dne či měsíce, číselné vyjádření roku, měsíce, dne, hodin, minut, sekund a milisekund.



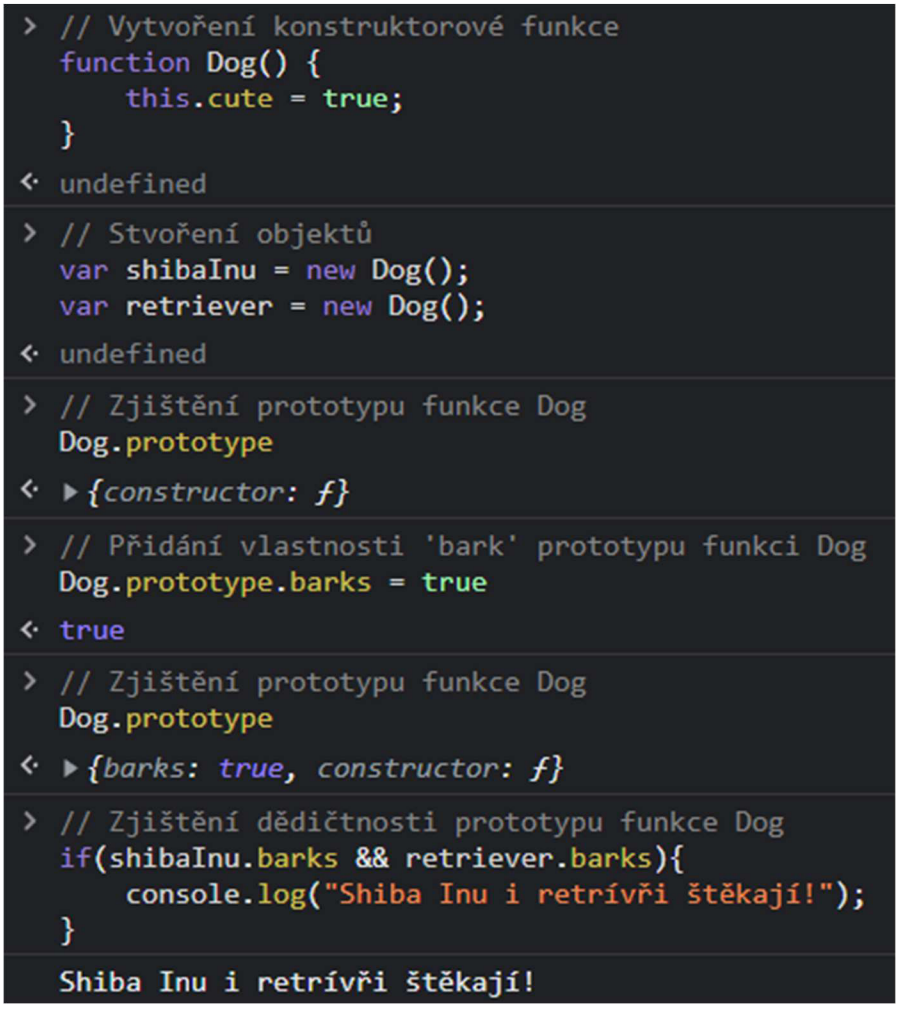
Obrázek 13: Příklad vytvoření objektu s aktuálním datumem a časem, ukázka metod třídy Date (zdroj: vlastní tvorba)

Funkcionální programování vyžaduje psaní výrazů (v matematickém smyslu) a tzv. „čistých“ funkcí, tedy funkcí, které při identických vstupech vrací vždy stejný výstup. Výrazy a funkce by neměly ve funkcionálním programování ovlivnit stav a data, tento přístup se zaměřuje na psaní kódu, který je snáze čitelný a odolný vůči chybám (Aryan, n.d.) (Jak se naučit programovat: Úvod do funkcionálního programování, n.d.).

JS poskytuje řadu čistých funkcí (metod) využitelných v programu, které pro každý vstup mají fixní výstup, zároveň neovlivňují okolí. Například pole lze filtrovat pomocí funkce filter, která přijímá jako parametr podmínku pro zachování prvků v poli. Spojit dvě pole lze pomocí funkce concat, která přiřadí ke stávajícímu poli vložený parametr (pole) (Aryan, n.d.).

Procedurální paradigma v JS poskytuje možnost psát algoritmy, které se ve zdrojovém kódu provedou v přesném pořadí za sebou. Kód lze zpřehlednit, zkrátit a zrychlit využitím funkcí (procedur neboli podprogramů), které lze znovu použít kdekoliv v kódu (18progrjazyky, n.d.).

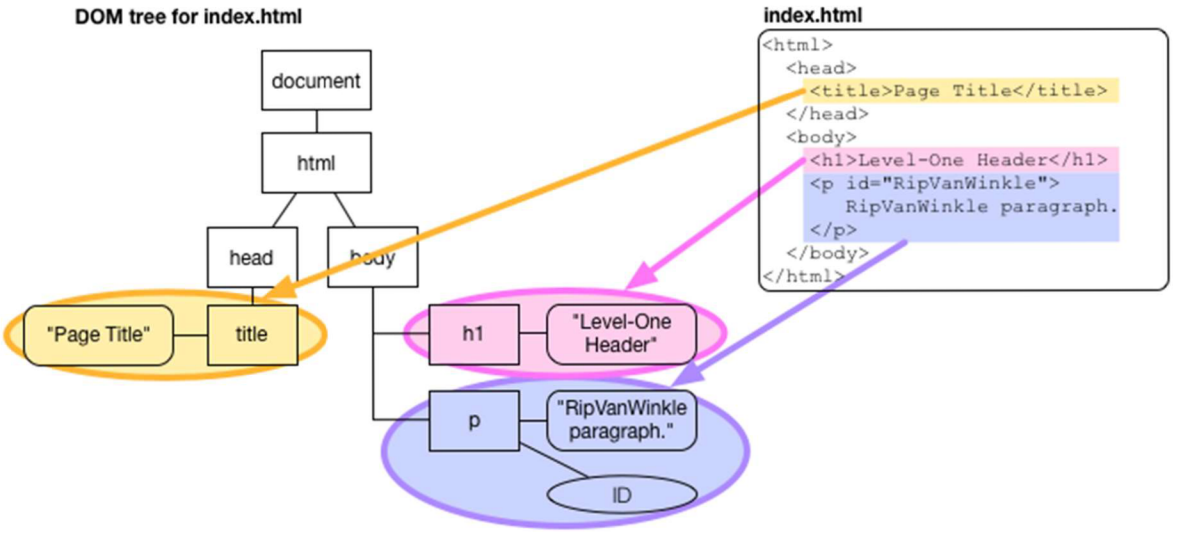
V JS má každá funkce a objekt vlastnost zvanou prototyp. *Prototypově orientované paradigma* v JS umožní za běhu programu libovolně měnit či přidávat vlastnosti a metody funkce konstruktor pomocí zmíněné výchozí vlastnosti prototype, objekty následně dědí tyto vlastnosti a metody (JavaScript Prototype (with Examples), n.d.).



Obrázek 14: Ukázka využití prototypově orientovaného paradigmatu v JS (zdroj: vlastní tvorba)

3.2.3.2 Událostmi řízený jazyk

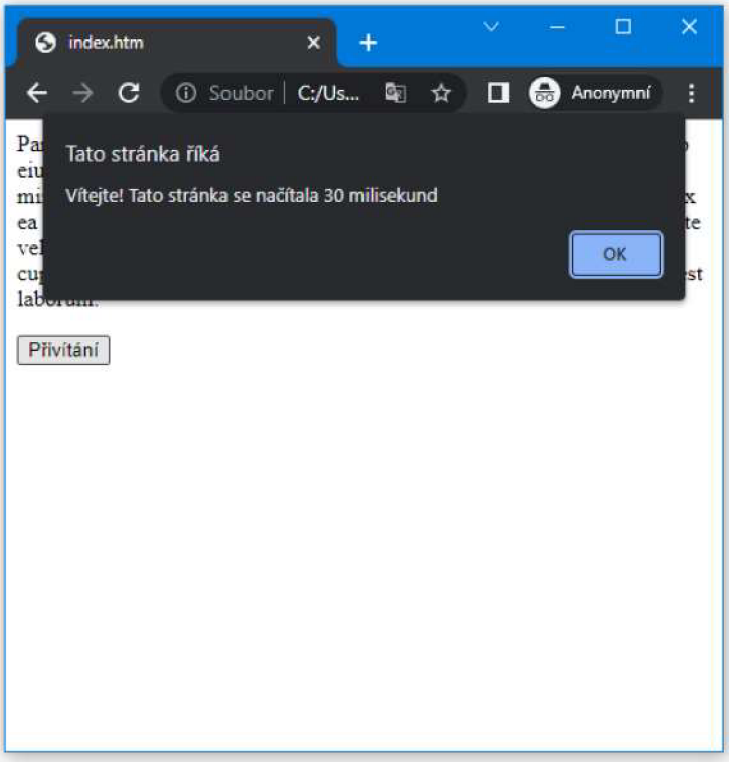
Při programování webových aplikací je jasné, že uživatel bude komunikovat s aplikací pomocí událostí, které spustí například při kliknutí na tlačítko, zadání textu do pole, pohybu myší atd. Pro interakci uživatele s aplikací lze využít takzvané naslouchače událostí, které umožňují takové interakce zaznamenávat. Jeden element může mít i více naslouchačů. JS umožňuje zaznamenávat různé typy událostí probíhajících v prohlížeči při zobrazování dokumentu (například načtení dokumentu) pomocí tzv. „*DOM on-event handler*“, handler může být ovšem pouze jeden na takovou událost. *DOM* je zkratka pro „document object model“ neboli objektový model HTML dokumentu. Tento model funguje na principu hlavního uzlu neboli kořenu, který představuje samotný dokument, a ten má pod sebou ve stromové struktuře vnořeny všechny další elementy dokumentu. Každý uzel má své atributy zobrazeny v takové struktuře. JS umožňuje také vytváření vlastních specifických událostí.



Obrázek 15: Vizualizace DOM (zdroj: https://i.stack.imgur.com/IERlg.png)

HTML + JS

```
index.htm
<div>
 <p>Paragraf. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
</div>
<button id="welcome-button">Přivítání</button>
<script type="text/javascript">
  var button = document.querySelector("#welcome-button");
  var domLoaded;

  window.onload = function () {
    loadTime = window.performance.timing.domContentLoadedEventEnd -
               window.performance.timing.navigationStart;
  }

  button.addEventListener("click", function () {
    alert("Vítejte! Tato stránka se načítala" + loadTime + " milisekund");
  })
</script>
```



Obrázek 16: Ukázka "DOM on-event handler" a naslouchače událostí (zdroj: vlastní tvorba)

3.2.3.3 Reprezentace dat pomocí DOM

Pokud má webová aplikace pomocí událostí od uživatele získávat a zpracovávat data, je zřejmé, že je musí také vizuálně pro uživatele interpretovat. V případě webových aplikací tak učiní v HTML struktuře dokumentu, který interpret JS může upravovat přes DOM, stejně jako přes něj může naslouchat událostem. Problematika zobrazení dat v DOM spočívá v synchronizaci modelu DOM a datové struktury aplikace, lze tento problém vyřešit pomocí vlastního atributu data-*.

> *„Pokud budeme potřebovat uložit k elementu pro něj specifická data, můžete to provést prostřednictvím atributu data-*. To je výhoda oproti ukládání takových dat do pole/objektu, protože bychom museli synchronizovat elementy modelu DOM s tímto polem/objektem, kdybychom chtěli načítat tato data. Místo toho vyhledáme element modelu DOM a přímo z něho načteme hodnotu jeho atributu data-*“* (A. Pehlivanian, D. Nguyen, 2014, JavaScript okamžitě, s. 112)

3.2.3.4 Dynamicky typovaný jazyk

Javascript je jazyk takzvaně dynamicky typovaný. To znamená, že interpret přiřadí proměnným jejich typ (logickou hodnotu, objekt, číslo, textový řetězec) po spuštění programu na základě jejich hodnoty. Typ proměnné se tedy nedeklaruje v kódu, stejnou vlastnost sdílí i například programovací jazyk Python.

3.2.4 Přeměna webové aplikace na progresivní webovou aplikaci

3.2.4.1 Zabezpečený „hyper text transfer protocol“

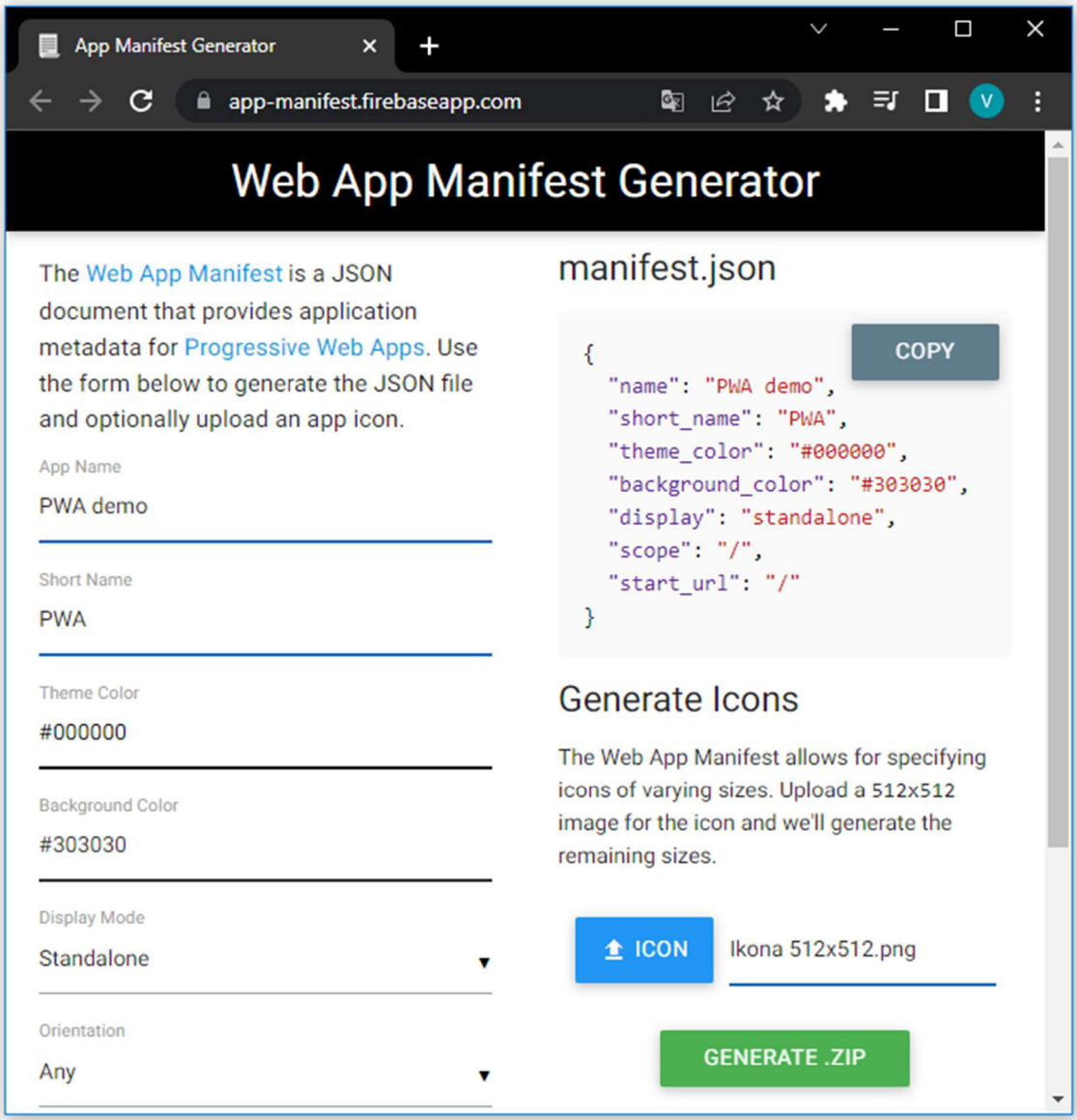
„Service worker“ je zcela esenciální technologie při tvorbě PWA, k jeho fungování je potřeba server se zabezpečeným protokolem, například *SSL* – Secure Sockets Layer. Jedná se o protektivní vrstvu protokolu, která zajišťuje zabezpečené internetové připojení a enkrypcí chrání veškerá data, která jsou posílána mezi dvěma systémy. S nezabezpečeným HTTP protokolem nemůže být webová aplikace PWA vyvinuta (P. Karwatka, A. Kwiecień, F. Rakowski, K. Grzybowska, 2019) (What is SSL, TLS and HTTPS?, n.d.).

