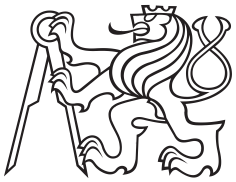


České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Srovnání technologií pro tvorbu univerzálních aplikací

Bedřich Schindler

Vedoucí práce: RNDr. Ondřej Žára
Srpen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Schindler** Jméno: **Bedřich** Osobní číslo: **465937**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Srovnání technologií pro tvorbu univerzálních aplikací

Název bakalářské práce anglicky:

Universal app technology comparison

Pokyny pro vypracování:

Porovnejte dostupné technologie pro tvorbu univerzálních (web + nativní) aplikací: PWA, Electron, Hybrid. Uvažte podporu jak plnohodnotných (desktopových) počítačů, tak mobilních zařízení na platformách iOS a Android.
Kromě technologického aspektu srovnejte též možnost publikace výsledných aplikací.
Vyberte technologii, která by byla nejvhodnější pro následnou implementaci aplikace pro evidenci účetních a provozních dat. Svůj výběr dobře odůvodněte a aplikaci sestrojte. Hlavní uživatelské scénáře budou:
- Evidence klientů a projektů
- Evidence odvedené práce a správa souvisejících faktur
- Evidence placených záloh na dani, zdravotního a sociálního pojištění
- Zobrazení přehledů těchto dat
- Upozornění na různé časové limity vycházející z výše uvedených položek

Nad výslednou aplikací proveďte kvalitativní uživatelské testování a porovnejte ji s existujícími řešeními. Popište způsob zabezpečení uživatelských dat a technologické limity zvolené aplikační platformy, na které aplikace naráží.

Seznam doporučené literatury:

Dean Alan Hume: Progressive Web Apps, Manning Publications 2017, ISBN 1617294586
Steve Kinney: Electron in Action, Manning Publications 2018, ISBN 9781617294143
https://developer.mozilla.org/en-US/docs/Web

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ondřej Žára, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **30.09.2021**

RNDr. Ondřej Žára podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Rád bych poděkoval vedoucímu práce RNDr. Ondřeji Žárovi za pomoc při sestavování specifikace bakalářské práce, její implementaci a zpětnou vazbu v jejím průběhu .

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu a zdroje, a to v souladu s Metodickým pokynem č. 1/2009 o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 1. sprna 2020

Abstrakt

Cílem této bakalářské práce je porovnání dostupných technologií pro tvorbu multiplatformních aplikací za pomoci webových technologií se zaměřením na tři konkrétní technologie, a to na progresivní webové aplikace, Electron a hybridní aplikace. V případě nativních aplikací bude uvažována jak podpora plnohodnotných desktopových systémů, tak podpora mobilních zařízení využívající systémy Android, iOS a iPadOS.

Na základě zjištěných skutečností bude zvolena nejvhodnější technologie pro aplikaci k evidenci účetních a provozních dat osob samostatně výdělečně činných, ve které bude aplikace implementována.

Práce je určena především pro vývojáře, jež znají technologie HTML, CSS a především JavaScript, jeho syntax a základní API.

Klíčová slova: multiplatformní aplikace, progresivní webové aplikace, PWA, Electron, hybridní aplikace

Vedoucí práce: RNDr. Ondřej Žára

Abstract

The aim of this bachelor thesis is to compare technologies for multi-platform application development like progressive web applications, Electron and hybrid applications. In case of native applications, full-fledged desktop operating systems and mobile operating systems Android, iOS and iPadOS are considered.

Based on the findings, the most suitable technology will be selected for the application managing accounting and operating data of self-employed persons, in which the application will be implemented.

The thesis aims to web developers familiar with HTML, CSS and especially JavaScript, its syntax and basic API.

Keywords: multi-platform applications, progressive web applications, PWA, Electron, hybrid applications

Title translation: Universal application technology comparison

Obsah

Obrázky

Kapitola 1
Úvod

1.1 Motivace

V poslední době jsem se v oblasti vývoje webových aplikací začal čím dál tím více setkávat s dotazy a požadavky klientů na možnosti využití jejich současné webové aplikace a jejího transformování do desktopové nebo mobilní verze. Klienti se brání příliš velkým investicím do vývoje, neboť musejí podobnou aplikaci zaplatit několikrát. Nejprve jako aplikaci webovou, podruhé jako aplikaci desktopovou a dále jako aplikaci mobilní, a to pro systém Android a systémy iOS a iPadOS.

Výše zmiňovaný problém má však řadu možných řešení a ušetří nejenom peníze na vývoji několika aplikací, ale také potřebu znát a ovládat několik programovacích jazyků a technologií současně. A právě o možných řešeních tohoto problému je tato práce.

1.2 Možnosti řešení

Tato práce je úzce spjata s webovými technologiemi HTML, CSS a JavaScript, jež pro nás budou základním stavebním kamenem, od kterého se bude práce odvíjet, přičemž se od čtenáře této práce předpokládá elementární znalost těchto zmíněných technologií.

V současné době se vymýšlejí stále nová řešení, pomocí kterých můžeme využívat webové technologie pro vývoj multiplatformních aplikací, takže není možné v práci obsáhnout všechna řešení. Můžeme však vybrat ty nejrozšířenější technologie s největším potenciálem a o těch se v práci zmínit. První z nich jsou progresivní webové aplikace, druhou z nich je Electron a poslední z nich jsou hybridní aplikace.

1.3 Cíl

Cílem práce je seznámit vás s výše uvedenými technologiemi, porovnat je mezi sebou a vybrat vhodnou technologii pro implementaci aplikace pro evidenci účetních a provozních dat. Aplikace bude implementována jako webová aplikace, a poté využita v rámci zvolené technologie.

Popis implementace, omezení pro jednotlivé platformy a výsledky uživatelského testování dané aplikace, budou v práci taktéž popsány.

Kapitola 2
Progresivní webové aplikace

2.1 Definice

Progresivní webové aplikace [3], zkráceně označovány jako PWA, jsou aplikace založené na webových technologiích HTML, CSS a JavaScriptu. V základu se jedná o běžnou webovou aplikaci, která je navíc instalovatelná na mobilní i desktopové systémy. Proto, aby mohla být aplikace prohlášena za PWA, musí splňovat následující kritéria [4].

2.2 Kritéria

2.2.1 Nalezitelná (Discoverable)

Aplikace je snadněji nalezitelná pomocí vyhledávačů, a to především díky speciálnímu Web App Manifestu [5, 6], popisujícího základní informace o aplikaci a o jejím chování. Díky meta informacím je jakožto webová stránka lépe srozumitelná pro vyhledávací algoritmy a vyhledávače by tak měly poskytovat relevantnější výsledky vyhledávání a pomáhat tak uživatelům dané aplikace snáze vyhledat.

2.2.2 Odkazovatelná (Linkable)

Aplikace je dosažitelná pomocí pouhé URL a není tak nutné provádět její vyhledávání a instalaci přes specializované obchody s aplikacemi jako je AppStore či GooglePlay.

2.2.3 Instalovatelná (Installable)

Aplikaci je možno nainstalovat na plochu zařízení pouhým odesláním aplikace na plochu zařízení, tedy bez zbytečného stahování a instalování přes specializované obchody s aplikacemi jako je AppStore či GooglePlay, jako je to u běžných nativních aplikací.

2.2.4 Progresivní (Progressive)

Aplikaci je možné spustit na libovolném zařízení v libovolném prohlížeči a splňuje přístup postupného zlepšování (anglicky Progressive enhancement)[7].

2.2.5 Responzivní (Responsive)

Aplikace se přizpůsobuje různým rozlišením a zobrazuje se správně na desktopu, tabletu i telefonu. Využívá pro to Media Queries a Viewport.

2.2.6 Bezpečná (Safe)

Aplikace klade důraz na bezpečnost a je servírována pomocí protokolu HTTPS.

2.2.7 Nezávislá na konektivitě (Network independent)

Aplikace je nezávislá na konektivitě. Aplikace a její stránky správně fungují i bez ohledu na pomalé nebo žádné připojení, přičemž informace o žádném připojení jsou patřičně zobrazeny uživateli.