Hosting se zabezpečeným HTTP protokolem poskytuje například webová platforma *Netlify*, na které se dá zdarma zveřejnit libovolné množství stránek domény druhého řádu

s omezenou datovou kapacitou. To znamená, že doména bude vypadat například takto: „nazev-stranky.netlify.app“ a nebude smět překročit limit 100 GB (dále ještě další omezení, například z hlediska objemu návštěvnosti) na všechny zveřejněné projekty. Na Netlify se dá zaregistrovat pomocí účtu *Github*. Po přihlášení si uživatel může zvolit, jaký repositář obsahující zdrojový kód ze svého Github účtu chce publikovat, a vybere si doménu. Následně si může zvolit automatické publikování všech změn v tomto repositáři. To znamená, že každá změna ve zdrojovém kódu v repositáři bude publikována. Netlify poskytuje možnost vidět historii změn a staré verze kódu obnovit do běžící aplikace.

3.2.4.2 Manifest webové aplikace

Manifest webové aplikace je soubor *JSON* – JavaScript Object Notation, který přikazuje prohlížeči, jak PWA zobrazit. Obsahuje veškerá důležitá metadata typu název aplikace, jazyk, informace o ikonách aplikace (název, velikost a zdroj), začínající *URL* – Uniform Resource Locator, barva tématu a další (P. Karwatka, A. Kwiecień, F. Rakowski, K. Grzybowska, 2019).



Obrázek 17: Nástroj pro vytvoření manifestu (zdroj: https://app-manifest.firebaseapp.com/)

3.2.4.3 Service worker

Service worker je JS soubor běžící na pozadí ve webovém prohlížeči uživatele a přináší funkce PWA jako jsou:

- Notifikace.
- Synchronizace na pozadí.
- „Asset caching“ zodpovědný za fungování PWA off-line, stahuje do zařízení uživatele důležitý kód potřebný k běhu aplikace (HTML, CSS, JS soubory) (P. Karwatka, A. Kwiecień, F. Rakowski, K. Grzybowska, 2019).

3.2.5 Testování PWA

Testování webových aplikacích obecně jde provést mnoha různými způsoby. Jedním z nejsložitějších způsobů je mít k dispozici jednotlivá zařízení, na kterých aplikaci testovat, každé zařízení pak má i nainstalované testované prohlížeče (pokud to prohlížeč umí, lze otestovat i proces nainstalování a zobrazení standalone PWA). Tento velmi nákladný způsob emuluje webová aplikace *Browserstack*, která má flotilu zařízení určených na testování, mobilních zařízení různých značek i desktopy. Tyto fyzická zařízení následně sdílí svou obrazovku a tester toto zařízení ovládá.

3.2.5.1 Audit

Audit PWA lze provést pomocí open-source aplikace pro diagnostiku webových aplikací *Lighthouse*. Aplikace po dokončení auditu spuštěného se zvolenými kritérii vrátí hodnocení kvality webové stránky či aplikace na základě zvolených kritérií jako je rychlost, přístupnost, optimalizace pro vyhledávače nebo například nejlepší praktiky. Umí také validovat aspekty PWA. Hodnocení testované webové stránky je vyčteno a popsáno u každého kritéria, první se zobrazují nesplněné požadavky jednotlivých kritérií.

4 Vlastní práce

Ve vlastní práci se nachází popis vývoje aplikace sloužící jako osobní asistent. Jedná se o off-line a čistě front-end aplikaci, která umožňuje uživateli přidávat si úkoly s prioritou a časem dokončení, a to formou přidání data termínu dokončení (tzv. „dead-line“) či přidáním časových rámců od momentů přidání úkolů (minuty až měsíce). Nedokončené úkoly se řadí nejdříve sestupně dle priority (od úkolu označeným prioritou „nejvyšší“ přes „střední“ po „nejnižší“) a následně dle času (od nejbližšího dead-linu po nejzazší). Dokončené úkoly jsou řazeny až za nedokončenými. Pokud se uživatel nechtěně překlikne, nebo bude mít jakýkoliv jiný důvod úkol nesprávně označit za dokončený, bude mu nabídnuta možnost vrátit úkol zpět stejným tlačítkem mezi probíhající. Až po smazání úkolu již nebude možné ho vrátit.

Další funkcí je psaní krátkých poznámek, také ohodnocené prioritou a dle ní seřazené, které lze následně smazat.

Uživatel má možnost nastavit si vzhled aplikace, ve výchozím nastavení se aplikuje tmavý či světlý režim dle nastavení prohlížeče uživatele, ale zůstává možnost si režim změnit. Poslední funkcí je možnost nastavit si barevný vzhled aplikace, jímž bude barevný přechod pozadí úvodní sekce a vybraných ikon priorit úkolů (střední priorita).

Celá aplikace je tzv. „single-page“, veškerý obsah je tedy v jednom HTML dokumentu (stránce). Pro navigaci po aplikaci jsou použité speciální HTML bloky, které se zobrazí či schovají v okně aplikace nebo webového prohlížeče. Celá aplikace se tedy načte pouze jednou a při pohybu v ní nebude docházet k přesměrovávání a načítání další stránky.

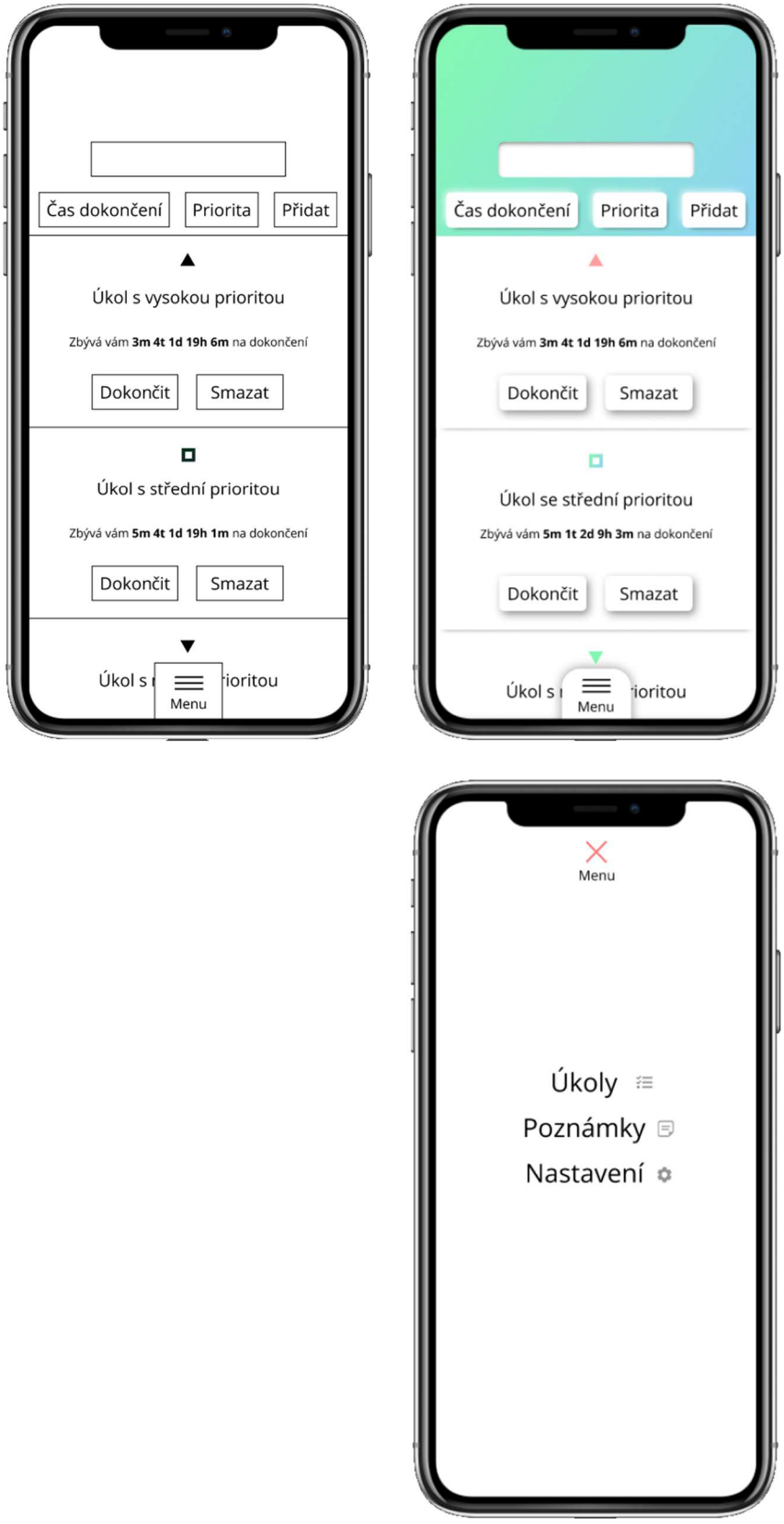
Data aplikace se ukládají přes lokální úložiště prohlížeče na zařízení uživatele (obvykle 5 MB).

4.1 Návrh aplikace

Návrh aplikace byl vyvíjen tzv. *Mobile First* filozofií, to znamená, že při návrhu rozhraní se zaměřuje nejdříve na telefony. Michálek (2017) uvádí, že kromě přizpůsobení se popularitě používání smartphonů k prohlížení webu, nutí tato filozofie navrhovat aplikace jednodušší a dle jeho domnění celkově lepší.

Návrh byl vytvořen nejdříve jako drátový model s ikonami pro velikost displeje smartphonu iPhone X, následně se aplikoval vizuální design prototypu aplikace – konkrétně „stránky" (ve smyslu vizuálním) s úkoly a viditelným menu na navigaci po aplikaci. Zbylé dvě funkce (poznámky a nastavení) nepotřebují jakýkoliv designový návrh, jelikož prvky jako tlačítka, texty, stíny, rozvržení jednotlivých elementů atp. se nastylují stejným způsobem jako stránka s úkoly.

Design aplikace byl vyvíjen tak, aby byl jednoduchý, přehledný a hlavně intuitivní. V úvodní sekci na každé „stránce", kde uživatel zadává své poznámky či úkoly, je barevný diagonální přechod dvou pastelových barev (zelená a modrá). Jak je uvedeno výše, uživatel si následně tyto barvy může libovolně měnit. Stejný barevný přechod má i výplň piktogramu znázorňujícího střední prioritu. Vysoká a nízká priorita má také pastelové barvy, konkrétně růžovou a světle zelenou.



Obrázek 18: Grafický návrh aplikace (zdroj: vlastní tvorba)

4.2 HTML

Aplikace má pouze jeden HTML dokument zvaný index.htm. Takové pojmenování zajistí, že se při navštívení domény této aplikace bez zadaného specifického souboru v URL adresáři prohlížeče zobrazí v prohlížeči právě tento dokument, jak je popsáno v kapitole 3.2.1.

V dokumentu jsou uvedeny všechny věci, které by měl HTML dokument obsahovat, viz kapitola 3.2.1:

- <!DOCTYPE html> deklaruje, že se jedná o HTML dokument;
- <html> je kořenový element, atribut lang značí prohlížeči jazyk dokumentu a dir směr písma;
- <head> obsahuje veškeré důležité elementy, které obsahují meta-informace (elementy meta) o dokumentu a další důležité informace. Elementy meta mají vždy jeden atribut, který předává prohlížeči obsah sdělení celého elementu. První meta element obsahuje atribut charset, který uvádí kódování dokumentu, konkrétně univerzální formát UTF-8 (jeden znak má 8 bajtů) napříč všemi jazyky podporující emoji a další speciální znaky. Další element je viewport, popsaný v kapitole 3.2.1.1. Nadcházející dva elementy slouží jako zdroj pro aplikování fontů a skript funkcí, které jsou použity k práci s časem ve zdrojovém kódu samotné aplikace. Zbylé elementy link aplikují vytvořené kaskádové styly aplikace, ikonu a manifest popsaný v kapitole 3.2.4.2. Element title slouží k zobrazení titulku dokumentu zobrazovaného v kartě prohlížeče spolu s ikonou. Poslední potomek hlavičky obsahuje skript k aplikování tmavého či světlého režimu aplikace dle nastavení prohlížeče (pokud si uživatel následně nenastaví jinak) a uživatelem nastaveného barevného přechodu. Tento potomek se nachází v hlavičce z důvodu, že je před tělem (vizuální částí v prohlížeči) dokumentu. Tím pádem se preferované nastavení vzhledu aplikace nastaví ještě předtím, než se načtou vizuální elementy dokumentu. Takto nemůže dojít k aplikování výchozích stylů ještě před jejich zobrazením, což by mohlo způsobit rychlé probliknutí, jelikož by skript správné styly aplikoval až po vykreslení vizuálních elementů. Obsah tohoto skriptu je popsán v kapitole 4.6;
- <body> obsahuje 5 elementů div. První je menu s „odkazy“ na „stránky“ aplikace (nedochází k přesměrování na jiný dokument, ale pouze k zviditelnění dané „stránky“ a k zneviditelnění ostatních „stránek“), zbylé elementy se třídou page

slouží jako zmiňované „stránky“. Poslední element je skript obsahující zdrojový kód aplikace. Je umístěn jako poslední, jelikož obsahuje proměnné, které se odkazují na předchozí elementy v dokumentu. Pokud by byl umístěn dříve, mohlo by dojít k chybě způsobené uložením nenačtených elementů do proměnných.

```
···<!DOCTYPE html> == $0
  <html lang="cs" dir="ltr">
  ▼<head>
     <meta charset="utf-8">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <link href="https://fonts.googleapis.com/css2?family=Open+Sans&display=swap" rel="stylesheet">
     <script src="https://momentjs.com/downloads/moment.min.js"></script>
     <link rel="stylesheet" href="style.css">
     <link rel="icon" type="image/png" sizes="145x145" href="icon_145.png">
     <link rel="manifest" href="manifest.json">
     <title>PWA demo</title>
    ▶<script type="text/javascript">…</script>
   </head>
  ▼<body>
    ▶<div class="main-menu">…</div>
    ▶<div class="page" id="tasks">…</div>
    ▶<div class="page" id="notes">…</div>
    ▶<div class="page" id="settings">…</div>
     <script type="text/javascript" src="app.js"></script>
   </body>
  </html>
```

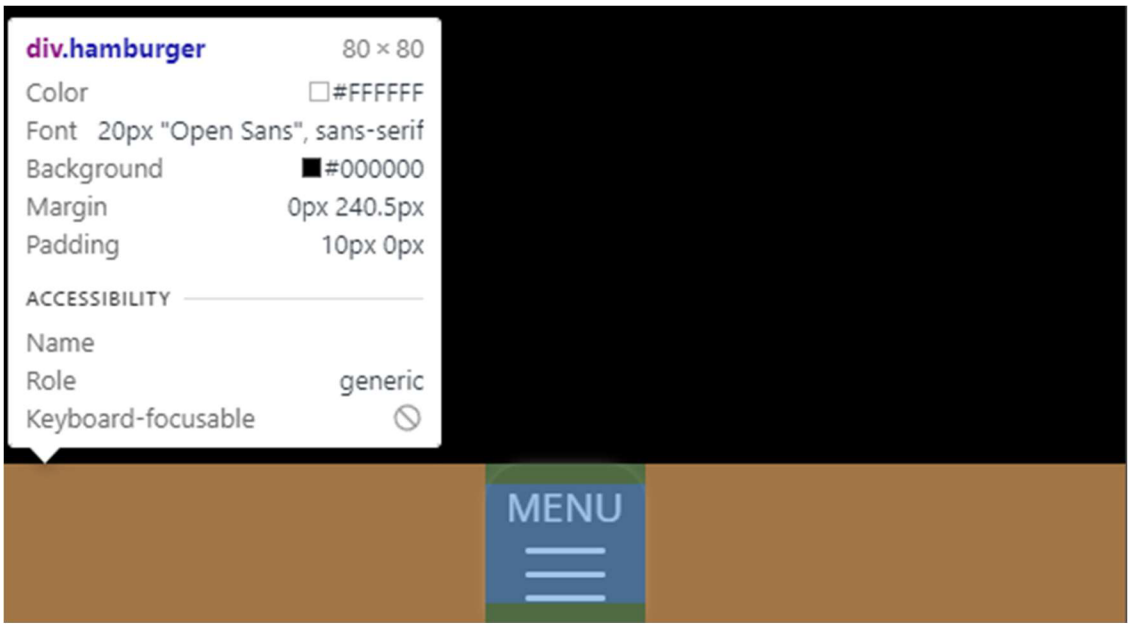
Obrázek 19: HTML soubor s elementy hlavičky a těla dokumentu (zdroj: vlastní tvorba)