K dosažení nezávislosti na konektivitě se využívá služeb běžících na pozadí (anglicky Service Workers)[8], jež se využívá například k obsluze síťových požadavků v režimu offline v kombinaci s Cache API [9] pro jejich ukládání. Pro ukládání dat lze využít služeb Web Storage či IndexedDB.

2.2.8 Znovuzapojující uživatele (Re-engageable)

Aplikace se snaží aktualizacemi obsahu uživatele znovuzapojit do používání aplikace, a to například za pomoci služeb běžících na pozadí, služby Notifications API [10], která zajišťuje notifikování uživatele anebo pomocí Web Push API, která slouží k notifikování uživatele přímo ze serveru, a to i pokud není aplikace právě aktivní.

2.3 Implementace

Pro hlubší pochopení progresivních webových aplikací je nejlepší ukázat velmi primitivní strukturu ukázkové aplikace, díky níž je možné snáze pochopit, jak PWA aplikace může vypadat a co je nezbytné minimum, které musí splňovat.

Budeme uvažovat následující strukturu:

- index.html
- app.css
- app.js
- sw.js
- .webmanifest

- icons *(složka s ikonami)*

V následujících podkapitolách nastíním význam jednotlivých souborů a význam jednotlivých meta dat a konstruktů, jež jednotlivé soubory obsahují.

2.3.1 Soubor .webmanifest (Web App Manifest)

Vstupním souborem aplikace je sice soubor index.html, ale jeho hlavička obsahuje některé z parametrů obsažených právě v souboru .webmanifest, takže účel tohoto souboru představím nejdříve.

Soubor .webmanifest, někdy také manifest.json, je soubor ve formátu JSON obsahující výčet informací specifikující informace o aplikaci a jejím chování. Uvedené informace jsou využívány jak aplikací, kterou je možné nainstalovat na plochu zařízení, tak například i prohlížečem nebo vyhledávacími algoritmy.

Níže popíši význam a použití jednotlivých atributů:

- **name** - Název aplikace zobrazovaný při instalaci aplikace a následně i u aplikace uložené na ploše uživatele.
- **short_name** - Krátký název aplikace zobrazovaný ve stejných případech jako parameter `name`, avšak používaný v těch případech, kdy není na obrazovce dostatek místa. Často bývá použit jako název aplikace na ploše zařízení právě místo parametru `name`.
- **start_url** - Absolutní nebo relativní URL, ze které má být aplikace načtena při svém spuštění.
- **display** - Mód zobrazení aplikace určující vzhled aplikace a jeho kontrolní prvky. Mód zobrazení může nabývat následujících hodnot, jež jsou však orientační, neboť si je každý prohlížeč implementuje svým způsobem.
  - **browser** - Standardní zobrazení aplikace využívající všechny prvky webového prohlížeče.
  - **minimal-ui** - Zobrazení aplikace využívající typicky pouze prvky webového prohlížeče pro navigaci.
  - **standalone** - Obdobné zobrazení jako v předchozím případě, typicky obsahující záhlaví se jménem aplikace a případně jeho ikonou.
  - **fullscreen** - Zobrazení aplikace překrývající celou část obrazovky zařízení.
- **background_color** - Barva pozadí aplikace využívající se při načítání aplikace.
- **theme_color** - Primární barva aplikace využívaná například v záhlaví a stavové liště aplikace.
- **orientation** - Určuje orientaci aplikace, kterou má aplikace využívat pro svoje zobrazení. Stejně jako u atributu `display` je i tento atribut implementován v každém prohlížeči svým způsobem, případně ještě není

implementován vůbec. Orientačně však může nabývat kupříkladu hodnot `any`, `natural`, `landscape` či `portrait`.

- **icons** - Obsahuje seznam ikon v různých velikostech. Vhodné je definovat hned několik různých velikostí, aby si prohlížeč mohl vybrat vyhovující rozlišení. Tyto ikony jsou používány jako ikony nainstalované aplikace nebo na načítací obrazovce aplikace (anglicky Splash Screen).

Podpora výše uvedených atributů a jejich chování se liší v závislosti na operačním systému a prohlížeči, takže je potřeba vždy zjistit aktuální podporu jednotlivých atributů napříč operačními systémy a prohlížeči, a také počítat s tím, že ne ve všech případech dosáhnete tíženého výsledku. Nutno podotknout, že se situace postupně zlepšuje. Některé prohlížeče zároveň využívají obdobných parametrů v hlavičce souboru index.html, takže je podobné atributy nezbytné definovat i tam.

Pro shrnutí informací o Web App Manifestu ještě přikládám ukázku reálného použití:

```
{
  "name": "Finance management",
  "short_name": "Finances",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#fafafa",
  "theme_color": "#009688",
  "orientation": "portrait",
  "icons": [
    {
      "src": "\/icons\/android-icon-36x36.png",
      "sizes": "36x36",
      "type": "image\/png",
      "density": "0.75"
    },
    // Další definice ikon
  ]
}
```

2.3.2 Soubor index.html

Definice souboru index.html se nijak neliší od HTML souboru webových stránek, které známe. Pouze pro potřeby progresivních webových aplikací musíme specifikovat některé věci, se kterými se na běžných webových stránkách nepotkáme. Nejprve přiložím ukázkový soubor index.html z reálné aplikace, a poté zmíním některé části specifické pro PWA.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1"
    >
    <title>Finance management</title>
    <meta
      name="description"
      content="Aplikace pro správu činností OSVČ"
    >
    <meta name="author" content="Bedřich Schindler">
    <meta
      name="apple-mobile-web-app-title"
      content="eOSVČ"
    >
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="theme-color" content="#009688">
    <link rel="manifest" href="/.webmanifest">
    <link
      rel="apple-touch-icon"
      sizes="57x57"
      href="/icons/apple-icon-57x57.png"
    >
    <!-- Další větší ikony pro Apple -->
    <link
      rel="icon shortcut"
      type="image/png"
      sizes="32x32"
      href="/icons/android-icon-32x32.png"
    >
    <!-- Další větší ikony -->
    <link rel="stylesheet" href="./app.css">
    <script src="/app.js" defer></script>
    </head>
    <body>
      <main id="content" class="content">
        <noscript>
          Enable JavaScript to run this application.
        </noscript>
      </div>
    </body>
</html>
```

Jak si můžeme všimnout, tak ke standardním definicím v záhlaví HTML souboru jako je kódování, titulek stránky, popis stránky, definice stylů nebo odkaz na JavaScriptový soubor, přibyly další definice.

Nejpodstatnější je `<link rel="manifest" href="/.webmanifest" >`, jež určuje cestu k Web App Manifestu zmiňovaném v předchozí podkapitole a `<script src="/app.js" defer></script>`, která určuje cestu k hlavnímu JavaScriptovému souboru app.js, který v naší PWA aplikaci bude hrát důležitou roli, o které se zmíním v následující kapitole. JavaScriptový soubor sw.js, který jsem uvedl ve struktuře aplikace, z definic v hlavičce tohoto souboru záměrně vynechávám, taktéž se o něj zmíním později.

Kromě výše zmíněných je tu řada další definic, především pro zařízení Apple [11], které si přiblížíme detailněji. Ty, u kterých existuje ekvivalent ve Web App Manifestu tuto skutečnost uvedeme. V seznamu uvedeme některé definice, které v ukázkovém souboru index.html nenalezneme.

- **apple-mobile-web-app-capable** - Na zařízeních Apple určuje, zda má být aplikace zobrazena v režimu přes celou obrazovku či nikoliv. Nejnovější verze prohlížeče Safari využívají již atribut `display` z Web App Manifestu, ale pro zpětnou kompatibilitu se doporučuje ho stále nastavovat.
- **apple-mobile-web-app-title** - Na zařízeních Apple určuje název aplikace zobrazovaný při instalaci aplikace a následně i u aplikace uložené na ploše uživatele. Nejnovější verze prohlížeče Safari využívají již atribut `short_name` z Web App Manifestu.
- **apple-touch-icon** - Na zařízeních společnosti Apple definuje seznam ikon v různých velikostech. Vhodné je definovat hned několik různých velikostí podle velikostí zařízení společnosti Apple, aby si prohlížeč mohl vybrat vyhovující rozlišení.
- **apple-touch-startup-image** - Na zařízeních společnosti Apple definuje seznam obrázků, které budou použity jako načítací obrazovky aplikace (anglicky Splash Screen). Tyto obrázky musí odpovídat přesným rozměrům jednotlivých zařízení.
- **apple-mobile-web-app-status-bar-style** - Na zařízeních společnosti Apple nastavuje styl stavové lišty zařízení na bílé pozadí s černým textem (`default`), černé pozadí s bílým textem (`black`) nebo na bílý text s barvou pozadí HTML elementu `body` (`black-translucent`).
- **theme-color** - Nastavuje barvu stavové lišty zařízení. Většina prohlížečů (kromě Safari) využívá již atribut `theme_color` z Web App Manifestu.

2.3.3 Soubor app.css

U stylů není vyžadována žádná zvláštní implementace, pouze to, aby byla aplikace responzivní, čehož lze dosáhnout pomocí Media Queries, což je však v současné době u nově vyvíjených aplikací samozřejmostí.

2.3.4 Soubor app.js

Soubor app.js je v našem příkladu hlavním JavaScriptovým souborem. Jeho úkoly je kromě vytváření obsahu aplikace, registrace souboru sw.js jako služby běžící na pozadí a vyžádání oprávnění pro přístup k notifikacím.

Registraci souboru sw.js jako služby běžící na pozadí provedeme následujícím způsobem:

```
if('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js');
};
```

Vyžádání oprávnění pro přístup k notifikacím provedeme následujícím způsobem:

```
Notification.requestPermission().then(function(result) {
  if (result === 'default') {
    console.error('Žádost o povolení notifikací byla zrušena.');
    return;
  }

  if (result === 'denied') {
    console.error('Žádost o povolení notifikací byla zamítnuta.');
    return;
  }

  console.log('Žádost o povolení notifikací byla schválena.');
});
```

Detekce stavu přípojení k internetu a detekce změn připojení k internetu se provede následujícím způsobem:

```
// Stav připojení k internetu
const isOnline = navigator.onLine;

//Detekce změn přípojení k internetu
const detectOnlineStatus = (event) => {
  if (navigator.onLine) {
    console.log('Připojení k internetu bylo navázáno.');
  } else {
    console.log('Připojení k internetu bylo ztraceno.');
  }
}

window.addEventListener('online',  detectOnlineStatus);
window.addEventListener('offline', detectOnlineStatus);
```

Dále by následoval samotný kód aplikace, který by kupříkladu využíval vytváření notifikací pomocí Notifications API či Push API, získávání dat pomocí Fetch API či jejich ukládání pomocí Web Storage API, ale to není náplní této ukázky. Ta slouží pouze pro demonstraci základního nastavení aplikace pro splňování podmínek PWA a ukázek nástrojů, které je pro to možné využít.

2.3.5 Soubor sw.js (Service worker)

Podpora služby běžící na pozadí (anglicky Service Worker) je pro PWA jedna z klíčových záležitostí, neboť by bez ni nebylo možné zajistit nezávislost na konektivitě. Uvnitř služby běžící na pozadí totiž využijeme Cache API, abychom uložili do mezipaměti prohlížeče potřebné soubory a mohli je tak využít i v případě, kdy nemáme k dispozici připojení k internetu.

Při instalaci služby uložíme pomocí Cache API všechny soubory, které chceme mít dostupné, i když nemáme k dispozici připojení k internetu.

```
const cacheName = 'finance-management';
const cacheVersion = 'v1.0.0';
const cacheFiles = [
  '/',
  '/index.html',
  '/app.css',
  '/app.js',
  '/icons/android-icon-36x36.png',
  // Zde následuje výpis dalších cest k ikonám
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(`${cacheName}-${cacheVersion}`).then(
        (cache) => cache.addAll(cacheFiles)
    )
  );
});
```

Při následném síťovém požadavku na soubory uložené v mezipaměti dojde díky následujícímu konstruktu k jejich načtení z mezipaměti a umožní tak fungování aplikace v režimu bez připojení k internetu. Pokud není odpověď na požadavek uložena v mezipaměti, tak je tam přidána.

```
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((result) => {
      if (result) {
        return result;
      }
```

```
      return fetch(event.request).then((response) =>
        caches.open(`${cacheName}-${cacheVersion}`)
          .then((cache) => {
            cache.put(event.request, response.clone());

            return response;
          });
      );
    })
  );
});
```

Výše uvedený kód nám sice ukládá a načítá soubory uložené v mezipaměti, ale jakmile je jednou uloží, tak je už vždy načítá z mezipaměti. Aby nám soubory ze staré mezipaměti nenačítal, provedeme změnu proměnné určující verzi cache:

```
const cacheVersion = 'v2.0.0';
```

Když změníme proměnnou s verzí dat uložených v mezipaměti, tak se při dalším zavolání služby na pozadí zaregistruje služba na pozadí s novou verzí a ta stará se přestane využívat.

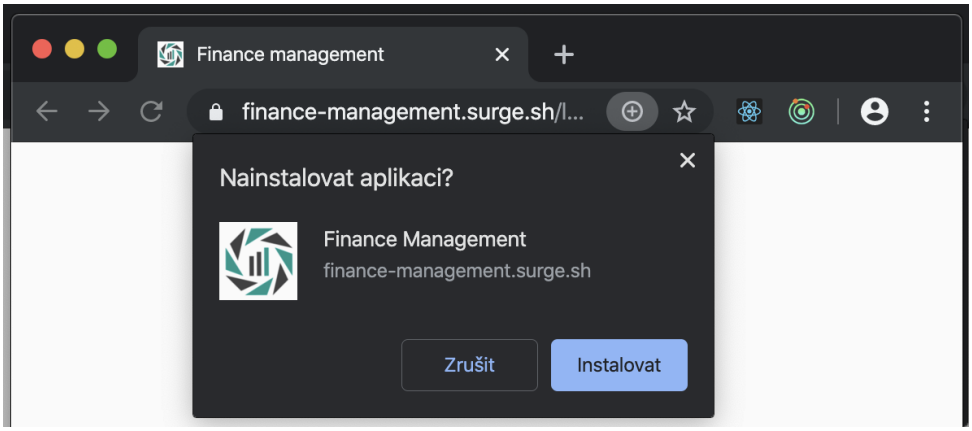
Výše uvedené řešení je sice vhodné pro ukládání všech požadavků a tedy i souborů, které naše aplikace provede, ale na druhou stranu můžeme chtít ukládat do mezipaměti pouze nějaké soubory a zbytek požadavků ukládat do mezipaměti pouze v případě potřeby. Toho můžeme docílit detekcí metody požadavku, typu vráceného souboru či podle předem připraveného seznamu v kombinaci s detekcí stavu připojení k internetu. To však záleží na vaší potřebě, a tudíž je zbytečné uvádět několik různých příkladů.

2.4 Podpora

2.4.1 Desktop

Jediný prohlížeč, který podporuje nainstalování PWA aplikace na desktopové zařízení, je prohlížeč Google Chrome, a to na všech hlavních operačních systémech Windows, MacOS, Linux a Chrome OS.

Pokud je webová aplikace zároveň PWA aplikací, tak se ve stavové liště prohlížeče zobrazí ikona plus s případným nápisem Instalovat, pomocí které můžete aplikaci nainstalovat na plochu.

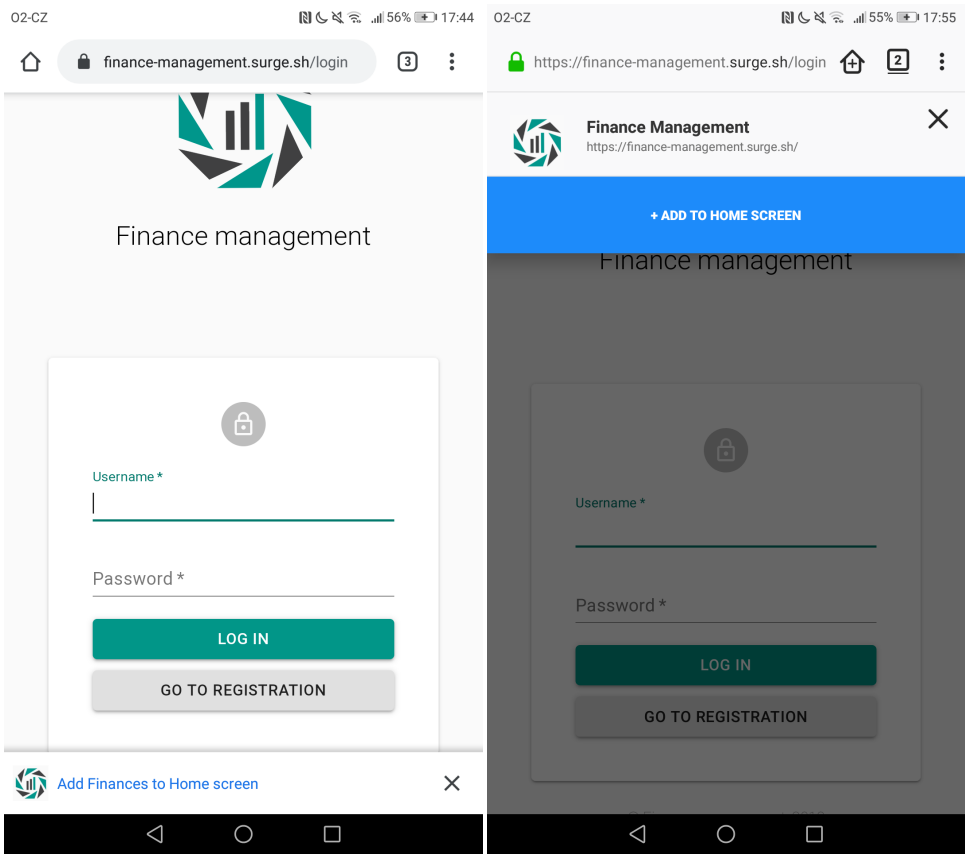


Obrázek 2.1: Instalace PWA aplikace v desktopovém prohlížeči Chrome

2.4.2 Mobilní zařízení se systémem Android

Na systému Android PWA aplikaci nainstalujete například prostřednictvím prohlížeče Google Chrome, který po vstupu na stránku zobrazí instalační banner s tlačítkem Přidat na obrazovku (anglicky Add to home screen, zkráceně A2HS)[12]. Obdobná situace je ve Firefoxu, kde se instalační banner zobrazí po stisknutí tlačítka domečku se znamínkem plus nebo pomocí prohlížeče Opera, kde instalaci provedete pomocí stisknutí tlačítka se znamínkem plus. U dalších méně používaných prohlížečů se podpora liší a je třeba třeba si pro každý takový prohlížeč ověřit, jestli je nebo není možné využít tento prohlížeč pro instalaci PWA.

Instalovatelnost aplikace je sice základním předpokladem PWA aplikace, ale není jediným podstatným ukazatelem. Dalším podstatným ukazatelem pro nás může být podpora jednotlivých části Web Manifest API, a to podpora jednotlivých atributů uvedených ve Web App Manifestu nebo například podpora událostí `onbeforeinstallprompt` či `appinstalled`. Zrovna tyto události jsou podporované pouze v prohlížeči Google Chrome. Další podstatným API pro nás může být například Notifications API nebo Push API, avšak i tyto API mají v dnešní době podporu už i u většiny méně používaných prohlížečů.

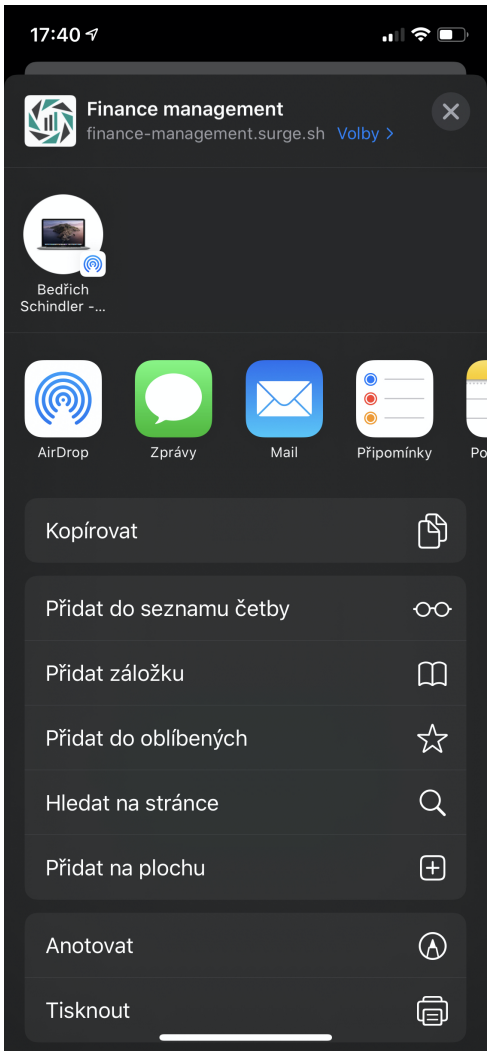


Obrázek 2.2: Instalace PWA aplikace v mobilních prohlížečích Chrome a Firefox

2.4.3 Mobilní zařízení se systémy iOS a iPadOS

Horší podpora je však na mobilních zařízeních se systémy iOS a iPadOS. Jediný prohlížeč s částečnou podporou PWA je Safari. Instalace aplikace probíhá stisknutím tlačítka Sdílet a následným kliknutím na tlačítko Přidat na plochu. Ostatní prohlížeče v rámci iOS a iPadOS instalaci PWA aplikace na plochu nepodporují.

Stejně jako většina prohlížečů na Androidu, Safari taktéž nepodporuje události `onbeforeinstallprompt` či `appinstalled`, jež jsou součásti Web Manifest API. Některé prvky Web App Manifestu, jak bylo možné vidět v rozboru jednotlivých souborů, jsou uvedeny pro správnou funkčnost i v souboru index.html. Bohužel Safari v současné době ještě stále nepodporuje ani Notifications API nebo Push API, takže používání notifikací na mobilních zařízeních společnosti Apple není možné.



Obrázek 2.3: Instalace PWA aplikace v mobilním prohlížeči Safari

Kapitola 3
Electron

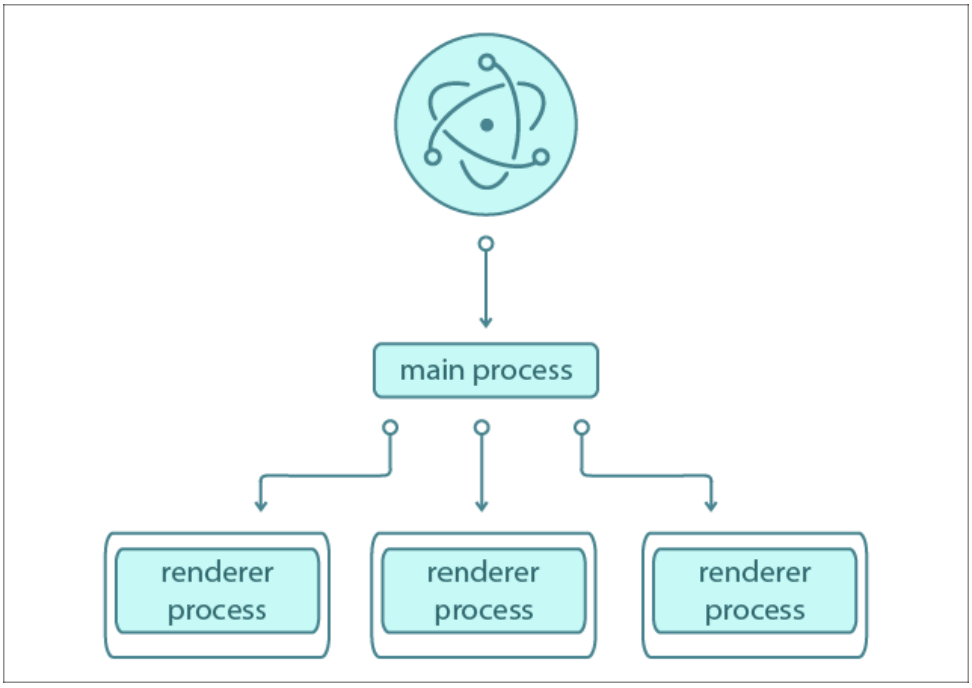
3.1 Definice

Electron [13] je JavaScriptová knihovna sloužící pro vývoj multiplatformních desktopových aplikací pomocí webových technologií HTML, CSS a JavaScriptu. Oproti PWA se Electron zaměřuje výhradně na vývoj desktopových aplikací pro operační systémy MacOS, Linux a Windows. Electron nevyužívá nainstalovaný prohlížeč uživatele jako tomu je u PWA, ale využívá vlastní upravenou distribuci prohlížeče Chromium, a to společně s Node.js.

3.2 Princip

Při spuštění aplikace napsané v Electronu je spuštěn hlavní proces (anglicky Main Process) [1], respektive hlavní JavaScriptový soubor, který se stará o běh aplikace, komunikaci s operačním systémem a jeho grafickým rozhraním a správu jednotlivých oken.

Otevření nového okna znamená spuštění nového vykreslovacího procesu (anglicky Renderer Process) [1], který se již stará o vykreslení konkrétního okna pomocí HTML, CSS a JavaScriptu. Vykreslovacích procesů, tedy oken aplikace, může běžet několik najednou, všechna jsou na sobě nezávislá, hlavní proces však může spolu s jednotlivými vykreslovacími procesy komunikovat.



Obrázek 3.1: Princip Electronu [převzato z webu medium.com][1]

Bezespornými výhodami oproti PWA je možnost využívání Node.js a všech jeho knihoven, přístup k řadě nativních součástí systému jako jsou notifikace, hlavní lišta s aplikacemi, oznamovací oblast a mnoho dalších API, která bychom ve standardních prohlížečích těžko hledaly.

Nevýhodou oproti PWA je nutnost instalace běhového prostředí Node.js a vlastní distribuce samotné aplikace, neboť nic takového jako tlačítko s nápisem Přidat na plochu se při navštívení webové stránky v případě aplikace napsané v Electronu nezobrazí. Výsledná aplikace napsaná v Electronu je totiž standardní spustitelný soubor a je nutné ji distribuovat vlastní cestou. Další nevýhodou je složitější prostředí pro vývoj, sestavení a distribuci aplikace, což je zapříčiněno požadavkem na lokální instalaci prostředí Node.js. V dnešní době samozřejmě webové aplikace i PWA aplikace využívají složitějších prostředí pro vývoj, neboť se používá řada nástrojů pro transpilování JavaScriptu, kontrolu syntaxe a podobně, ale v základu nic takového, na rozdíl od Electronu, není vyžadováno.

V následujících podkapitolách se seznámíme s ukázkovou strukturou aplikace, jejími soubory a ukážeme si práci s některými API, které Electron nabízí.

3.3 Implementace

Pro hlubší pochopení Electronu je nejlepší ukázat velmi primitivní strukturu ukázkové aplikace, díky níž je možné snáze pochopit, jak aplikace napsaná v Electronu může vypadat a co je nezbytné minimum, které musí splňovat.

V této kapitole je předpokladem lokálně nainstalované prostředí Node.js. Budeme uvažovat následující strukturu:

- package.json
- index.js
- app.html
- app.js
- app.css

V následujících podkapitolách nastíníme význam jednotlivých souborů a význam jednotlivých meta dat a konstruktů, jež jednotlivé soubory obsahují a uvedeme některá výběrová API, která nám běžné prohlížeče neposkytují.

3.3.1 Soubor package.json

Soubor package.json [14] je vstupní bod aplikace, který mimo jiné definuje informace o aplikaci jako je jeho název, popis, verze, licence, informace o prostředích, pro kterou je aplikace kompilována nebo seznam všech závislostí, které aplikace potřebuje pro svůj běh nebo vývoj. Hlavně ale definuje cestu k souboru, který se spustí jakožto dříve zmiňovaný hlavní proces.

Níže popíši význam a použití jednotlivých atributů:

- **name** - Vývojový název aplikace, který je psán pomocí zápisu zvanému kebab-case a používán pouze při vývoji aplikace. Nejedná se o název aplikace, který se nám zobrazuje u spustitelného souboru.
- **version** - Verze aplikace.
- **homepage** - URL adresa odkazující na hlavní stránku aplikace.
- **repository** - URL adresa odkazující na hlavní stránku repozitáře zdrojových kódů.
- **main** - Cesta k hlavnímu JavaScriptovému souboru, který je spuštěn jako hlavní proces aplikace.
- **build** - Informace používající se při sestavování spustitelného souboru.
  - **appId** - Standardizovaný identifikátor aplikace používaný při instalaci a registraci do operačního systému. Obvykle se píše ve tvaru `com.electron.appName`, kde `appName` je systémové jméno vaší aplikace, obvykle využívající zápis snakeCase.
  - **productName** - Název aplikace.
  - **copyright** - Text obsahující krátký text o právech k aplikaci, často obsahuje rok vytvoření a vlastníka a je zobrazován v podrobnostech o aplikaci.

- **scripts** - Seznam příkazů, které se používají při vývoji, testování a sestavení aplikace. Ve své podstatě slouží pro odstínění složitých příkazů, které uvnitř jednotlivých skriptů můžete definovat. Následně voláme jednotlivé příkazy pomocí nainstalovaného příkazu `npm`, kde parametr je alias konkrétního skriptu, např. `npm start`, `npm build` nebo `npm test`.
  - **start** - Obvykle spustí aplikaci ve vývojovém režimu.
  - **build** - Obvykle sestaví a spustí aplikaci.
  - **test** - Obvykle spustí testy aplikace.
- **author** - Obsahuje jméno a e-mail na autora.
  - **name** - Jméno autora nebo vlastníka aplikace.
  - **email** - E-mail autora nebo vlastníka aplikace
- **license** - Typ licence.
- **dependencies** - Seznam všech knihoven, na kterých závisí aplikace.
- **devDpendencies** - Seznam všech knihoven, na kterých závisí vývoj aplikace.

Výše uvedený seznam je orientační, ale dostačující pro vývoj i distribuci aplikace napsané v Electronu. Soubor package.json může obsahovat více parametrů a to především u parametru `build`, který nabízí dlouhou řadu dalších nastavení, která je však závislá na zvoleném nástroji pro sestavování aplikace a samozřejmě na vašich konkrétních požadavcích. V rámci tohoto příkladu jsem zvolil pro sestavování aplikace nástroj electron-builder [15] a jeho nutné parametry pro sestavení aplikace, ale můžete použít jakýkoliv z dalších nástrojů, o kterých se zmíním později.

Pro shrnutí informací o souboru package.json ještě přikládám ukázku reálného použití:

```
{
  "name": "finance-management",
  "version": "1.0.0",
  "description": "Application for finance management",
  "homepage": "https://finance-management.example",
  "repository": "https://github.com/bedrich-schindler/fin-mgmt",
  "main": "index.js",
  "build": {
    "appId": "com.electron.financeManagement",
    "productName": "Finance management",
    "copyright": "Copyright © 2019, Bedrich Schindler"
  },
  "scripts": {
    "build": "electron-builder -mlw",
    "start": "electron ."
```

```
  },
  "author": {
    "name": "Bedrich Schindler",
    "email": "bedrich.schindler@gmail.com"
  },
  "license": "ISC",
  "devDependencies": {
    "electron": "^9.1.0",
    "electron-builder": "^22.7.0"
  }
}
```

Pokud máte nainstalovaný Node.js, vytvořený výše uvedený soubor a použijete příkaz `npm install`, tak by se vám měly stáhnout všechny závislosti a měli byste tak být připraveni postoupit do další fáze.

3.3.2 Soubor index.js (hlavní proces)

Soubor index.js je hlavní JavaScriptový soubor, který tvoří hlavní proces naší aplikace. Jeho hlavním úkolem je komunikace s operačním systémem a jeho grafickým rozhraním, otevírání a správa oken aplikace a komunikace s nimi. Nejprve si ukážeme základní konstrukty k inicializaci Electronu a vytvoření nového okna, posléze se podíváme na některá ze zajímavých API, která nám dovolují komunikaci s jednotlivými částmi operačního systému.