4.2.1 Navigace aplikace

Element zobrazující celou navigaci má třídu main-menu, uvnitř tohoto elementu se nachází div se třídou hamburger (webdesignérský žargon, označení pro piktogram navigace). Tento element obsahuje název MENU pro indikaci, že po rozkliknutí budou vidět položky tohoto menu. Dále obsahuje tři elementy div, které tvoří piktogram tří čar, znázorňující ikonu navigace. Tento piktogram je vytvořen pomocí HTML a CSS kvůli snadnosti implementace animace po kliknutí, kdy element s třídou 2 zmizí a zbylé elementy line se přetransformují v červený křížek indikující zavření navigace po kliknutí. Element třídy options uvnitř navigace obsahuje všechny „odkazy“ na „stránky“ spolu s jejich ikonou. Ikony položek navigace jsou v kódování BASE64, který umožňuje digitální soubory (v tomto případě malé PNG obrázky) zakódovat do HTML souboru jako text v atributu zdroje obrázku.

```
<div class="main-menu"> == $0
  <div class="hamburger">
    <p>MENU</p>
    <div class="line 1"></div>
    <div class="line 2"></div>
    <div class="line 3"></div>
  </div>
  <div class="options">  flex
    <div class="item">  flex
      <p onclick="selectPage('tasks')">Úkoly</p>
      <img src="…AcgVeobDa0QAAAABJRU5ErkJggg==">
    </div>
    <div class="item">…</div>  flex
    <div class="item">…</div>  flex
  </div>
</div>
```

Obrázek 20: HTML kód navigace aplikace (zdroj: vlastní tvorba)

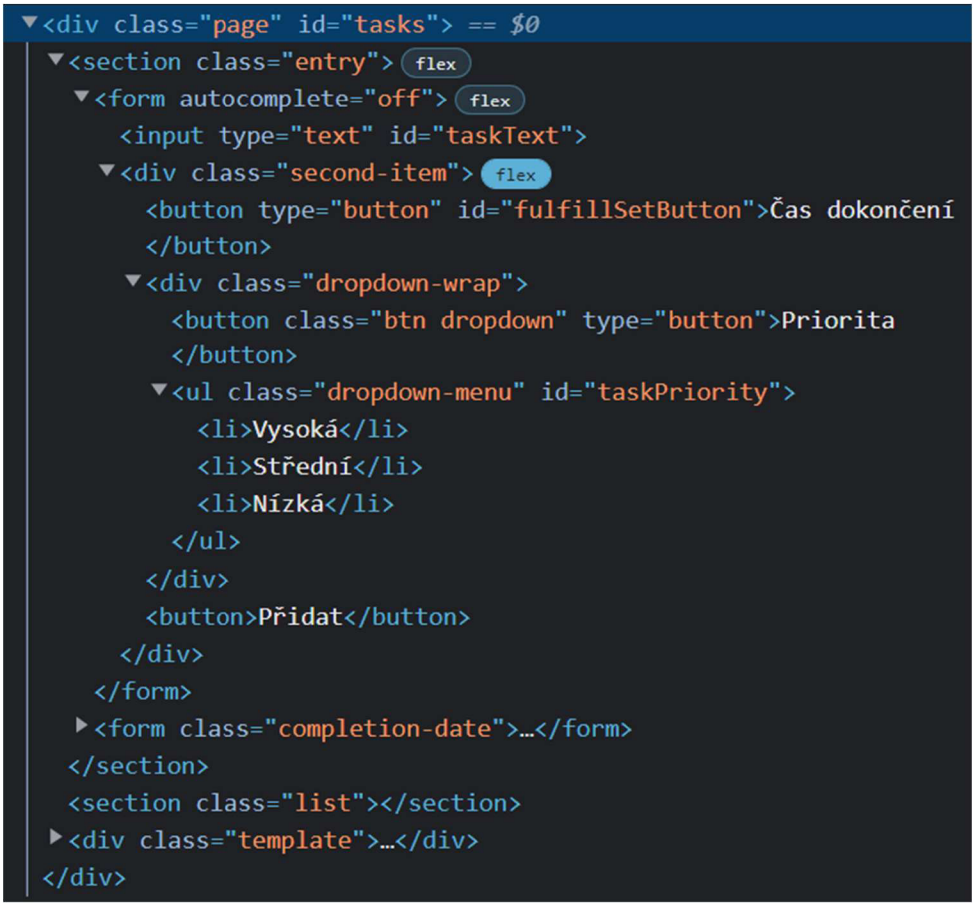


Obrázek 21: Zobrazení elementu třídy hamburger v dev tools (zdroj: vlastní tvorba)

4.2.2 Sekce s úkoly

Každá sekce má atribut id, který slouží k jejímu identifikování a je esenciální k následné interakci uživatele přes JavaScript. První potomek je sekce pro zadání požadovaného úkolu, proto nese atribut s třídou entry, uvnitř je formulář – element form pro zadání veškerých informací o úkolu s výjimkou času dokončení, k němuž slouží formulář další. Tento formulář má atribut autocomplete s hodnotou off. To dle webového článku HTML input autocomplete Attribute zajistí, že prohlížeč nebude uživateli nabízet možnost vyplnění formuláře, která není v tomto případě pro uživatele žádoucí. Uvnitř formuláře se nachází element input umožňující uživateli zadat text úkolu. Aby byl input pro vkládání textu, musí mít atribut type s hodnotou text, tím prohlížeč ví, že se jedná o vstup prostého textu. Druhý potomek je element div. Ten „zaobalí“ zbývající tři prvky tohoto formuláře, aby se při vykreslení na menších obrazovkách zařadily pod textový vstup společně. Uvnitř se nachází element button sloužící jako tlačítko k otevření formuláře pro zadání datumu dokončení úkolu a další div, jenž obsahuje tlačítko priority, které po kliknutí zobrazí element ul (unordered list) se seznamem priorit. Jednotlivé položky seznamu jsou elementy li (list item). Jako poslední element formuláře je tlačítko pro přidání úkolu. Za formulářem se nachází již zmiňovaný formulář k zadání termínu dokončení úkolu.

Druhá sekce má třídu list, jelikož je určena pro zobrazování seznamu vytvořených úkolů. Za ní je element se třídou template, ten obsahuje HTML šablonu pro úkol.



Obrázek 22: Sekce s úkoly (zdroj: vlastní tvorba)

4.2.2.1 Formulář pro zadání termínu úkolu

Aplikace má dvě možnosti zadání termínu k dokončení úkolu. Tou první je přidání časových rámců od měsíce až po minuty, a druhou zadání datumu, kdy má být úkol dokončen. Obě tyto možnosti jsou zobrazeny ve formuláři, přepíná se mezi nimi tlačítkem s atributem sateDateInp. Ve výchozím nastavení je zobrazena možnost přidání pomocí časových rámců, proto má tlačítko text „Datum“.

První možnost je reprezentována elementem div s id completion-date_time. Nadpis h2 oznamuje, že následující vstupy umožňují přidávat počet časových rámců, po jejichž uplynutí má být úkol dokončen. Každá možnost má dvě tlačítka button s textem „+“ a „-“ (přidání a odebrání počtu rámců) a element input typu number, který zajistí, že na mobilních zařízeních bude virtuální klávesnice pro zadávání časového rámce pouze číselná. Předdefinovaná hodnota elementu input (atribut value) je nula, minimální hodnota je také

nastavena na nulu a maximální hodnota je rovna maximálnímu počtu časových rámců v jednom vyšším časovém rámci, například počet měsíců má maximální hodnotu 12, jelikož rok má dvanáct měsíců atp. Jako posledním potomkem časového rámce je paragraf p s názvem časového rámce.

Druhá možnost je reprezentována elementem div s id completion-date_date. Nadpis h2 indikuje, že úkol by měl být dokončen nejpozději v zadaném datu. Následuje div se vstupem typu datetime-local, který je popsaný v kapitole 3.2.1.1.

```
<form class="completion-date"> == $0
  <div id="completion-date_time"> flex
    <h2>Dokončit za:</h2>
    <div> flex
      <button onclick="increaseValue(months)">+</button>
      <input id="months" type="number" value="0" min="0" max="12">
      <button onclick="decreaseValue(months)">-</button>
      <p>měsíců</p>
    </div>
    <div>…</div> flex
    <div>…</div> flex
    <div>…</div> flex
    <div>…</div> flex
  </div>
  <div id="completion-date_date" style="display:none; margin: 1rem 0">
    <h2>Dokončit v:</h2>
    <div> flex
      <input id="set-datetime-inp" type="datetime-local"> flex
    </div>
  </div>
  <button id="setDateInp">Datum</button>
  <button>Nastavit</button>
</form>
</section>
```

Obrázek 23: Struktura formuláře pro zadání času dokončení úkolu (zdroj: vlastní tvorba)

4.2.2.2 Šablona úkolu

Element div se třídou template slouží k poskytnutí zdrojového kódu JS interpretovi v DOM pro tvorbu nových úkolů i poznámek. Uvnitř se nachází div třídy task, který kromě třídy obsahuje i speciálně vytvořený atribut data-index, jenž uchovává index neboli ukazatel úkolu potřebný k synchronizaci objektů datové struktury programu a reprezentací v DOM, jak je popsáno v kapitole 3.2.3.3.

Uvnitř elementu třídy task se nachází element pro zobrazení priority úkolu, složený element třídy text-items k zobrazení textových položek úkolu a poslední element obsahující tlačítka pro dokončení a odstranění úkolu.

```
<div class="template">
  <div data-index="" class="task">
    <div class="priority"></div>
    <div class="text-items">
      <p class="text"></p>
      <p class="time">Zbývá vám <span></span> na dokončení</p>
    </div>
    <div class="button-list">
      <button type="button" class="fulfill">Dokončit</button>
      <button type="button" class="delete">Smazat</button>
    </div>
  </div>
</div>
```

Obrázek 24: HTML šablona úkolu (zdroj: vlastní tvorba)

4.2.3 Sekce s poznámkami

Sekce s poznámkami obsahuje stejné prvky, které jsou popsány v předchozí kapitole, ale v signifikantně menší míře. Chybí zde veškeré formuláře potřebné pro práci s datumy a některé textové elementy v šabloně úkolu. Také jsou přejmenovány všechny atributy id pro správnou identifikaci v DOM. Element v šabloně má změněnou třídu na note.

```
<div class="page" id="notes"> == $0
  <section class="entry"> flex
    <form autocomplete="off"> flex
      <input type="text" id="noteText">
      <div class="second-item"> flex
        <div class="dropdown-wrap">
          <button class="btn dropdown" type="button">Priorita</button>
          <ul class="dropdown-menu" id="notePriority">
            <li>Vysoká</li>
            <li>Střední</li>
            <li>Nízká</li>
          </ul>
        </div>
        <button>Přidat</button>
      </div>
    </form>
  </section>
  <section class="list"></section>
  <div class="template">
    <div data-index class="note"> flex
      <div class="priority"></div>
      <p class="text"></p>
      <button type="button" class="delete">Smazat</button>
    </div>
  </div>
</div>
```

Obrázek 25: Sekce s poznámkami (zdroj: vlastní tvorba)

4.2.4 Sekce s nastavením

Sekce s nastavením obsahuje dvě položky. Tou první je zapnutí / vypnutí tmavého režimu (element button této položky mění svůj text dle nastaveného režimu „Vypnutý“ / „Zapnutý“). Druhá položka nastavuje barevný přechod. K nastavení barvy využívá dvou tlačítek – elementů input typu color, ty po kliknutí aktivují výběr barvy generovaný prohlížečem. Pozadí obou tlačítek se po změně barevného přechodu změní na vybrané barvy, proto atribut *style* obsahuje CSS kód background: var(--trans-*), jedná se o CSS proměnné popsané v kapitole 4.3.1.

```
<div class="page" id="settings">
    <div class="item">
      <p>Tmavý režim</p>
      <button type="button" name="theme"></button>
    </div>
    <div class="item">
      <p>Barevný přechod</p>
      <div class="color-inputs">
        <input type="color" name="trans-left" style="background: var(--trans-left)" value=""></input>
        <input type="color" name="trans-right" style="background: var(--trans-right)" value=""></input>
      </div>
    </div>
```

Obrázek 26: Sekce s nastavením (zdroj: vlastní tvorba)

4.3 CSS

Prvních 184 řádků obsahuje kód normalize.css pro odstranění rozdílů mezi prohlížeči, jak je zmíněno v kapitole 3.2.2.3. Font je importován zavináčovým pravidlem z cloudové služby Google fonts způsobem popsaným v kapitole 3.2.2.5 a výchozí velikost html elementu je nastavena na *20* CSS pixelů.

Všem elementům je deklarována vlastnost webkit-tap-highlight-color: transparent, aby nedocházelo k výchozímu zvýrazňování po dotyku na element v dotykových zařízeních, které je z hlediska uživatelského prožitku dle názoru autora přitěžující.

4.3.1 CSS proměnné a tmavý režim

Pro celý dokument jsou CSS proměnnými (viz kapitola 3.2.2.8) nastaveny čtyři barvy. Dvě pro lineární přechod, --background pro barvu pozadí elementů aplikace a --color pro barvu textu a stínů prvků aplikace. Všechny elementy, které mají nastavenou barvu vycházející z tmavého či světlého režimu, mají nastavenou vlastnost transition na 1 vteřinu, aby nedocházelo k instantní změně režimu aplikace, ale k plynulému přechodu z tmavých na

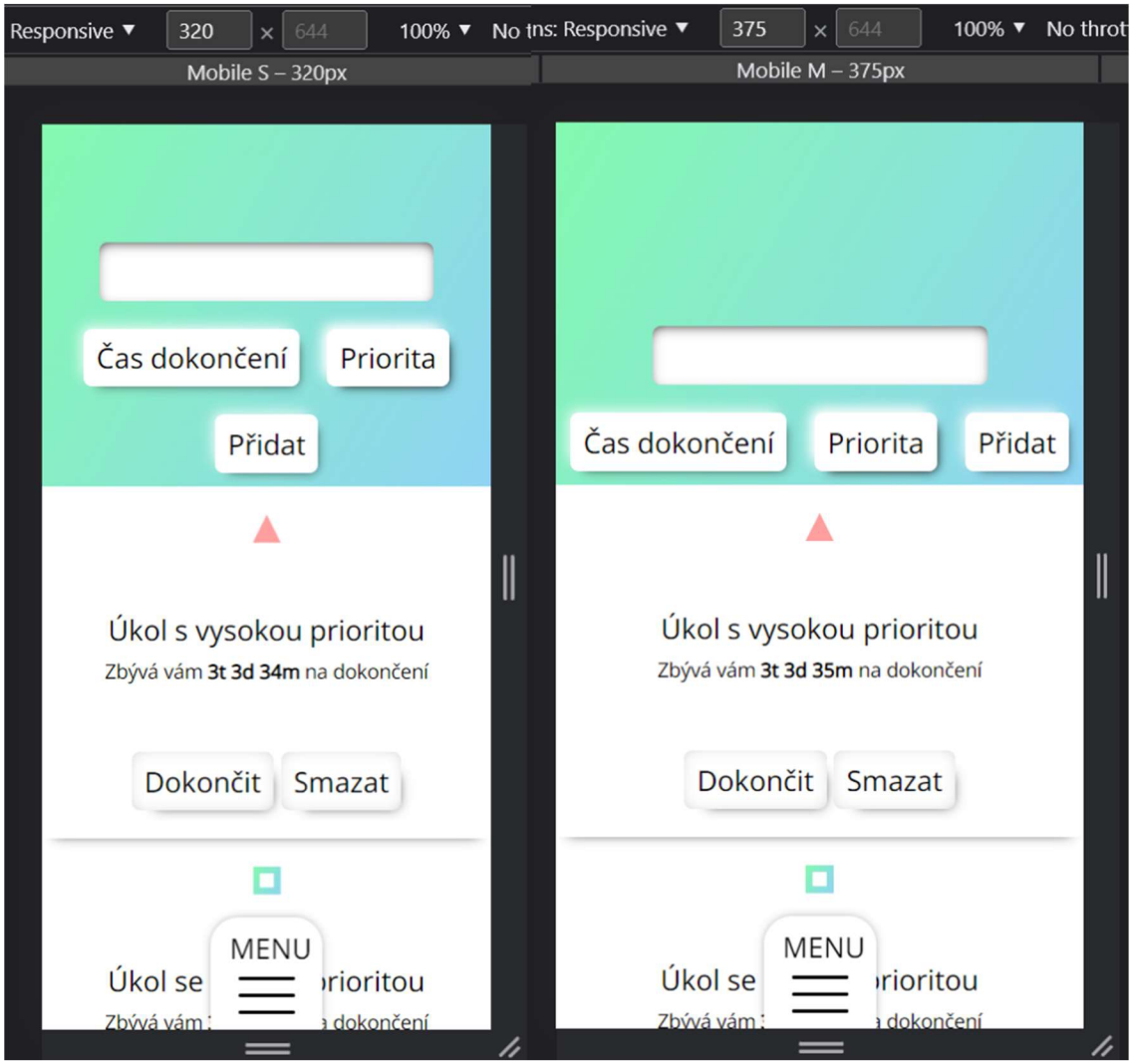
světlé barvy a naopak. Zavináčovým pravidlem @media (prefers-color-scheme: dark) se režim aplikace nastaví nejprve dle výchozího nastavení prohlížeče a následně dle toho, jak si to uživatel nastaví sám.

```css
:root {
  --trans-left: #84fab0; /* Levá část přechodu */
  --trans-right: #8fd3f4; /* Pravá část přechodu */
  --background: #fff;
  --color: #222;
}

body, button, input {
  color: var(--color);
  background: var(--background);
  transition: all 1s;
}

@media (prefers-color-scheme: dark) {
  :root {
    --background: #000;
    --color: #fff;
  }
}
```

Obrázek 27: Deklarování CSS proměnných a tmavého režimu (zdroj: vlastní tvorba)

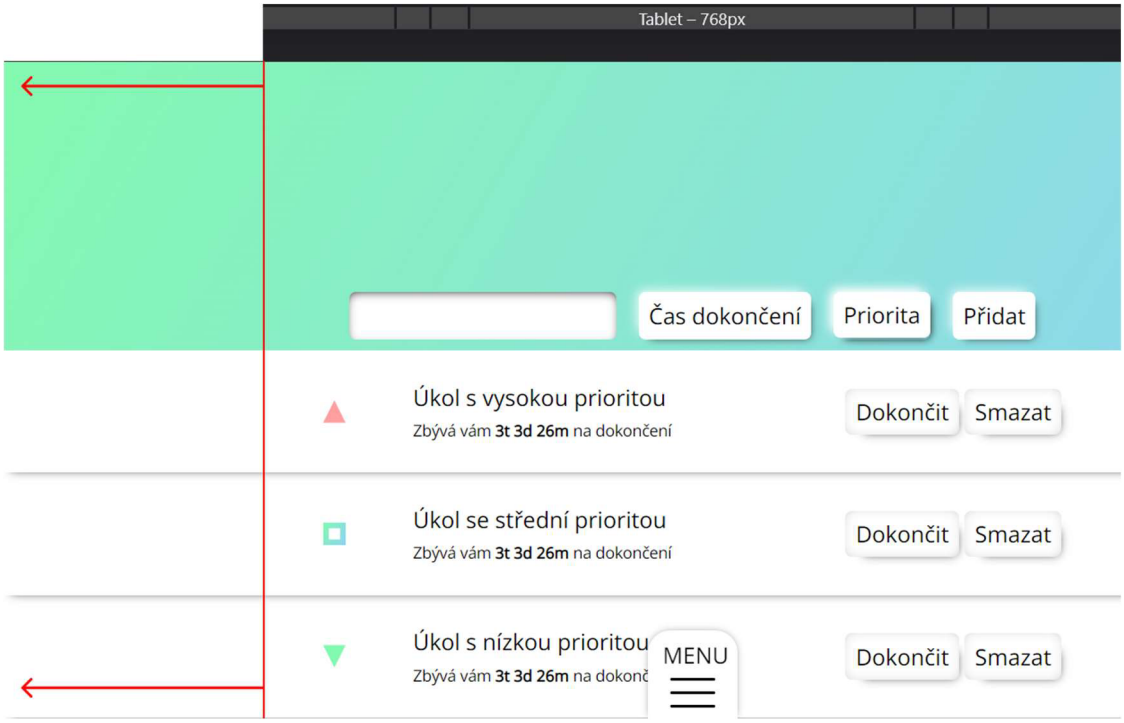
4.3.2 Responzivita aplikace

Jak je zmíněno v kapitole 4.1, design vychází z mobilních zařízení a je navržen pro telefon iPhone XR, který má šířku smartphonu velikosti M (kolem 375 CSS pixelů). Veškeré složitější prvky, které se dají rozložit na širší obrazovce, proto mají deklarovanou vlastnost flexbox, která zajišťuje, že na větších displejích jsou prvky zarovnány do řady, zatímco na menších se řadí pod sebe.



Obrázek 28: Zařízení šířky S a M (zdroj: vlastní tvorba)

Maximální šířka obsahu aplikace je určena velikostí tabletu (768 CSS pixelů). Michálek (2017) uvádí, že širší textový obsah by způsobil na širokých displejích nepřehlednost a k množství obsahu je dle autora širší velikost kontraproduktivní. Obsah je zarovnán na střed obrazovky a za touto hranicí končí. Zbylá plocha viewportu zařízení je po obou stranách symetricky vyplněna prázdným místem se zachováním designu elementů, viz obrázek 29.



Obrázek 29: Zobrazení na širších zařízeních (zdroj: vlastní tvorba)

4.3.3 Usnadněný přístup osobám se zrakovým postižením

Jak bylo popsáno v kapitole 3.2.2.9, pro osoby se zhoršeným zrakem existuje nastavení webového prohlížeče na zvětšení obsahu webu. Aplikace byla napsána tak, aby se veškeré důležité prvky daly zvětšovat pomocí tohoto nastavení.

4.4 JavaScript

4.4.1 Caching verze

Aby prohlížeč pokaždé načetl správný skript a stahoval nový pouze pokud v něm proběhly změny, je třeba zajistit *caching*. Atributu src v elementu script s atributem src „app.js“ se nastaví text tohoto atributu na „app.js?v= + *aktuální verze aplikace*“. Pro promítnutí změny uživateli, který má prohlížeč se starým cache souborem skriptu, stačí verzi přepsat na novější.

4.4.2 Načtení aplikace

Pro načtení aplikace je využit *event-handler* popsaný v kapitole 3.2.3.2. Při první návštěvě se uživateli zobrazí stránka s úkoly. Při opakovaných návštěvách se však vždy načte

uživatelem naposledy navštívená stránka uložená v lokálním úložišti. Následně se z lokálního úložiště načtou uživatelem vytvořené úkoly a poznámky a vypíší se do DOM.

```
window.onload = function (){
  const page = localStorage.getItem("page");
  if(page === null){
    selectPage("tasks");
  } else {
    selectPage(page);
  }
  loadTasks();
  loadNotes();
  updateList();
}
```

Obrázek 30: Načtení aplikace (zdroj: vlastní tvorba)

4.4.3 Ovládání aplikace

Aplikace se ovládá pomocí uživatelem spuštěných událostí, které jsou popsané v kapitole 3.2.3.2, ty spouští podprogramy jim přidané. Dynamické změny, jako je otevření menu, které změní svou pozici, vysune část skrytou pod displejem, zaplní celou obrazovku a transformuje piktogram hamburger menu na červený křížek indikující uzavření po kliknutí, se provedou přidáním či odebráním třídy atributu class v DOM. Tato změna po zaznamenání prohlížečem vyvolá napsanou CSS animaci, která se provede v deklarovaném čase.

V příkladu s navigací se bude jednat o přepínání třídy active pomocí funkce toggle zavolanou po kliknutí na hamburger piktogram a zavření menu po kliknutí na zvolenou stránku. Aktuální stránka po této události zmizí nastavením CSS vlastnosti display na none a zvolená stránka se zobrazí přepsáním vlastnosti display v atributu style na block. V případě navigace se nesmí zapomenout, že uživatel může kdykoliv aplikaci zavřít a chtít se vrátit následným otevřením zpět na stránku, kde skončil. To lze zařídit uložením stránky do lokálního úložiště, které tuto informaci poskytne při načtení aplikace napsaným podprogramem selectPage(), viz předchozí kapitola. Menu aplikace mění pro intuitivnější prožitek z používání při otevření popisek „MENU“ na „ZAVŘÍT“ (a naopak v případě zavření). Učinit tak lze pomocí vlastnosti textContent uzlu v DOM, které se změní požadovaný text.

```
const hamburger = document.querySelector(".hamburger"),
      mainMenu = document.querySelector(".main-menu"),
      options = mainMenu.querySelector(".options").querySelectorAll("p");

hamburger.addEventListener("click", function() {
  mainMenu.classList.toggle("active");
  let text = mainMenu.querySelector("p");
  if(text.textContent == "MENU"){
    text.textContent = "ZAVŘÍT";
  } else {
    text.textContent = "MENU";
  }
});

options.forEach(link => {
  link.addEventListener("click", function() {
    mainMenu.classList.remove("active");
    mainMenu.querySelector("p").textContent = "MENU";
  });
});

function selectPage(name) {
  document.querySelectorAll(".page").forEach(page => {
    if (page.id === name) {
      page.style.display = "block";
    } else {
      page.style.display = "none";
    }
  });
  localStorage.setItem("page", name);
}
```

Obrázek 31: Ovládání navigace a přepínání „stránek“ (zdroj: vlastní tvorba)

4.4.3.1 Přidávání časových rámců

Jak je zmíněno v kapitole 4.2.2.1, časové rámce se přidávají tlačítky „+“ a „-“ ve formuláři pro zadání termínu úkolu. Z obrázku 23 lze vyčíst, že element button v časovém rámci má atribut onclick, tento element spustí napsaný skript v atributu. Jedná se o funkce increaseValue(input) a decreaseValue(input) definované ve zdrojovém skriptu app.js. Obě funkce nejdříve zabrání defaultní akci tlačítka definované prohlížečem (způsobí obnovení stránky), poté přidají jeden časový rámec a následně zjistí maximální či minimální uvedenou hodnotu atributu max / min (viz obrázek 23). Pokud výsledné číslo po přičtení nesplňuje požadovaný limit, upraví jeho hodnotu zavoláním patřičné funkce na snížení / zvýšení dané hodnoty.

```
const minutes = document.getElementById("minutes"),
      hours = document.getElementById("hours"),
      days = document.getElementById("days"),
      weeks = document.getElementById("weeks"),
      months = document.getElementById("months");

let completionDate;

function increaseValue(input) {
  preventDefault();
  input.value = parseInt(input.value) + 1;
    if (input.value > parseInt(input.getAttribute("max"))) {
      decreaseValue(input);
    }
}

function decreaseValue(input) {
  preventDefault();
  input.value = parseInt(input.value) - 1;
    if (input.value < parseInt(input.getAttribute("min"))) {
      increaseValue(input);
    }
}
```

Obrázek 32: Funkce pro přidávání časových rámců (zdroj: vlastní tvorba)

4.4.4 Práce s uživatelskými daty v operační paměti

4.4.4.1 Úkoly

Data jsou uložena na principu objektového programování, každý úkol či poznámka jsou tedy uloženy jako objekty v seznamu. Třída pro objekt se dá v JS vytvořit více způsoby, v aplikaci je zvolen způsob napsání třídy jako funkce s danými parametry, která vrátí předané parametry jako jeho vlastnosti.

Při řešení praktické části této práce bylo zjištěno, že ke každému úkolu se kvůli lokálnímu úložišti, které nedokáže uložit metodu úkolu, jelikož nedokáže uložit funkce obecně (Úvod do JSON, n.d.), musí přidat funkce, která je uložena do proměnné timeLeftFnc. Ta vrátí textový řetězec zbývajícího času na dokončení úkolu, popřípadě logickou hodnotu false, pokud měl být úkol již dokončen. Celý objekt se vytvoří zavoláním konstruktoru new Task([parametry]).

```
function Task(text, priority, date, done){
  return {
    text,
    priority,
    date,
    done
  };
}

function tasksAssFnc(){
  tasks.forEach(task => {
    task.timeLeft = timeLeftFnc;
  });
}
```

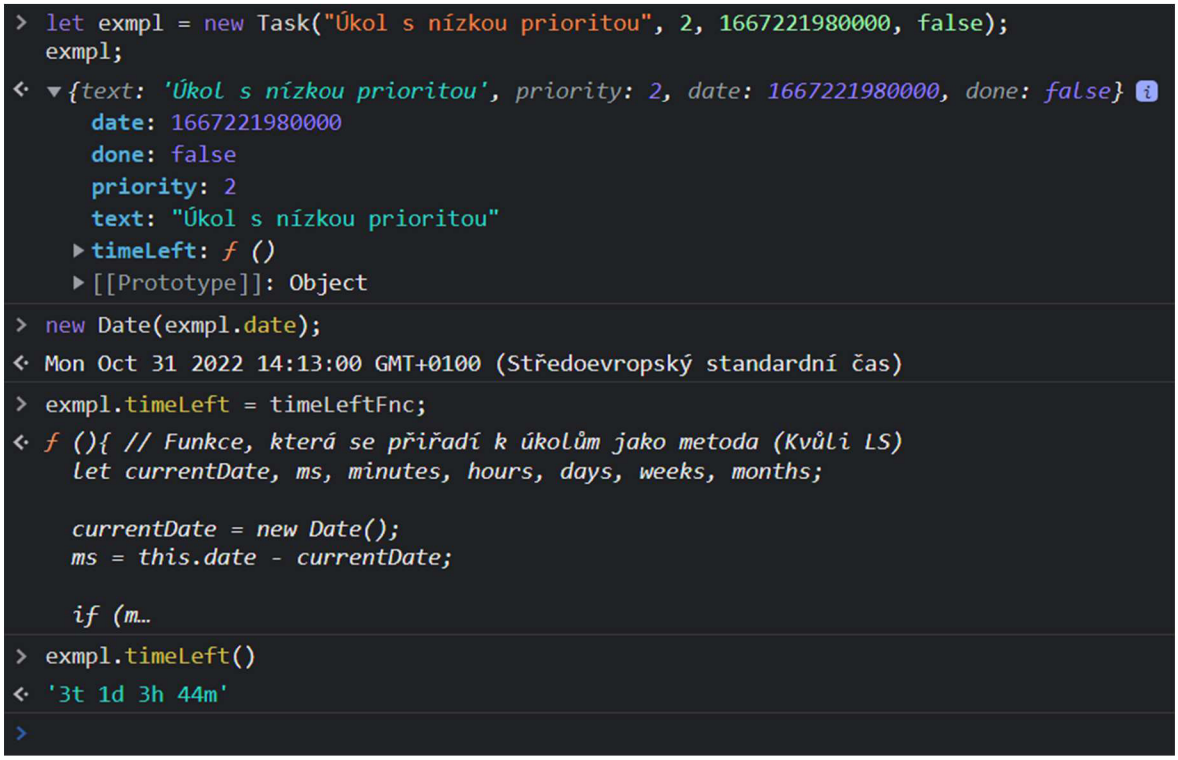
Obrázek 33: Datová struktura jednotlivých úkolů (zdroj: vlastní tvorba)

Výše zmíněná metoda nejdříve uloží aktuální datum s časem a odečte ho od datumu s časem úkolu. To je možné díky tomu, že v JS je datum uloženo v objektu, který má v sobě počet milisekund od první hodiny ranní prvního ledna 1970 (v zóně GMT+0100), viz obrázek 34.

```
> let d = new Date(0)
<· undefined
> d
<· Thu Jan 01 1970 01:00:00 GMT+0100 (Středoevropský standardní čas)
> d.getTime()
<· 0
```

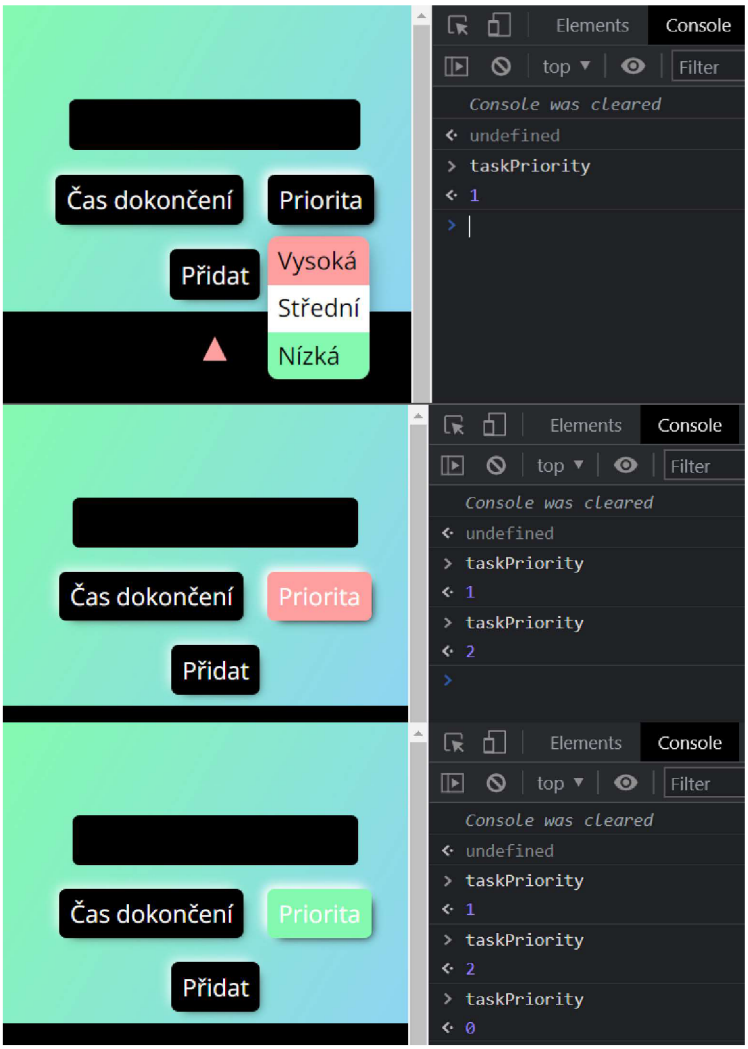
Obrázek 34: Vytvoření data předáním parametru milisekund, zjištění milisekund metodou getTime() (zdroj: vlastní tvorba)

Následně vypočítá několika operacemi dělení a *modulo* (v JS označen znakem „%“) časové rámce, které nejprve vypíše sestupně a následně pouze ty, které mají větší hodnotu než 0. Tyto kroky vytvoří textový řetězec, který na příkladu u úkolu, na jehož dokončení bude zbývat 5 dní, 3 hodiny, 25 minut a 6 vteřin, bude obsahovat *5d 3h 25m 6s*.