```
const {
  app,
  BrowserWindow,
} = require('electron');
```

Na začátku si naimportujeme `app`, což je hlavní proměnná pro komunikaci hlavního procesu a naší aplikace a `BrowserWindow`, což je třída představující okno aplikace.

```
const createBrowserWindow = () => {
  browserWindow = new BrowserWindow({
    width: 1024,
    height: 768,
    webPreferences: {
      nodeIntegration: true
    }
  });

  browserWindow.loadFile('app.html');
  browserWindow.on('closed', () => {
    browserWindow = null;
  });
};
```

Pomocí funkce `createBrowserWindow` si vytvoříme novou instanci okna s požadovanými rozměry a povolíme oknu přístup k prostředí Node.js. Hned poté si do okna načteme příslušný HTML soubor, v našem případě app.html, a na konci celého bloku funkce provedeme vymazání nepotřebné proměnné. O vymazání proměnné by se při úplném ukončení aplikace postaral Electron sám, ale v případě, že bychom měli příklad s několika okny, bychom značně šetřili paměť počítače.

```
app.on('ready', createBrowserWindow);
```

Pro otevření okna je zapotřebí, abychom po spuštění aplikace, tedy při nastání události `ready`, zavolali výše nadefinovanou funkci pro vytvoření okna.

```
app.on('activate', () => {
  if (browserWindow === null) {
    createBrowserWindow();
  }
});
```

Operační systém MacOS, na rozdíl o operačních systémů Windows a Linux, při stisknutí křížku aplikace danou aplikaci úplně neukončí, ale přesune ji do Docku, tedy hlavní lišty aplikací. Pokud chci na MacOS aplikaci zcela vypnout, tak musím na aplikaci v Docku kliknout pravým tlačítkem a kliknout na Ukončit nebo použít klávesovou zkratku Command+Q. Protože aplikace tedy stále běží v Docku, tak kliknutím na její ikonu dojde k zavolání události `activate` a musím okno aplikace otevřít opakovaným zavoláním výše definované funkce pro vytvoření okna. Událost `activate` se však volá například i po kliknutí na notifikaci v notifikačním centru, takže je potřeba detekovat, jestli okno není náhodou již otevřené.

```
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
});
```

Posledním nezbytným krokem je zavření celé aplikace při zavření všech oken, což uděláme po nastání události `window-all-closed`.

Za předpokladu, že máme již vytvořený soubor app.html, bychom teď celou aplikaci mohli spustit pomocí příkazu `npm start` a aplikace by naběhla. Vše výše uvedené je tedy nezbytná část pro správné spuštění, běh a ukončení aplikace na všech podporovaných systémech.

V druhé části této podkapitoly zmíníme některé ze zajímavých API, která Electron na rozdíl od PWA poskytuje.

Tray API

Tray je prvek v oznamovací oblasti hlavního panelu, který disponuje ikonou a případně vysouvacím menu, které může obsahovat celou řadu prvků jako je text, zaškrtávací pole, přepínací pole, vnořené menu či separátor.

```
const { Tray } = require('electron');
let tray = null;

const createBrowserWindow = () => {
    // ...
    tray = new Tray('./icon.png');
    const trayMenu = Menu.buildFromTemplate([
        {
            label: 'Balance: 450000 CZK',
            type: 'normal',
            enabled: false,
        },
        { type: 'separator' },
        {
            label: 'Add new revenue',
            type: 'normal',
            click: () => console.log('New revenue'),
        },
        // ...
    ]);
    tray.setContextMenu(trayMenu);
};
```

Tray definujeme jakožto globální proměnou v rámci hlavního procesu, aby nedošlo k vymazání této proměnné v rámci garbage collectoru Electronu, který uvolňuje nepoužívanou paměť procesu. Uvnitř funkce `createBrowserWindow` nejdříve nadefinujeme ikonu, která bude zobrazena v rámci oznamovací oblasti hlavního panelu, a poté vysouvací menu s textem, oddělovačem a dvěma tlačítky, které se zobrazí po kliknutí na ikonu. Pokud chceme Tray odebrat, stačí zavolat funkci `tray.destroy()`.

Obrázek 3.2: Tray

TouchBar API

Některá zařízení Apple MacBook Pro disponují tzv. TouchBarem, což je dotykový pruh nad klávesnicí, který umožňuje uživateli ovládat základní systémová nastavení a interagovat s aplikacemi. Electron disponuje v rámci hlavního procesu API pro ovládání zobrazení elementů na TouchBaru, jimiž může být text, tlačítko, element pro výběr barvy, element pro výběr hodnoty z rozsahu, popř. elementy na seskupování zmíněných elementů.

```
const { TouchBar } = require('electron');
const {
  TouchBarLabel,
  TouchBarButton,
  TouchBarSpacer,
} = TouchBar;

const createBrowserWindow = () => {
  // ...
  const balanceLabel = new TouchBarLabel({
        label: 'Balance: 450000 CZK',
  });
  const spacer = new TouchBarSpacer({
        size: 'flexible',
  });
  const addExpenseButton = new TouchBarButton({
    label: 'Add expense',
    click: () => {},
  });
  const touchBar = new TouchBar({
    items: [
      balanceLabel,
      spacer,
      addExpenseButton,
      // ...
    ],
  });

  browserWindow.setTouchBar(touchBar);
};
```

Ukázkový kód zobrazí v TouchBaru text a dvě tlačítka, která jsou oddělena flexibilní mezerou. Výsledek je vidět na přiloženém obrázku.

Obrázek 3.3: TouchBar

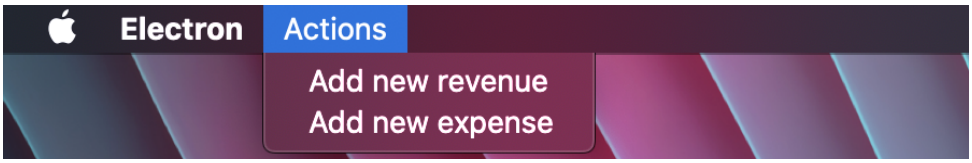
Menu API

Menu slouží k vytvoření nabídky akcí, které aplikace podporuje a zobrazuje je ve stavové liště. Mimo jiné lze pomocí Menu i definovat klávesové zkratky jednotlivých akcí.

```
const menu = Menu.buildFromTemplate([
  { role: 'appMenu' },
  {
    label: 'Actions',
    submenu: [
      {
        label: 'Add new revenue',
        type: 'normal',
        click: () => console.log('New revenue'),
        accelerator: 'CmdOrCtrl+1',
      },
      {
        label: 'Add new expense',
        type: 'normal',
        click: () => console.log('New expense'),
        accelerator: 'CmdOrCtrl+2',
      }
    ]
  }
]);

Menu.setApplicationMenu(menu);
```

Ukázkový kód vytvoří nabídku dvou akcí s možností zavolat obě dvě akce stisknutím příslušných klávesových zkratek.



Obrázek 3.4: Menu

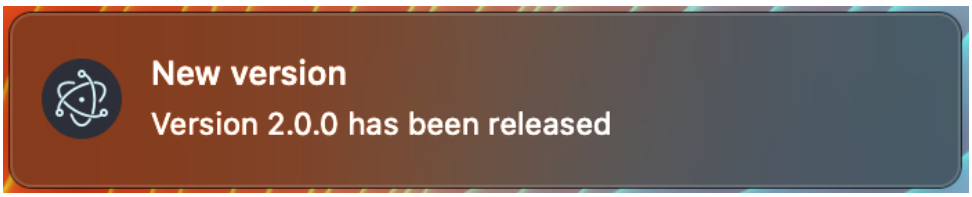
Notification API

Pro vytváření notifikací v rámci hlavního procesu nelze použít HTML5 Notification API, které se používá běžně ve webových aplikacích, ale musíte využít třídu Notification, která je součástí Electronu. V rámci vykreslovacího procesu však HTML5 Noficications API použít lze.

```
let notification = null;
const createBrowserWindow = () => {
  // ...
  notification = new Notification({
    title: 'New version',
    body: `Version ${appVersion} has been released`,
  });
  notification.show();

  setTimeout(() => { notification.close(); }, 15000);
};
```

Ukázkový kód zobrazí notifikaci, která po 15 sekundách opět zmizí.



Obrázek 3.5: Notifikace

Progress Bar API

Pokud aplikace vyžaduje operace, při nichž je vhodné vizualizovat procentuální stav dokončení dané operace, tak operační systémy MacOS a Windows umí u ikony aplikace v hlavní nabídce zobrazit progres.

```
let progress = 0;
const progressInterval = setInterval(() => {
    browserWindow.setProgressBar(progress++ / 100);

    if (progress === 100) {
        browserWindow.setProgressBar(-1);
        clearInterval(progressInterval);
    }
}, 100);
```

Ukázkový kód postupně zobrazí stav od nuly do sta procent, a pak se nastavením na hodnotu -1 resetuje a vizualizaci skryje.

Obrázek 3.6: Progress

Další API

Electron poskytuje řadu dalších API, která jsou při vytváření běžných aplikací více či méně potřebná, a proto jsem tu zmínil jen ty nejzajímavější. Pro kompletní seznam API [16] navštivte webové stránky Electronu a jeho sekci s dokumentací.

3.3.3 Soubor app.html

Už jsem představil vstupní bod aplikace i řadu zajímavých API, jež Electron poskytuje, ale stále jsem nepředstavil, jak zobrazit obsah jednotlivých oken. Obsah jednotlivých oken není nic jiného než HTML, CSS a JavaScript, tak jak ho známe z běžných webových aplikací, který je v případě Electronu obohacen o funkce, které nejsou v běžném prohlížeči dostupné.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1"
    >
    <title>Finance Management</title>
    <meta
      http-equiv="Content-Security-Policy"
      content="script-src 'self' 'unsafe-inline';"
    />
   <link rel="stylesheet" href="app.css">
   <script src="app.js"></script>
  </head>
  <body>
    <main id="content" class="content"></main>
  </body>
</html>
```

Jak je vidět, u Electronu není zapotřebí žádných speciálních definic, jako je tomu v případě PWA. Jedná se o HTML soubor s běžnými definicemi v záhlaví, které známe z běžných webových aplikací. Z toho samého důvodu přeskočím popis souboru app.css s CSS styly, protože zde není nic nestandardního.

3.3.4 Soubor app.js (vykreslovací proces)

Vykreslovacím procesem by se v kontextu Electronu mohl nazvat již soubor app.html, ale protože pro přehlednost dělíme HTML, CSS a JS do zvláštních souborů, tak je vykreslovací proces delegován právě na soubor app.js.

V rámci vykreslovacího procesu můžete s aplikací nakládat úplně stejně jako v případě webové aplikace, a to díky již dříve zmiňovanému Chromiu.

Chromium poskytuje všechny API, které známe z vývoje webových aplikací jako třeba Storage API, Notifications API, Service Worker API atd.

Electron však do vykreslovacího procesu přidává proměnnou `remote`, která nám poskytuje možnost komunikovat s hlavním procesem a volat funkce a třídy, které nám běžně nabízí hlavní proces. Další zajímavou proměnnou, která je v rámci vykreslovacího procesu k dispozici, je `ipcRenderer`. Ta slouží k asynchronní komunikaci s hlavním vykreslovacím procesem, kde existuje pro opačné účely proměnná `ipcMain`. Právě ukázku této komunikace si ukážeme.

```
const { ipcRenderer } = require('electron');

const categories = [
  {
    label: 'Lunch',
    click: () => {},
  },
  {
    label: 'Dinner',
    click: () => {},
  },
];

ipcRenderer.invoke('replaceTouchBarContent', categories)
  .then((result) => {
    console.log(result ? 'Updated' : 'Not updated');
  });
```

V rámci vykreslovacího procesu pomocí `ipcRenderer` odešleme asynchronní zprávu s libovolnými parametry a případně zareagujeme na odpověď, která se nám dostane z hlavního procesu.

```
const { ipcMain } = require('electron');

ipcMain.handle('replaceTouchBarContent', async (e, categories) => {
    return replaceTouchBarContent(categories);
});
```

V rámci hlavního procesu musíme mít zároveň nadefinovaný posluchač na příchozí zprávy z vykreslovacích procesů, které zpracují náš požadavek a vrátí nám výsledek.

3.4 Sestavování a distribuce

Jak již bylo řečeno, sestavení a distribuce aplikace vytvořené v Electronu je trochu složitější. V oficiální dokumentaci Electronu je sice popsán způsob, jak aplikaci sestavit manuálně, ale takový přístup není efektivní, a proto je vhodné použít nějaký z nástrojů, který je pro tyto účely vyvinut. Oficiální dokumentace se zmiňuje o těchto třech:

- electron-builder
- electron-forge
- electron-packager

Tyto nástroje slouží především k tomu, že na základě konfiguračního souboru provedou správné sestavení aplikace a provedou všechny nezbytné kroky k tomu, aby na jejich výstupu byly spustitelné soubory, popř. instalátory vaší aplikace. Dokáží však obstarat i automatické aktualizace aplikace.

Použití nástroje electron-builder, a to včetně základní konfigurace, jsme měli možnost vidět v popisu souboru package.json. Důvodem použití právě tohoto nástroje je největší komunita a používanost vůbec. electron-forge je, na rozdíl od nástroje electron-builder, nástroj nejen na sestavování a distribuci vaší aplikace, ale je to celý balíček pro vývoj aplikací v Electronu. Poslední zmiňovaný electron-packager plní stejné funkce electron-builder, ale je vyvíjen jako součást nástroje electron-forge.

Distribuce výsledných spustitelných souborů, případně instalačních souborů, je čistě ve vaší režii, případně můžeme využít publikování např. přes platformy AppStore či Windows Store.

Kapitola 4
Hybridní aplikace

4.1 Definice

Hybridní aplikace je nativní aplikace využívající prvku WebView [17, 18], ve kterém zobrazuje webovou stránku. Komponenta WebView je nativní prvek, který představuje vykreslovací okno prohlížeče, který se liší v závislosti na platformě.

V případě systémů iOS, iPadOS a MacOS komponenta WebView využívá prohlížeče Safari, v případě systému Android prohlížeče Google Chrome. Hybridní aplikace nejsou omezeny pouze na mobilní operační systémy, ale je možné je vyvíjet i pro desktopové operační systémy. Stačí aby daný programovací jazyk a platforma měly k dispozici prvek WebView.

4.2 Princip

Webové technologie HTML, CSS a JavaScript nám sice umožňují poměrně rychle vytvořit aplikaci, ale v některých případech nám nestačí API, která prohlížeče poskytují a potřebujeme využívat další funkce dané platformy, které jsou dostupné právě pouze v rámci nativních aplikací.

Hybridnost aplikace spočívá tedy v tom, že hlavní funkcionalita aplikace se odehrává v rámci jedné webové aplikace, která buď pomocí zpráv nebo pomocí vystavené třídy nativní aplikace do webové aplikace, komunikuje s nativní aplikací a nativní aplikace pak obstarává funkcionalitu, která v rámci prohlížeče není dostupná. Komunikace funguje i v opačném směru, tedy z nativní aplikace do webové aplikace, a to buď pomocí zpráv nebo přímo pomocí spouštění JavaScriptového kódu.

Bezespornou výhodou je velmi rychlý vývoj multiplatformních aplikací, protože většinu funkcionality obstarává aplikace napsaná v HTML, CSS a JavaScriptu a nativní část je tedy velmi minimalistická v porovnání s vývojem kompletně nativní aplikace. Další výhodou s tím spojenou je sdílení poměrně velké části kódu, protože část napsaná pomocí webových technologií je sdílena napříč všemi platformami a pro jednotlivé podporované platformy pouze vyvíjíme základní aplikace s WebView a případně s dalšími funkcemi, o které chceme aplikaci obohatit.

Nevýhodami jsou nutná znalost těch programovacích jazyků, jež jsou nezbytné pro vývoj nativních části dané aplikace či využívání nenativních prvků.