Obrázek 35: Vytvoření objektu úkolu a přidání jeho metody (zdroj: vlastní tvorba)

Pro každé přidání úkolu se musí propojit jednotlivé elementy v DOM s JS pomocí uložení jejich odkazu do proměnné. Pro vytvoření úkolu je třeba získat elementy obsahující text úkolu, prioritu a datum dokončení. U textového elementu se zjišťuje jeho hodnota pomocí DOM vlastnosti value. Priorita se ukládá do proměnné taskPriority, výši priority představuje celočíselná hodnota v intervalu <2;0>, pro každý list-item priority je připojen naslouchač událostí (popsaný v kapitole 3.2.3.2), který zjistí výši zvolené priority a nastaví dle toho globální proměnnou taskPriority. Barva tlačítka pro nastavení priority je zabarvená dle aktuálně zvolené priority, viz obrázek 36.



Obrázek 36: Nastavení priority a vizuální zpětná vazba (zdroj: vlastní tvorba)

Jak je popsáno v kapitole 4.2.2.1, datum dokončení se dá vytvořit přidáváním časových rámců či zvolením přesného datumu a času. Pokud má uživatel zvolenou možnost přidávání časových rámců, tudíž tlačítko na přepínání vstupu má hodnotu „Datum“, stanoví se termín dokončení přidáním uživatelem nastavených časových rámců. K tomu je využita knihovna funkcí moment.js, která umožní tyto rámce přidat k aktuálnímu datu s časem. V opačném případě se zvolí datum nastavené v elementu input typu date-time. Přidání k aktuálnímu datu proběhne v momentě, kdy se spustí vytvořená funkce updateDate() kliknutím na tlačítko „Přidat“.

```
const minutes = document.getElementById("minutes"),
      hours = document.getElementById("hours"),
      days = document.getElementById("days"),
      weeks = document.getElementById("weeks"),
      months = document.getElementById("months");

let completionDate;
function updateDate() {
  if(setDateInp.textContent == "Datum"){
    completionDate =  moment().add(parseInt(months.value), "months")
                              .add(parseInt(weeks.value), "weeks")
                              .add(parseInt(days.value), "days")
                              .add(parseInt(hours.value), "hours")
                              .add(parseInt(minutes.value), "minutes");
    months.value = "0";
    weeks.value = "0";
    days.value = "0";
    hours.value = "0";
    minutes.value = "0";
  } else {
    let inp = document.querySelector("#set-datetime-inp").value;
    completionDate = moment(inp);
  }
}
```

Obrázek 37: Globální proměnné časových rámců a funkce updateDate() (zdroj: vlastní tvorba)

Pokud je vyplněn text úkolu, přidá se po stisku tlačítka „Přidat“ úkol do seznamu úkolů. Jak je zmíněno výše, nejdříve se spustí funkce updateDate(), následně se vytvoří proměnná task s objektem třídy Task() s textem úkolu, prioritou a časem ve formátu objektu knihovny *moment.js*, z níž se vezme datum objektu typu Date() pomocí vlastnosti _d, a úkol se nastaví na nedokončený. Tento úkol se přidá do seznamu úkolů, poté se zavolá funkce updateList(), popsaná v následujícím odstavci, která vypíše všechny úkoly do DOM. Formulář se následně vrátí do původního stavu, textový vstup se nastaví na prázdný, priorita se nastaví na výchozí a jejímu tlačítku se vrátí původní vzhled, a zároveň zmizí rozbalovací menu na výběr priority. Na závěr se všechny úkoly uloží do lokálního úložiště zavoláním funkce saveTasks() – podobná funkce pro uložení poznámek je popsána v kapitole 4.4.5.

Funkce updateList() přiřadí všem úkolům funkci na vrácení zbývajícího času, setřídí seznam úkolů a poznámek, vyčistí zastaralý DOM seznam úkolů a poznámek, do kterého následně pomocí metody forEach() vloží všechny úkoly zavoláním funkce listTask(obj, index).

```
// Přidání úkolu
taskForm.querySelector(".second-item").lastElementChild.onclick = function() {
  let textInput = document.getElementById("taskText");
  if (textInput.value === "") {
    alert("Zadejte text úkolu.");
  } else {
    updateDate();
    let task = new Task(textInput.value, taskPriority, completionDate._d, false);
    tasks.push(task);
    updateList();
    textInput.value = "";
    taskPriority = 1;
    document.querySelector("#tasks .btn.dropdown").className = "btn dropdown";
    document.querySelector("#taskPriority").classList.remove("show");
    saveTasks();
  }
};

// Update listu
function updateList() {
  tasksAssFnc();
  sortTasks();
  sortNotes();
  tasksList.textContent = "";  // Vyčištění listu před přidáním setřízených úkolů
  notesList.textContent = "";
  tasks.forEach(function(task){
    listTask(task, tasks.indexOf(task));
  });
  notes.forEach(function(note){
    listNote(note, notes.indexOf(note));
  });
}
```

Obrázek 38: Přidání úkolu a update listu (zdroj: vlastní tvorba)

Funkce listTask(obj, index) nejdříve naklonuje HTML šablonu úkolu pomocí metody cloneNode(), následně uloží odkazy na její vnořené elementy a atribut data-set-index a vloží všechny údaje do patřičných elementů a atributů. Elementu priority připíše třídu low či high, pokud se jedná o úkol s nízkou či vysokou prioritou, tato třída nastaví elementu barvu dané priority a tvar šipky nahoru či dolů. Dále přidá tlačítkům do atributu onclick funkci na odstranění či dokončení úkolu, té předá pořadí daného úkolu, aby interpret JS věděl, který úkol má dokončit či odstranit. Pokud měl být úkol již dokončen, nastaví nápis pod textem úkolu na tučný „Úkol již měl být splněn!“, v opačném případě nastaví tento text na „Zbývá vám“ + tučným písmem textový řetězec metody timeLeft() + „na dokončení“. Poslední příkaz této funkce vezme DOM seznam úkolů a metodou appendChild() vloží na konec tohoto seznamu vytvořený HTML element úkolu.

Aby se dal úkol dokončit či vrátit mezi dokončené, je napsána funkce fulfillTaskToggle(index). Jak je již dříve zmíněno, tato funkce se volá pomocí tlačítka na

dokončení / vrácení úkolu a je jí předáno pořadí úkolu. Uvnitř funkce je podmínka zjišťující, zda je úkol dokončen. Pokud ano, přidá úkolu v DOM třídu slideaway, která spustí animaci odsunutí úkolu pryč, následně přidá naslouchač události transitionend, který se spustí po dokončení animace. V ten moment se úkol označí za dokončený a spustí se vnořená funkce update(), která změnu uloží do lokálního úložiště a obnoví seznam, aby byly změny zaznamenány v DOM. V opačném případě proběhne ta stejná věc, úkol se ale označí zpět na nedokončený. Funkce deleteTask(index) provede stejnou změnu, úkol však před uložením a promítnutím změny v DOM vymaže ze seznamu pomocí metody splice(index, číslo), která odstraní jeden objekt (úkol) z jeho stávajícího místa v seznamu. Uživatel má tak animovanou změnu odstranění, dokončení či vrácení úkolu.

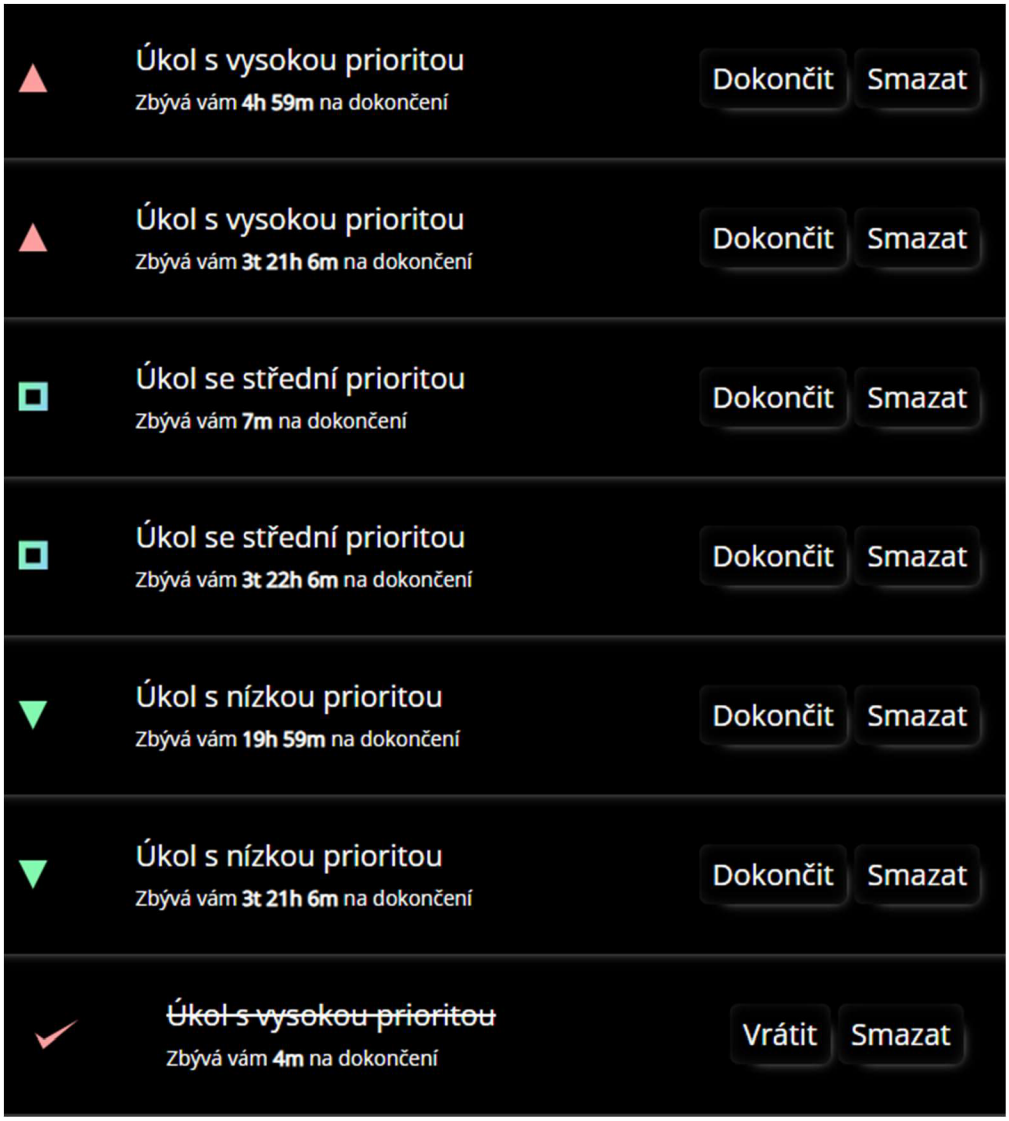
```
// Toggle na dokončení úkolu
function fulfillTaskToggle(i) {
  let div = tasksList.querySelector("[data-index='"+ i +"']");
  function update(){
    saveTasks();
    updateList();
  }
  if (tasks[i].done === true) {
    div.classList.add("slideaway");
    div.addEventListener("transitionend",function(){
        tasks[i].done = false;
        update();
    });
  } else {
    div.classList.add("slideaway");
    div.addEventListener("transitionend",function(){
        tasks[i].done = true;
        update();
    });
  }
}

function deleteTask(i) {
  let div = tasksList.querySelector("[data-index='"+ i +"']");
  div.classList.add("slideaway");
  div.addEventListener("transitionend",function(){
    tasks.splice(i, 1);
    saveTasks();
    updateList();
  });
}
```

Obrázek 39: Přepínání dokončení úkolu a odstranění úkolu (zdroj: vlastní tvorba)

V úvodu praktické části je popsáno třídění úkolů. Pro to je napsána funkce sortTasks(). Je volána pokaždé, kdy uživatel provede jakoukoliv změnu, která by se v seznamu úkolů mohla promítnout. Třídění dle kritéria je prováděno pomocí metody sort(a, b), která vrací podmínku řazení. Pokaždé se tedy porovnávají dvě vlastnosti, dle kterých se úkol seřazuje, znaménkem mínus.

```
// Třízení úkolů
function sortTasks() {
  tasks.sort(function (a, b) {
      return a.date - b.date;
  });
  tasks.sort(function (a, b) {
      return b.priority - a.priority;
  });
  tasks.sort(function (a, b) {
      return a.done - b.done;
  });
}
```

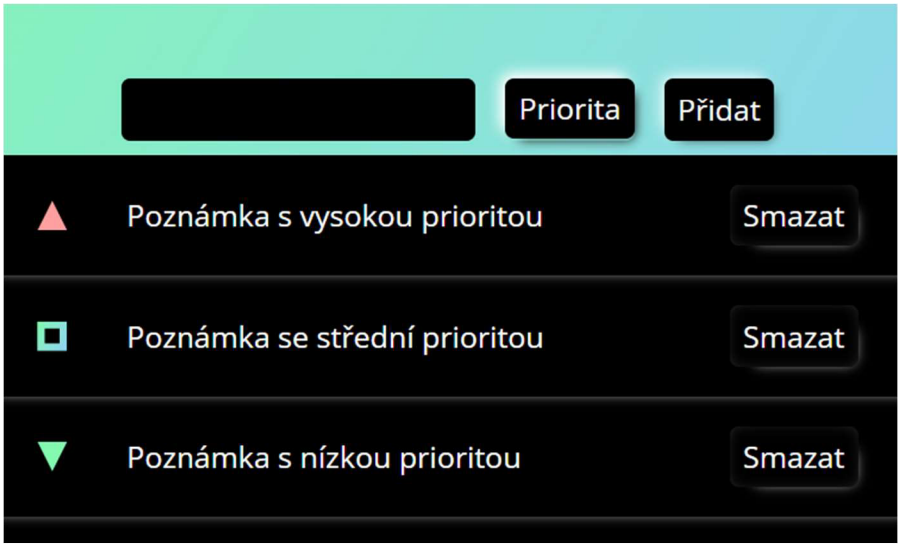
Obrázek 40: Třídění úkolů (zdroj: vlastní tvorba)



Obrázek 41: Ukázka třídění úkolů (zdroj: vlastní tvorba)

4.4.4.2 Poznámky

Poznámky fungují na stejném principu jako úkoly s tím rozdílem, že je odstraněn veškerý kód týkající se práce s časem a dokončováním, jelikož poznámky tuto funkci nepotřebují. Poznámky mají pouze text, řadí se dle priority a dají se odstranit. Kód je z toho důvodu výše popsaným způsobem upraven. Z napsaného textu vyplývá, že výhoda takto napsaného kódu je opětovná použitelnost.



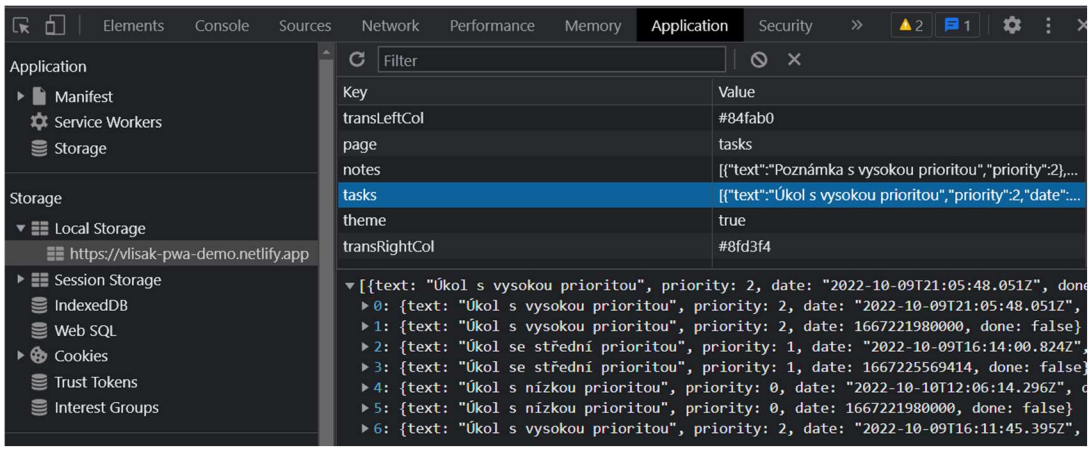
Obrázek 42: Ukázka sekce poznámek (zdroj: vlastní tvorba)

4.4.5 Operace v lokálním úložišti

V lokálním úložišti jsou uloženy následující údaje potřebné k opětovnému použití při otevření aplikace:

- Barevný přechod;
- aktuální stránka;
- vytvořené poznámky a úkoly;
- použití tmavého režimu.

V desktopovém prohlížeči Chrome je přístup k těmto datům v nástrojích pro vývojáře. Vývojář si tím může zkontrolovat, zda se data do lokálního úložiště ukládají a jakým způsobem.



Obrázek 43: Ukázka lokálního úložiště (zdroj: vlastní tvorba)

Z obrázku 43 je patrné, že lokální úložiště má datovou strukturu slovníku, kde každá položka funguje jako přístupový klíč, a do jeho hodnoty se ukládají data. Do položky lokálního úložiště se uloží data zavoláním metody setItem(klíč) a získají se metodou getItem(klíč). Na obrázku 44 je zobrazeno získání a uložení dat v podobě seznamu, konkrétně objektů s poznámkami. Při načítání dat se nejdříve ověří, zda je obsah poznámek prázdný, tudíž by položka v lokálním úložišti ani nemohla existovat. Pokud ano, není co načíst a z funkce se vyskočí. V opačném případě se načtou data v podobě seznamu metodou parse() zavolanou na objektu JSON, který umožňuje data převést z JSON řetězce na seznam a naopak. Data se uloží metodou stringify(), která seznam poznámek převede na textový řetězec ve formátu JSON, následně se tato data uloží do lokálního uložiště zmiňovanou metodou setItem().

```
// Automatické ukládání do Lokálního úložiště
function loadNotes() {
  let ls = localStorage.getItem("notes");
  if (ls.length === 0 || ls === null) {
    return;
  }
  let notesDeserialized = JSON.parse(ls);
  notes = notesDeserialized;
}

function saveNotes() {
  let notesSerialized = JSON.stringify(notes);
  localStorage.setItem("notes", notesSerialized);
}
```

Obrázek 44: Ukládání a načítání dat z lokálního úložiště (zdroj: vlastní tvorba)

4.4.6 Asynchronní obnova dat

Jelikož JS je programovací jazyk, který běží pouze na jednom vlákně, nelze napsat prostý nekonečný cyklus na obnovení dat, který by běžel asynchronně se zbytkem programu, jelikož po spuštění takového cyklu by byl interpret zaseklý právě v takovém cyklu a uživatel by nemohl s aplikací interagovat. Tento problém se vyřeší asynchronní funkcí setInterval(funkce, celočíselný počet milisekund), která po jejím zavolání spustí interval, který za čas uvedený v druhém parametru spustí funkci uvedenou jako parametr první. V tomto případě se bude jednat o funkci, která pro každý úkol zjistí jeho uzel v DOM a změní text zbývajícího času na aktuální. Tento kód interval provede jednou za vteřinu, viz obrázek 45.

```
// Asynchroní aktualizace DOM listu s úkoly

setInterval(function () {
  tasks.forEach(function (task) {
    let taskDiv = tasksList.querySelector("[data-index='"+ tasks.indexOf(task) +"']"),
        timeElement = taskDiv.getElementsByClassName("time")[0].getElementsByTagName("span")[0];

    if (task.timeLeft() === false) {
      let timeElementParent = taskDiv.getElementsByClassName("time")[0];
      timeElementParent.innerHTML = "";
      let outOfTimeText = document.createTextNode("Úkol již měl být splněn!");
      let outOfTimeEl = document.createElement("span");

      outOfTimeEl.appendChild(outOfTimeText);
      timeElementParent.appendChild(outOfTimeEl);
    } else {
      timeElement.textContent = task.timeLeft();
    }
  });
}, 1000);
```

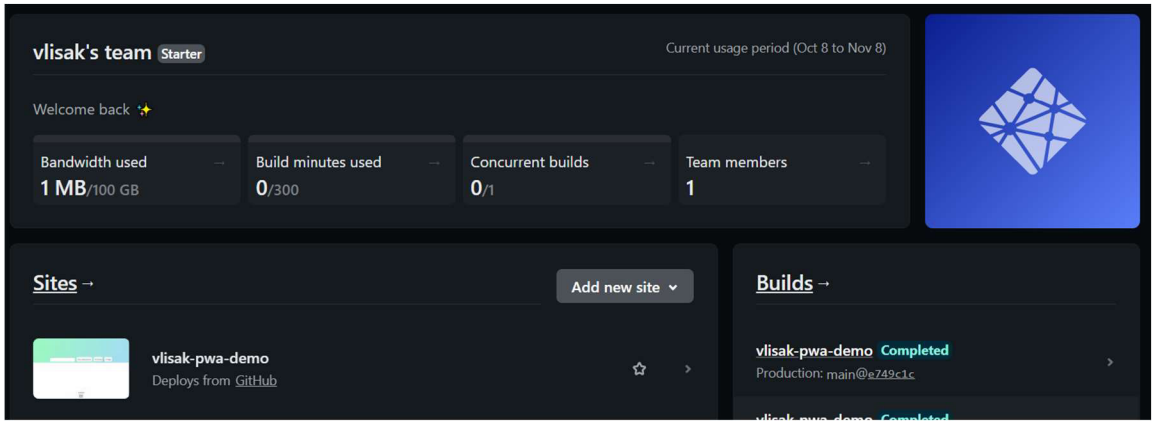
Obrázek 45: Asynchronní aktualizace zbývajícího času úkolů (zdroj: vlastní tvorba)

4.5 Přeměna na progresivní webovou aplikaci

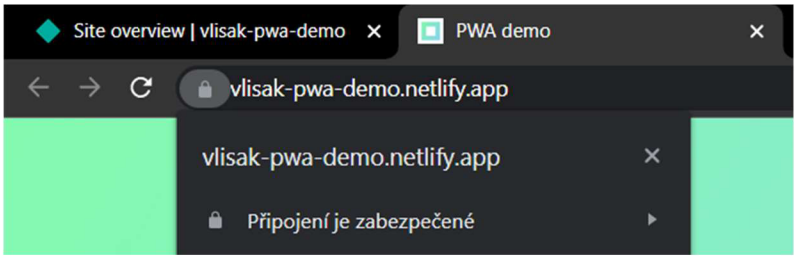
V kapitole 3.2.4 se píše o základních předpokladech pro přeměnu webové aplikace na PWA. Aby PWA mohla fungovat, potřebuje server se zabezpečeným HTTPS protokolem, dále webový manifest a pro funkčnost off-line je zapotřebí service worker, který zajistí asset caching.

4.5.1 Hosting s HTTPS protokolem

Pro hosting je zvolena aplikace *Netlify* s repositářem *Github*, popsané technologie v kapitole 3.2.4.1. Pro aplikaci vyvíjenou ve vlastní práci je zvolena doména „vlisak-pwa-demo.netlify.app“.



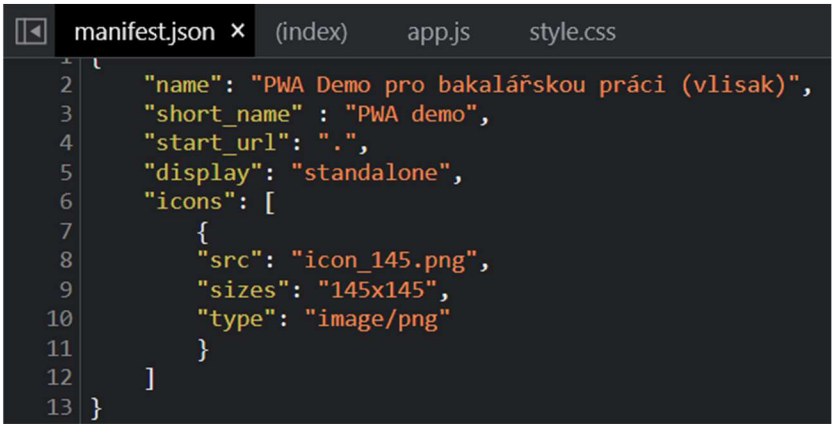
Obrázek 46: Úvodní stránka Netlify po přihlášení s publikovaným projektem a přehledem posledních změn (zdroj: vlastní tvorba)



Obrázek 47: Ukázka URL aplikace se zabezpečeným protokolem (zdroj: vlastní tvorba)

4.5.2 Manifest PWA

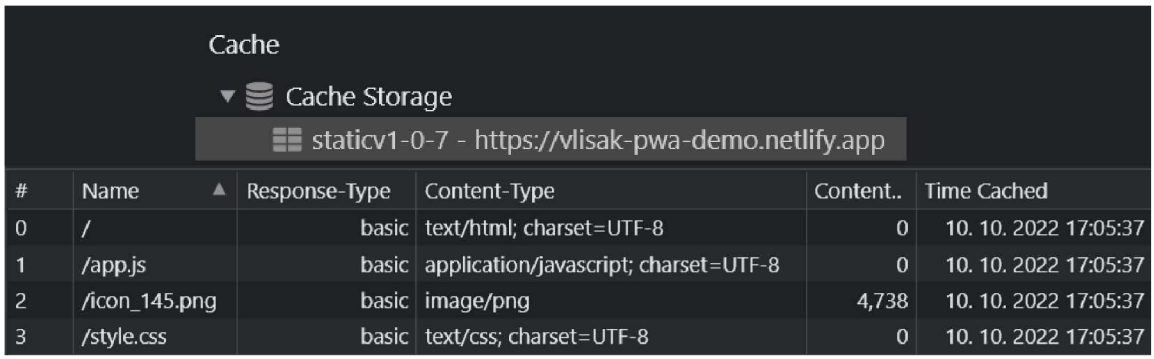
V kapitole 3.2.4.2 je popsán postup při tvorbě manifestu. V manifestu demonstrované aplikace je zvoleno *jméno* „PWA Demo pro bakalářskou práci (vlisak)“, tento název se bude zobrazovat například v názvu okna spuštěné aplikace na platformě Windows, v zástupci aplikace, nebo například v operačním systému Android při načítání aplikace. Krátký název je užíván například pod ikonou aplikace na mobilních zařízeních (Android, iOS). Tečka ve vlastnosti start_url značí, že aplikace bude začínat na výchozí stránce, v případě demonstrované aplikace se zobrazí soubor index.htm. Zobrazení jako samostatné aplikace v samostatném okně se nastaví vlastností display na standalone. Jako *ikona* je zvolen jeden PNG obrázek.



Obrázek 48: Manifest demonstrované aplikace (zdroj: vlastní tvorba)

4.5.3 Service worker

V kapitole 3.2.4.3 jsou uvedeny funkce PWA, které přináší *service worker*, v případě demonstrované aplikace se jedná o *„Asset caching“*, který umožní, aby aplikace mohla běžet i off-line. V kódu app.js se spustí kód, který zaregistruje service worker, pokud existuje. V service workeru je kód, který po spuštění této události otevře *cache* prohlížeče, stáhne soubory index.html, style.css, app.js a ikonu aplikace a uloží ji pod klíčem aktuální verze. Následně odstraní veškeré soubory, které se neshodují s klíčem aktuální verze, a zůstanou tak pouze soubory aktuální verze. Aplikace poté funguje i off-line, jelikož veškerý zdrojový kód poslední verze, kdy zařízení bylo online, je stažený. Na obrázku č. 49 je index.htm uložený pod jménem „/“, a to z důvodu, že je to výchozí soubor, který prohlížeč vrací po navštívení URL adresy aplikace. Jedná se o stejný princip, proč uživatelé nemusí psát „vlisak-pwa-demo.netlify.app/index.htm“ ale stačí jen „vlisak-pwa-demo.netlify.app“, popsaný na začátku kapitoly 4.2.



#	Name	Response-Type	Content-Type	Content..	Time Cached
0	/	basic	text/html; charset=UTF-8	0	10. 10. 2022 17:05:37
1	/app.js	basic	application/javascript; charset=UTF-8	0	10. 10. 2022 17:05:37
2	/icon_145.png	basic	image/png	4,738	10. 10. 2022 17:05:37
3	/style.css	basic	text/css; charset=UTF-8	0	10. 10. 2022 17:05:37

Obrázek 49: Ukázka cache v nástroji pro vývojáře (zdroj: vlastní tvorba)

5 Výsledky a diskuse

5.1 Testování aplikace

Aplikace byla otestována v ohledech uživatelského prožitku, intuitivnosti, funkčnosti napříč platformami Windows 7-11 (prohlížeč Google Chrome 106), macOS (prohlížeč Google Chrome 106), iOS na iPhone 8 a novější (prohlížeč Safari, 2017) a Android na Samsung Galaxy S5 (prohlížeč Google Chrome, 2016). V poslední řadě byla aplikace auditována nástrojem *Lighthouse*, popsaným v kapitole 3.2.5.1, na prohlížeči Google Chrome v ohledech rychlosti („performance“), přístupnosti („accesibility“) a osvědčených postupů („best practices“).

Testování uživatelského prožitku neboli UIX (User Interface Experience) proběhlo u subjektů náročnějších pro ovládání mobilních zařízení. Oba subjekty mají telefon Xiaomi Redmi Note 10 5G a věk vyšší sedmdesáti let. První subjekt má minimální zkušenost s ovládáním výpočetní techniky obecně, používá ovšem mobilní zařízení platformy Android přes tři roky a předtím užíval mobilní zařízení iPad od roku 2011. Druhý subjekt má sice zkušenost s výpočetní technikou od poloviny 80. let, ale chytré mobilní zařízení ovšem používá necelý půlrok. Oba subjekty měly za úkol vytvořit úkoly různých priorit s přidáním časových rámců či nastavením datumu, vytvořit poznámky různých priorit a změnit si vzhled aplikace. Oba subjekty si všechny ovládací prvky bez vnější pomoci osvojily *do 2 minut*, jediný problém měly s nastavením datumu s časem, jelikož neměly zkušenost s tím, že vstup s datumem a časem se dá nastavit dotykem. Druhý subjekt se domníval, že datum je nastavený na dnešní a konkrétní čas se nastaví dotykem tlačítka „Čas“, které ovšem vede zpět na výchozí možnost přidávání časových rámců. Poskytl zpětnou vazbu, že pokud by byl tento vstup vyobrazen jako tlačítko, kliknul by na něj.

Testování napříč platformami proběhlo prostřednictvím webové aplikace *Browserstack*, popsané v kapitole 3.2.5. Veškeré testování na zařízeních zmíněných v prvním odstavci této kapitoly proběhlo bez problémů, až na zařízení iPhone 8 z roku 2017, které nevykreslilo či vykreslilo špatně několik prvků, nicméně ovládání aplikace nebylo signifikantně zhoršeno.

Nástroj *Lighthouse* vrátil následující výsledky:

- rychlost – *95 %*
- přístupnost – *100 %*
- osvědčené postupy – *92 %*

V ohledu rychlosti se při iniciálním načtení aplikace první text či obrázek načetl za *2,3* vteřiny, největší obrázek či text za *2,4* vteřiny a za stejný čas se aplikace načte na plně interaktivní. Podrobnosti o proběhlém auditu obsahuje následující tabulka.

Tabulka 1: Podmínky zachycení auditu (zdroj: Ligthouse, popsaný v kapitole)

Captured at Oct 15, 2022, 12:06 PM GMT+2	Emulated Moto G4 with Lighthouse 9.6.2	Single page load
Initial page load	Slow 4G throttling	Using Chromium 106.0.0.0 with devtools

5.2 Návrhy na vylepšení

Prvním návrhem je *zpřehlednění časového vstupu*, se kterým měly oba testovací subjekty problém. Tento problém by se mohl vyřešit přidáním jakékoliv indikace, že prvek je nastavitelný. Audit aplikace odhalil, že se dá ušetřit zhruba *1,3* vteřiny načtení prvního textu či obrázku odložením *načítání fontů* a knihovny *moment.js*. Aplikace se dá dále vyvíjet, aby splňovala více nároků na osobního asistenta. Mohla by například poskytovat notifikace o blížících se termínech dokončení úkolů přes service workera a informovat o podílu úspěšnosti dokončení úkolů v termínu dokončení. Na podobném principu jako funkce s úkoly by mohla být vyvinuta funkce sledování dodržování návyků, které si uživatel nastaví. Pokud by daná aplikace měla být komerční, potřebovala by identifikaci návštěvníka k analýze dat o jeho používání aplikace a přístup k trasovacím cookies k implementaci cílené reklamy. Pokud by byl uživatel spokojen s aplikací, byla by mu nabídnuta možnost prémiového členství, které cílené reklamy odstraní a nabídne mu synchronizaci jeho dat napříč užívanými zařízeními spolu se zálohou na serveru.

5.3 Nižší náklady na vývoj a údržbu

V kapitolách 3.1.1 až 3.1.4 je demonstrována teoretická náročnost vývoje *cross-platformové* nativní aplikace, ve stručnosti konkrétně v problematice obsáhlosti *codebase* kvůli front-end kódu, synchronizaci aktuálních verzí na jednotlivé platformy a správě v obchodech s aplikacemi. V případě, že například požadujeme, aby aplikace běžela na všech běžně používaných platformách, jak mobilních, tak desktopových, je třeba ideálně 5 verzí aplikace pro koncové platformy iOS, Android, MacOS, Linux a Windows. To znamená najmutí 5 programátorů či méně, pokud zvládnou vyvinout verze pro všechny platformy a zároveň je udržovat.

Proti tomu PWA přístup nabízí jeden zdrojový kód aplikace pro všechny tyto platformy spustitelný na jedné webové doméně pro všechny koncové zařízení. V praktické části je tento postup demonstrován na vyvinutí aplikace osobního asistenta, která je otestována pro běžně používané platformy v kapitole 5.1. Tímto způsobem byly kvalitativně manifestovány nižší náklady na vývoj a údržbu.

Pro přesnější demonstraci nižších nákladů na vývoj PWA oproti nativnímu přístupu – co se kvantitativních výsledků týče, by bylo zapotřebí například vyvinout aplikaci s *back-end* strukturou na 5 již zmíněných platforem a stejnou aplikaci (ideálně pixel na pixel) vyvinout i PWA přístupem, provést kalkulaci nákladů a ty následně porovnat. Ale zde uvedená tvrzení o jednoznačných výhodách PWA přístupu se opírají o vlastní praktické zkušenosti autora. Byla to ostatně také motivace pro samotný vznik PWA přístupu.

6 Závěr

V teoretické části byly popsány problémy vývoje nativní multiplatformní aplikace a webové technologie potřebné k vytvoření progresivní webové aplikace. Její první část se věnovala velmi abstraktnímu postupu vývoje nativní multiplatformní aplikace. Byly zde popsány problémy s velikostí *codebase*, které jsou zapříčiněny její nutnou rozmanitostí, respektive potřebou vést jednotlivé verze programu pro různé platformy. Poté byl popsán problém s umístěním do obchodu s aplikacemi. V následující části byly vysvětleny technologie progresivní webové aplikace, ovšem pouze z hlediska front-endu. Těmito technologiemi jsou značkovací jazyky HTML a CSS, dále programovací jazyk JavaScript a veškeré technologické postupy nutné k přeměně webové aplikace na progresivní webovou aplikaci.

Praktická část se zabývala vlastním příkladem PWA vývoje mobilní aplikace. Tato část práce využila popsané technologie a postupy, kterým se tato práce věnovala ve druhém úseku teoretické části, k popsání vývoje progresivní webové aplikace sloužící jako osobní asistent. Nejdříve byl popsán design aplikace, následně zdokumentovaná obsahová struktura HTML. Dále byl vysvětlen responzivní layout aplikace vytvořen kaskádovými styly a CSS proměnné, které následně pomocí JavaScriptu umožňují vizuální úpravy uživatelem. Závěrem podkapitoly o CSS bylo vysvětlení použití pružných jednotek pro usnadněný přístup zrakově postiženým. V části o JavaScriptu byl popsán caching verzí, načítání aplikace a její ovládání uživatelem, práce s uživatelskými daty a jejich průběžném ukládání v lokálním úložišti. Na závěr byla popsána asynchronní aktualizace údajů v aplikaci. Poslední kapitola vlastní práce byla věnována popisu implementace postupů pro vytvoření progresivní webové aplikace.

V úplném závěru této práce bylo provedeno testování aplikace, a to jak z hlediska uživatelské přívětivosti, tak z hlediska funkčnosti na různých zařízeních a rychlosti aplikace. Byly otestovány dva subjekty, na jejichž základě byly popsány návrhy na vylepšení vyvinuté aplikace. V poslední řadě byly posouzeny teoretické nízké náklady, které vycházejí z demonstrace menší, a především méně komplexnější *codebase*, která je tímto způsobem snazší na vývoj a údržbu.

7 Seznam použitých zdrojů

ABOUKHADIJE, F. (2014) [online] Twitter. Available at: <https://twitter.com/feross/status/459259593630433280> [Accessed 19 May 2022]

POTVIN, R. and LEVENBERG, J. (2016) *Why Google Stores Billions of Lines of Code in a Single Repository*. [ebook] ACM Digital Library, p.3. Available at: <https://dl.acm.org/doi/pdf/10.1145/2854146> [Accessed 19 May 2022]

PARWAL, A. and KARUPPAIYA, K. (2021) *Scaling of Uber's API gateway*. [online] Uber Engineering Blog. Available at: <https://eng.uber.com/scaling-api-gateway/> [Accessed 19 May 2022]

StatCounter Global Stats (2022) *Mobile Operating System Market Share Worldwide.* [online] Available at: <https://gs.statcounter.com/os-market-share/mobile/worldwide/2022> [Accessed 19 May 2022]

LUETIC, M. (2021) *Cross-platform vs. native mobile development | DECODE.* [online] DECODE Agency. Available at: <https://decode.agency/article/native-vs-cross-platform-mobile-apps/> [Accessed 19 May 2022]

ROOMI, M. (2021) *5 Advantages and Disadvantages of Native App | Drawbacks & Benefits of Native App.* [online] HitechWhizz - The Ultimate Tech Experience. Available at: <https://www.hitechwhizz.com/2021/04/5-advantages-and-disadvantages-drawbacks-benefits-of-native-app.html> [Accessed 19 May 2022]

CAREY, R. (2015) *Electronic Recollections*, By Ricard Carey. [online] AppStorey. Available at: <https://appstorey.com/2015/07/17/electronic-recollections-by-ricard-carey/> [Accessed 19 May 2022]

ZDRAVESKA, S. (2022) *Apple App Store vs Google Play Store (2022 Comparison).* [online] CyberCrew. Available at: <https://cybercrew.uk/software/app-store-vs-play-store/> [Accessed 19 May 2022]

VISWANATHAN, P. (2020) *iOS App Store vs. Google Play Store: Which Is Better for App Developers?* [online] Lifewire. Available at: <https://www.lifewire.com/ios-app-store-vs-google-play-store-for-app-developers-2373130> [Accessed 19 May 2022]

HINDY, J. (2021) *How to install third party apps without the Google Play Store.* [online] Android Authority. Available at: <https://www.androidauthority.com/how-to-install-apks-31494/> [Accessed 19 May 2022]

W3schools.com (2022) *Introduction to HTML*. [online] Available at: <https://www.w3schools.com/html/html_intro.asp> [Accessed 29 June 2022]

GALLAGHER, N., n.d. GitHub - necolas/normalize.css: *A modern alternative to CSS resets.* [online] GitHub. Available at: <https://github.com/necolas/normalize.css/> [Accessed 29 July 2022]

DEL VALE, A. (2020) *Paradigms In JavaScript.* [online] DEV Community. Available at: <https://dev.to/alamin__yusuf/paradigms-in-javascript-1m31> [Accessed 24 August 2022]

ARYAN, A. (n.d) *Functional Programming in JavaScript.* [online] Toptal Engineering Blog. Available at: <https://www.toptal.com/javascript/functional-programming-javascript> [Accessed 24 August 2022]

Jaksenaucitprogramovat.py.cz (n.d) J*ak se naučit programovat: Úvod do funkcionálního programování.* [online] Available at: <http://jaksenaucitprogramovat.py.cz/cztutfctnl.html> [Accessed 24 August 2022]

(n.d.) 18progrjazyky. [ebook] Available at: <http://www.gymozart.8u.cz/souborygympl/elearning/svt/teorie/18progrjazyky.pdf> [Accessed 24 August 2022]

Programiz.com (n.d.) *JavaScript Prototype (with Examples).* [online] Available at: <https://www.programiz.com/javascript/prototype> [Accessed 24 August 2022]

Digicert (n.d.) *What is SSL, TLS and HTTPS?* [online] Available at: <https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https> [Accessed 25 August 2022]

Apps.microsoft.com (n.d.) *Microsoft Apps.* [online] Available at: <https://apps.microsoft.com/store/apps?hl=cs-cz&gl=CZ> [Accessed 2 September 2022]

Apps.apple.com (n.d.) *Stahování na iTunes (Mac App Store).* [online] Available at: <https://apps.apple.com/cz/genre/mac/id39?l=cs&mt=12> [Accessed 2 September 2022]

KYRNIN, J. (2020) *Understanding the Index.html Page on a Website* [online] ThoughtCo. Available at: <https://www.thoughtco.com/index-html-page-3466505> [Accessed 28 September 2022]

W3schools.com (n.d) *HTML input autocomplete Attribute* [online] Available at: <https://www.w3schools.com/tags/att_input_autocomplete.asp> [Accessed 29 September 2022]
W3schools.com (n.d) *HTML input type="datetime-local"* [online] Available at: <https://www.w3schools.com/tags/att_input_type_datetime-local.asp> [Accessed 29 September 2022]
Developer.mozilla.org (n.d.) *Using CSS custom properties (variables) - CSS: Cascading Style Sheets | MDN* [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties> [Accessed 2 October 2022]
(n.d.) *Úvod do JSON* [online] Available at: <https://www.json.org/json-cz.html> [Accessed 8 October 2022]
W3schools.com (n.d.) *W3Schools Tryit Editor* [online] Available at: <https://www.w3schools.com/csS/tryit.asp?filename=trycss3_flexbox_website2> [Accessed 8 October 2022]
PEHLIVANIAN, A., NGUYEN, D. (2014) *Javascript okamžitě*, Computer Press, ISBN 978-80-251-4163-2
KARWATKA, P., KWIECIEŃ, A., RAKOWSKI, F., GRZYBOWSKA, K. (2019) *The PWA Book*, [online] Available at: <https://www.divante.com/pwabook> [Accessed 8 October 2022]
MICHÁLEK, M. (2017) *Vzhůru do CSS3*, ISBN 978-80-260-8440-2
MICHÁLEK, M. (2017) *Vzhůru (responzivního) webdesignu*, ISBN 978-80-88253-00-6

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých pojmů a zkratek

PWA – progressive web application, progresivní webová aplikace

HTML – hypertext markup language, hypertextový značkovací jazyk

CSS – cascading style sheets, kaskádové styly

JS – javascript

DOM – document object model, objektový model dokumentu

URL – uniform resource locator, jednotný lokátor zdroje

JSON – javascript object notation, textový formát pro ukládání textu

Cache – vyrovnávací paměť, ve webových technologiích slouží například k uložení CSS stylů a front-end skriptů a dokumentů

Caching – uložení aktuální verze souborů do mezipaměti

Codebase – celkový veškerý napsaný kód potřebný k běhu aplikace

Event listener – naslouchač událostí, spouští proceduru připojenou k dané události v jejím nastání, může být více naslouchačů pro jednu událost

Handler – v případě událostí v DOM zpracovávaný JS se jedná o spouštěč události v jejím nastání, může být připojena pouze jedna funkce

Přílohy

Veškerý zdrojový kód je přístupný na zkrácené adrese shorturl.at/jkqGI, respektive originální adrese https://github.com/vlisak/pwa-demo-bachelor-thesis, a audit z aplikace Lighthouse na následujících 20 stranách.

https://vlisak-pwa-demo.netlify.app/

95 100 92 PWA

95 Performance

100 Accessibility

92 Best Practices

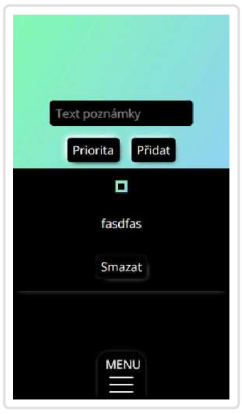
PWA

95

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100



METRICS Collapse view

■ First Contentful Paint
2.3 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.

● Time to Interactive
2.4 s
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.

● Speed Index
2.3 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.

● Total Blocking Time
0 ms
Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds. Learn more.

● Largest Contentful Paint
2.4 s
Largest Contentful Paint marks the time at which the largest text or image is painted. Learn more

● Cumulative Layout Shift
0
Cumulative Layout Shift measures the movement of visible elements within the viewport. Learn more.

View Original Trace View Treemap

Show audits relevant to: All FCP TBT LCP CLS

OPPORTUNITIES

Opportunity	Estimated Savings
Eliminate render-blocking resources	1.32 s

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more. FCP LCP

URL	Transfer Size	Potential Savings
/css2?family=Open+Sans&display=swap (fonts.googleapis.com)	0.7 KiB	760 ms
/downloads/moment.min.js (momentjs.com)	18.4 KiB	1,100 ms

These suggestions can help your page load faster. They don't directly affect the Performance score.

DIAGNOSTICS

Avoid chaining critical requests — 5 chains found

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more. FCP LCP

Maximum critical path latency: **540 ms**

Initial Navigation

- https://vlisak-pwa-demo.netlify.app
  - /css2?family=Open+Sans&display=swap (fonts.googleapis.com)
    - …v34/memSYaGs1….woff2 (fonts.gstatic.com) **- 10 ms, 11.38 KiB**
    - …v34/memSYaGs1….woff2 (fonts.gstatic.com) **- 10 ms, 8.21 KiB**
  - /style.css (vlisak-pwa-demo.netlify.app) **- 20 ms, 2.92 KiB**
  - /downloads/moment.min.js (momentjs.com) **- 310 ms, 18.41 KiB**
  - /app.js (vlisak-pwa-demo.netlify.app) **- 20 ms, 4.05 KiB**

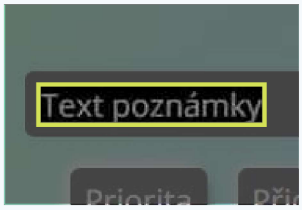
Keep request counts low and transfer sizes small — 13 requests • 71 KiB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	13	71.3 KiB
Script	3	22.6 KiB
Other	5	20.8 KiB
Font	2	19.6 KiB
Document	1	4.7 KiB
Stylesheet	2	3.6 KiB
Image	0	0.0 KiB
Media	0	0.0 KiB
Third-party	4	38.7 KiB

Largest Contentful Paint element — 1 element found

This is the largest contentful element painted within the viewport. Learn More LCP

Element	
	label

Avoid long main-thread tasks — 2 long tasks found

Lists the longest tasks on the main thread, useful for identifying worst contributors to input delay. Learn more TBT

URL	Start Time	Duration
https://vlisak-pwa-demo.netlify.app	895 ms	127 ms

URL	Start Time	Duration
https://vlisak-pwa-demo.netlify.app	752 ms	90 ms

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (35) Hide

- Properly size images

  Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

- Defer offscreen images

  Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

- Minify CSS

  Minifying CSS files can reduce network payload sizes. Learn more. FCP LCP

- Minify JavaScript

  Minifying JavaScript files can reduce payload sizes and script parse time. Learn more. FCP LCP

- Reduce unused CSS

  Reduce unused rules from stylesheets and defer CSS not used for above-the-fold content to decrease bytes consumed by network activity. Learn more. FCP LCP

- Reduce unused JavaScript

  Reduce unused JavaScript and defer loading scripts until they are required to decrease bytes consumed by network activity. Learn more. LCP

- Efficiently encode images

  Optimized images load faster and consume less cellular data. Learn more.

- Serve images in next-gen formats

  Image formats like WebP and AVIF often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

- Enable text compression

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more. FCP LCP

- Preconnect to required origins

Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn more. FCP LCP

- Initial server response time was short — Root document took 20 ms

Keep the server response time for the main document short because all other requests depend on it. Learn more. FCP LCP

URL	Time Spent
https://vlisak-pwa-demo.netlify.app	20 ms

- Avoid multiple page redirects

Redirects introduce additional delays before the page can be loaded. Learn more. FCP LCP

- Preload key requests

Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more. FCP LCP

- Use HTTP/2

HTTP/2 offers many benefits over HTTP/1.1, including binary headers and multiplexing. Learn more.

- Use video formats for animated content

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more LCP

- Remove duplicate modules in JavaScript bundles

Remove large, duplicate JavaScript modules from bundles to reduce unnecessary bytes consumed by network activity. TBT

- Avoid serving legacy JavaScript to modern browsers

Polyfills and transforms enable legacy browsers to use new JavaScript features. However, many aren't necessary for modern browsers. For your bundled JavaScript, adopt a modern script deployment strategy using module/nomodule feature detection to reduce the amount of code shipped to modern browsers, while retaining support for legacy browsers. Learn More TBT

Preload Largest Contentful Paint image

Preload the image used by the LCP element in order to improve your LCP time. Learn more. LCP

Avoids enormous network payloads — Total size was 71 KiB

Large network payloads cost users real money and are highly correlated with long load times. Learn more. LCP

Show 3rd-party resources (3)

URL	Transfer Size
/downloads/moment.min.js (momentjs.com)	18.4 KiB
…v34/memSYaGs1….woff2 (fonts.gstatic.com)	11.4 KiB
…v34/memSYaGs1….woff2 (fonts.gstatic.com)	8.2 KiB
/icon_145.png (vlisak-pwa-demo.netlify.app)	4.8 KiB
https://vlisak-pwa-demo.netlify.app	4.7 KiB
/icon_145.png (vlisak-pwa-demo.netlify.app)	4.7 KiB
https://vlisak-pwa-demo.netlify.app	4.5 KiB
/app.js (vlisak-pwa-demo.netlify.app)	4.1 KiB
/app.js (vlisak-pwa-demo.netlify.app)	4.0 KiB
/style.css (vlisak-pwa-demo.netlify.app)	2.9 KiB

Uses efficient cache policy on static assets — 1 resource found

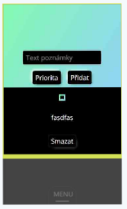
A long cache lifetime can speed up repeat visits to your page. Learn more.

URL	Cache TTL	Transfer Size

URL	Cache TTL	Transfer Size
/downloads/moment.min.js (momentjs.com)	4 h	18 KiB

● Avoids an excessive DOM size — 138 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more. TBT

Statistic	Element	Value
Total DOM Elements		138
Maximum DOM Depth	li	8
Maximum Child Elements	 body	6

○ User Timing marks and measures

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

● JavaScript execution time — 0.1 s

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more. TBT

URL	Total CPU Time	Script Evaluation	Script Parse
https://vlisak-pwa-demo.netlify.app	549 ms	45 ms	8 ms
Unattributable	196 ms	17 ms	0 ms
/app.js (vlisak-pwa-demo.netlify.app)	74 ms	48 ms	4 ms

● Minimizes main-thread work — 0.9 s

Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this. Learn more TBT

Category	Time Spent
Style & Layout	329 ms
Other	323 ms
Script Evaluation	141 ms
Parse HTML & CSS	38 ms
Script Parsing & Compilation	25 ms
Rendering	25 ms

● All text remains visible during webfont loads

Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more. FCP LCP

● Minimize third-party usage — Third-party code blocked the main thread for 0 ms

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Learn more. TBT

Third-Party	Transfer Size	Main-Thread Blocking Time
Google Fonts	20 KiB	0 ms
…v34/memSYaGs1….woff2 (fonts.gstatic.com)	11 KiB	0 ms
…v34/memSYaGs1….woff2 (fonts.gstatic.com)	8 KiB	0 ms

○ Lazy load third-party resources with facades

Some third-party embeds can be lazy loaded. Consider replacing them with a facade until they are required. Learn more. TBT

○ Largest Contentful Paint image was not lazily loaded

Above-the-fold images that are lazily loaded render later in the page lifecycle, which can delay the largest contentful paint. Learn more.

○ Avoid large layout shifts

These DOM elements contribute most to the CLS of the page. CLS

- Uses passive listeners to improve scrolling performance

Consider marking your touch and wheel event listeners as `passive` to improve your page's scroll performance. Learn more.

- Avoids `document.write()`

For users on slow connections, external scripts dynamically injected via `document.write()` can delay page load by tens of seconds. Learn more.

- Avoid non-composited animations

Animations which are not composited can be janky and increase CLS. Learn more CLS

- Image elements have explicit `width` and `height`

Set an explicit width and height on image elements to reduce layout shifts and improve CLS. Learn more CLS

- Has a `<meta name="viewport">` tag with `width` or `initial-scale`

A `<meta name="viewport">` not only optimizes your app for mobile screen sizes, but also prevents a 300 millisecond delay to user input. Learn more. TBT

- Avoids `unload` event listeners

The `unload` event does not fire reliably and listening for it can prevent browser optimizations like the Back-Forward Cache. Use `pagehide` or `visibilitychange` events instead. Learn more

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Hide

- The page has a logical tab order

Tabbing through the page follows the visual layout. Users cannot focus elements that are offscreen. Learn more.

- Interactive controls are keyboard focusable

  Custom interactive controls are keyboard focusable and display a focus indicator. Learn more.

- Interactive elements indicate their purpose and state

  Interactive elements, such as links and buttons, should indicate their state and be distinguishable from non-interactive elements. Learn more.

- The user's focus is directed to new content added to the page

  If new content, such as a dialog, is added to the page, the user's focus is directed to it. Learn more.

- User focus is not accidentally trapped in a region

  A user can tab into and out of any control or region without accidentally trapping their focus. Learn more.

- Custom controls have associated labels

  Custom interactive controls have associated labels, provided by aria-label or aria-labelledby. Learn more.

- Custom controls have ARIA roles

  Custom interactive controls have appropriate ARIA roles. Learn more.

- Visual order on the page follows DOM order

  DOM order matches the visual order, improving navigation for assistive technology. Learn more.

- Offscreen content is hidden from assistive technology

  Offscreen content is hidden with display: none or aria-hidden=true. Learn more.

- HTML5 landmark elements are used to improve navigation

  Landmark elements (<main>, <nav>, etc.) are used to improve the keyboard navigation of the page for assistive technology. Learn more.

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

PASSED AUDITS (12) Hide

- [aria-hidden="true"] is not present on the document <body>

  Assistive technologies, like screen readers, work inconsistently when `aria-hidden="true"` is set on the document `<body>`. Learn more.

- Buttons have an accessible name

  When a button doesn't have an accessible name, screen readers announce it as "button", making it unusable for users who rely on screen readers. Learn more.

- ARIA IDs are unique

  The value of an ARIA ID must be unique to prevent other instances from being overlooked by assistive technologies. Learn more.

- Image elements have [alt] attributes

  Informative elements should aim for short, descriptive alternate text. Decorative elements can be ignored with an empty alt attribute. Learn more.

- Form elements have associated labels

  Labels ensure that form controls are announced properly by assistive technologies, like screen readers. Learn more.

- [user-scalable="no"] is not used in the <meta name="viewport"> element and the [maximum-scale] attribute is not less than 5.

  Disabling zooming is problematic for users with low vision who rely on screen magnification to properly see the contents of a web page. Learn more.

- Background and foreground colors have a sufficient contrast ratio

  Low-contrast text is difficult or impossible for many users to read. Learn more.

- Document has a <title> element

  The title gives screen reader users an overview of the page, and search engine users rely on it heavily to determine if a page is relevant to their search. Learn more.

- <html> element has a [lang] attribute

  If a page doesn't specify a lang attribute, a screen reader assumes that the page is in the default language that the user chose when setting up the screen reader. If the page isn't actually in the default language, then the screen reader might not announce the page's text correctly. Learn more.

- `<html>` element has a valid value for its `[lang]` attribute

  Specifying a valid BCP 47 language helps screen readers announce text properly. Learn more.

- Lists contain only `<li>` elements and script supporting elements (`<script>` and `<template>`).

  Screen readers have a specific way of announcing lists. Ensuring proper list structure aids screen reader output. Learn more.

- List items (`<li>`) are contained within `<ul>` or `<ol>` parent elements

  Screen readers require list items (`<li>`) to be contained within a parent `<ul>` or `<ol>` to be announced properly. Learn more.

NOT APPLICABLE (32)

Hide

- `[accesskey]` values are unique

  Access keys let users quickly focus a part of the page. For proper navigation, each access key must be unique. Learn more.

- `[aria-*]` attributes match their roles

  Each ARIA `role` supports a specific subset of `aria-*` attributes. Mismatching these invalidates the `aria-*` attributes. Learn more.

- `button`, `link`, and `menuitem` elements have accessible names

  When an element doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

- `[aria-hidden="true"]` elements do not contain focusable descendents

  Focusable descendents within an `[aria-hidden="true"]` element prevent those interactive elements from being available to users of assistive technologies like screen readers. Learn more.

- ARIA input fields have accessible names

  When an input field doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

- ARIA `meter` elements have accessible names

When an element doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

ARIA `progressbar` elements have accessible names

When a `progressbar` element doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

`[role]`s have all required `[aria-*]` attributes

Some ARIA roles have required attributes that describe the state of the element to screen readers. Learn more.

Elements with an ARIA `[role]` that require children to contain a specific `[role]` have all required children.

Some ARIA parent roles must contain specific child roles to perform their intended accessibility functions. Learn more.

`[role]`s are contained by their required parent element

Some ARIA child roles must be contained by specific parent roles to properly perform their intended accessibility functions. Learn more.

`[role]` values are valid

ARIA roles must have valid values in order to perform their intended accessibility functions. Learn more.

ARIA toggle fields have accessible names

When a toggle field doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

ARIA `tooltip` elements have accessible names

When an element doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

ARIA `treeitem` elements have accessible names

When an element doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

`[aria-*]` attributes have valid values

Assistive technologies, like screen readers, can't interpret ARIA attributes with invalid values. Learn more.

`[aria-*]` attributes are valid and not misspelled

[aria-*] attributes are valid and not misspelled

Assistive technologies, like screen readers, can't interpret ARIA attributes with invalid names. Learn more.

The page contains a heading, skip link, or landmark region

Adding ways to bypass repetitive content lets keyboard users navigate the page more efficiently. Learn more.

<dl>'s contain only properly-ordered <dt> and <dd> groups, <script>, <template> or elements.

When definition lists are not properly marked up, screen readers may produce confusing or inaccurate output. Learn more.

Definition list items are wrapped in <dl> elements

Definition list items (`<dt>` and `<dd>`) must be wrapped in a parent `<dl>` element to ensure that screen readers can properly announce them. Learn more.

[id] attributes on active, focusable elements are unique

All focusable elements must have a unique `id` to ensure that they're visible to assistive technologies. Learn more.

No form fields have multiple labels

Form fields with multiple labels can be confusingly announced by assistive technologies like screen readers which use either the first, the last, or all of the labels. Learn more.

<frame> or <iframe> elements have a title

Screen reader users rely on frame titles to describe the contents of frames. Learn more.

Heading elements appear in a sequentially-descending order

Properly ordered headings that do not skip levels convey the semantic structure of the page, making it easier to navigate and understand when using assistive technologies. Learn more.

<input type="image"> elements have [alt] text

When an image is being used as an `<input>` button, providing alternative text can help screen reader users understand the purpose of the button. Learn more.

Links have a discernible name

Link text (and alternate text for images, when used as links) that is discernible, unique, and focusable improves the navigation experience for screen reader users. Learn more.

The document does not use <meta http-equiv="refresh">

The document does not use `<meta http-equiv="refresh">`

Users do not expect a page to refresh automatically, and doing so will move focus back to the top of the page. This may create a frustrating or confusing experience. Learn more.

`<object>` elements have alternate text

Screen readers cannot translate non-text content. Adding alternate text to `<object>` elements helps screen readers convey meaning to users. Learn more.

No element has a `[tabindex]` value greater than 0

A value greater than 0 implies an explicit navigation ordering. Although technically valid, this often creates frustrating experiences for users who rely on assistive technologies. Learn more.

Cells in a `<table>` element that use the `[headers]` attribute refer to table cells within the same table.

Screen readers have features to make navigating tables easier. Ensuring `<td>` cells using the `[headers]` attribute only refer to other cells in the same table may improve the experience for screen reader users. Learn more.

`<th>` elements and elements with `[role="columnheader"/"rowheader"]` have data cells they describe.

Screen readers have features to make navigating tables easier. Ensuring table headers always refer to some set of cells may improve the experience for screen reader users. Learn more.

`[lang]` attributes have a valid value

Specifying a valid BCP 47 language on elements helps ensure that text is pronounced correctly by a screen reader. Learn more.

`<video>` elements contain a `<track>` element with `[kind="captions"]`

When a video provides a caption it is easier for deaf and hearing impaired users to access its information. Learn more.

Best Practices

USER EXPERIENCE

Displays images with incorrect aspect ratio

Image display dimensions should match natural aspect ratio. Learn more.

	URL	Aspect Ratio (Displayed)	Aspect Ratio (Actual)
img	…	20 x 20 (1.00)	57 x 44 (1.30)

TRUST AND SAFETY

Ensure CSP is effective against XSS attacks

A strong Content Security Policy (CSP) significantly reduces the risk of cross-site scripting (XSS) attacks. Learn more

Description	Directive	Severity
No CSP found in enforcement mode		High

GENERAL

Detected JavaScript libraries

All front-end JavaScript libraries detected on the page. Learn more.

Name	Version
Moment.js	2.29.4

PASSED AUDITS (12) Hide

Uses HTTPS

All sites should be protected with HTTPS, even ones that don't handle sensitive data. This includes avoiding mixed content, where some resources are loaded over HTTP despite the initial request being served over HTTPS. HTTPS prevents intruders from tampering with or passively listening in on the communications between your app and your users, and is a prerequisite for HTTP/2 and many new web platform APIs. Learn more.

Avoids requesting the geolocation permission on page load

Users are mistrustful of or confused by sites that request their location without context. Consider tying the request to a user action instead. Learn more.

- Avoids requesting the notification permission on page load

Users are mistrustful of or confused by sites that request to send notifications without context. Consider tying the request to user gestures instead. Learn more.

- Avoids front-end JavaScript libraries with known security vulnerabilities

Some third-party scripts may contain known security vulnerabilities that are easily identified and exploited by attackers. Learn more.

- Allows users to paste into password fields

Preventing password pasting undermines good security policy. Learn more.

- Serves images with appropriate resolution

Image natural dimensions should be proportional to the display size and the pixel ratio to maximize image clarity. Learn more.

- Page has the HTML doctype

Specifying a doctype prevents the browser from switching to quirks-mode. Learn more.

- Properly defines charset

A character encoding declaration is required. It can be done with a `<meta>` tag in the first 1024 bytes of the HTML or in the Content-Type HTTP response header. Learn more.

- Avoids deprecated APIs

Deprecated APIs will eventually be removed from the browser. Learn more.

- No browser errors logged to the console

Errors logged to the console indicate unresolved problems. They can come from network request failures and other browser concerns. Learn more

- No issues in the `Issues` panel in Chrome Devtools

Issues logged to the `Issues` panel in Chrome Devtools indicate unresolved problems. They can come from network request failures, insufficient security controls, and other browser concerns. Open up the Issues panel in Chrome DevTools for more details on each issue.

Page has valid source maps

Source maps translate minified code to the original source code. This helps developers debug in production. In addition, Lighthouse is able to provide further insights. Consider deploying source maps to take advantage of these benefits. Learn more.

Show 3rd-party resources (1)

URL	Map URL
/downloads/moment.min.js (momentjs.com)	/downloads/moment.min.js.map (momentjs.com)
`Error: Failed fetching source map (404)`	

NOT APPLICABLE (1) Hide

Fonts with `font-display: optional` are preloaded

Preload `optional` fonts so first-time visitors may use them. Learn more

PWA

These checks validate the aspects of a Progressive Web App. Learn more.

INSTALLABLE

Web app manifest and service worker meet the installability requirements

Service worker is the technology that enables your app to use many Progressive Web App features, such as offline, add to homescreen, and push notifications. With proper service worker and manifest implementations, browsers can proactively prompt users to add your app to their homescreen, which can lead to higher engagement. Learn more.

PWA OPTIMIZED

Registers a service worker that controls page and `start_url`

The service worker is the technology that enables your app to use many Progressive Web App features, such as offline, add to homescreen, and push notifications. Learn more.

▲ Is not configured for a custom splash screen

Failures: Manifest does not have a PNG icon of at least 512px, Manifest does not have `background_color`, Manifest does not have `theme_color`.

A themed splash screen ensures a high-quality experience when users launch your app from their homescreens. Learn more.

▲ Does not set a theme color for the address bar.

Failures: Manifest does not have `theme_color`, No `<meta name="theme-color">` tag found.

The browser address bar can be themed to match your site. Learn more.

● Content is sized correctly for the viewport

If the width of your app's content doesn't match the width of the viewport, your app might not be optimized for mobile screens. Learn more.

● Has a `<meta name="viewport">` tag with `width` or `initial-scale`

A `<meta name="viewport">` not only optimizes your app for mobile screen sizes, but also prevents a 300 millisecond delay to user input. Learn more. TBT

▲ Does not provide a valid `apple-touch-icon`

For ideal appearance on iOS when users add a progressive web app to the home screen, define an `apple-touch-icon`. It must point to a non-transparent 192px (or 180px) square PNG. Learn More.

▲ Manifest doesn't have a maskable icon

A maskable icon ensures that the image fills the entire shape without being letterboxed when installing the app on a device. Learn more.

ADDITIONAL ITEMS TO MANUALLY CHECK (3)

Hide

○ Site works cross-browser

To reach the most number of users, sites should work across every major browser. Learn more.

○ Page transitions don't feel like they block on the network

Transitions should feel snappy as you tap around, even on a slow network. This experience is key to a user's perception of performance. Learn more.

○ Each page has a URL

Ensure individual pages are deep linkable via URL and that URLs are unique for the purpose of shareability on social media. Learn more.

These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

Captured at Oct 15, 2022, 12:06 PM GMT+2
Initial page load

Emulated Moto G4 with Lighthouse 9.6.2
Slow 4G throttling

Single page load

Using Chromium 106.0.0.0 with devtools

Generated by **Lighthouse** 9.6.2 | File an issue