Hybridní aplikace však nejsou příliš oblíbené a nejvíce tomu tak je na mobilních platformách iOS a iPadOS, proti kterým společnost Apple čím dál tím více zbrojí [19]. Pokud totiž uděláte hybridní aplikaci, jejíž hlavní funkcionalitu obstarává právě webová aplikace uvnitř WebView, tak podle upozornění společnosti Apple může poměrně jednodušše dojít k zamítnutí takové aplikace při pokusu o její zveřejnění v obchodu AppStore.

4.3 Ukázka

V rámci kapitoly o hybridních aplikacích vynechám komplexní ukázky kódu, jež jsem psal u PWA a Electronu, protože existuje celá řada programovacích jazyků a platforem, a tudíž není ani možné všechny ukázky vypsat. Ukážu však alespoň krátkou ukázku hybridní aplikace napsané v jazyce Swift, a to pro systémy iOS a iPadOS, a poté ukázku v jazyce Java pro systém Android.

4.3.1 Ukázka pro iOS a iPadOS

Pomocí vývojového prostředí Xcode si vytvoříme projekt typu Single View App pro systém iOS, a to pomocí jazyka Swift a uživatelského rozhraní Storyboard.

V rámci projektu si vytvoříme jednoduchý soubor index.html, ve kterém budeme mít tlačítko, které při kliknutí odešle do nativní aplikace zprávu, kterou nativní část aplikace zpracuje a zobrazí notifikaci. Pro demonstraci obousměrné komunikace dojde po zpracování zprávy nativní aplikací k vyvolání JavaScriptové události, na kterou webová aplikace zareaguje zobrazením textu pod tlačítkem.

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <script>
window.addEventListener('load', () => {
    const btn = document.getElementById('showNotification');
    const msg = 'This is notification from inside of WebView';
    btn.addEventListener('click', () => {
        window.webkit.messageHandlers.notificationHandler
            .postMessage(msg);
    });
})
window.addEventListener('ios.onNotify', () => {
    const text = document.createTextNode("Notification fired!");
    document.body.appendChild(text);
```

```
});
        </script>
    </head>
    <body>
        <button id="showNotification">Show notification</button>
    </body>
</html>
```

Kontrolér aplikace `ViewController`, rozšíříme tak, aby nám zobrazil prvek WebView, načetl do něj soubor index.html, naslouchal v rámci handleru `notificationHandler` na zprávu z něj odeslanou, a zobrazil notifikaci s textem, který v rámci zprávy obdržel.

Dále po obdržení zprávy spustíme JavaScriptový kód nad komponentou WebView, který zavolá událost, jež posléze vypíše text o úspěšném zobrazení notifikace. Detailnější popis daných konstruktů naleznete přímo v komentářích ukázky zdrojového kódu.

```
import UIKit
import WebKit
import UserNotifications

class ViewController: UIViewController, WKScriptMessageHandler {
  override func viewDidLoad() {
    super.viewDidLoad()

    // Registrace handleru pro zprávy vyvolaných z WebView
    let contentController = WKUserContentController()
    contentController.add(self, name: "notificationHandler" )

    // Konfigurace elementu WebView
    let config = WKWebViewConfiguration()
    config.userContentController = contentController

    // Inicializace a vložení elementu WebView do obrazovky
    let webView = WKWebView(
      frame: view.bounds,
      configuration: config
    )
    view.addSubview(webView)

    // Načtení souboru index.html do elementu WebView
    let url = Bundle.main.url(
      forResource: "index",
      withExtension: "html"
    )
    webView.load(URLRequest(url: url!))
```

```
    // Vyžádání oprávnění k notifikacím
    UNUserNotificationCenter.current()
      .requestAuthorization(options: [.alert]) {
      (granted, error) in
        if !granted { print("Not granted") }
    }
  }

  func sendNotification(_ message: String) {
    // Vytvoření notifikace
    let notification = UNMutableNotificationContent()
    notification.title = "Finanace management"
    notification.body = message

    // Vytvoření požadavku na zobrazení notifikace
    let trigger = UNTimeIntervalNotificationTrigger(
      timeInterval: 5,
      repeats: false
    )
    let request = UNNotificationRequest(
      identifier: "financeManagement",
      content: notification,
      trigger: trigger
    )

    // Registrace požadavku na zobrazení notifikace
    UNUserNotificationCenter.current()
      .add(request, withCompletionHandler: nil)

    for subview in view.subviews {
      if let webViewB = subview as? WKWebView {
        // Spustí Javascriptový kód, který zavolá událost
        webViewB.evaluateJavaScript(
"(() => { window.dispatchEvent(new Event('ios.onNotify'));})();"
        )
      }
    }
  }

  func userContentController(
    _ userContentController: WKUserContentController,
    didReceive message: WKScriptMessage
  ) {
    // Zpracování zprávy z index.html z WebView
    if (message.name == "notificationHandler") {
      self.sendNotification(message.body as! String)
```

```
    }
  }
}
```

Jak je tedy vidět v krátké ukázce, tak stačí vytvořit webovou aplikaci, načíst ji do elementu WebView a pomocí zpráv komunikovat s nativní částí aplikace, která může obsluhovat ty věci, které bychom pomocí webového prohlížeče nebyli schopni provést.

4.3.2 Ukázka pro Android

Pomocí vývojového prostředí Android Studio si založíme prázdný projekt pro mobilní zařízení, konkrétně v jazyce Java.

V rámci projektu si vytvoříme jednoduchý soubor index.html, ve kterém bude tlačítko, které při kliknutí zavolá funkci pro zobrazení notifikace. Tato funkce bude ve webové aplikace dostupná v rámci třídy, kterou si vystavíme prostřednictvím WebView.

Pro demonstraci obousměrné komunikace dojde po stisknutí tlačítka k jeho deaktivování. Po obnovení aplikace z nabídky otevřených aplikací vyvolá nativní aplikace JavaScriptovou událost, na kterou zareaguje webová aplikace znovuaktivováním tlačítka.

Obsah souboru index.html se liší od ukázky pro iOS a iPadOS pouze v bloku `script`, proto uvedu pouze tento kus kódu.

```
<script>
window.addEventListener('load', () => {
  const btn = document.getElementById('showNotification');
  const msg = 'This is notification from inside of WebView';

  btn.addEventListener('click', () => {
    androidNotificationHandlers.showNotification(msg);
    btn.disabled = true;
  });
});

window.addEventListener('android.onResume', () => {
  const btn = document.getElementById('showNotification');
  btn.disabled = false;
});
</script>
```

Hlavní obrazovku, jež bude sestávat z kontroléru MainActivity.java a definice obrazovky activity_main.xml, rozšíříme o prvek WebView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

V MainActivity.java v rámci WebView načteme soubor index.html a vystavíme prostřednictvím něj třídu, která bude dostupná v rámci webové aplikace a bude možné volat její funkce v rámci JavaScriptu.

Dále po obnovení aplikace z nabídky otevřených aplikací spustíme JavaScriptový kód nad komponentou WebView, jež zavolá událost pro znovuaktivování tlačítka. Detailnější popis daných konstruktů naleznete přímo v komentářích ukázky zdrojového kódu.

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.ValueCallback;
import android.webkit.WebView;

public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Načtení elementu WebView do proměnné
    WebView webView = findViewById(R.id.webView);

    // Načtení souboru index.html do elementu WebView
    webView.loadUrl("file:///android_asset/index.html");

    // Povolení JavaScriptu v rámci elementu WebView
    webView.getSettings().setJavaScriptEnabled(true);

    // Registrace handleru pro funkce vyvolaných z WebView
    webView.addJavascriptInterface(
      new JsMessageHandlers(this),
      "androidNotificationHandlers"
    );
```

```
  }

  @Override
  protected void onResume() {
    super.onResume();

    // Načtení elementu WebView do proměnné
    WebView webView = findViewById(R.id.webView);

    // Vyvolání JavaScriptové události
    webView.evaluateJavascript(
      "(() => { window.dispatchEvent(" +
      "new Event('android.onResume'));" +
      "})();",
      null
    );
  }
}
```

Třída JsMessageHandlers.java, resp. její funkce pro zobrazení notifikace, bude vystavena v rámci prvku WebView, takže bude dostupná uvnitř JavaScriptu v index.html. Při jejím zavolání dojde k zobrazení notifikace.

```
import android.content.Context;
import android.webkit.JavascriptInterface;
import android.widget.Toast;

public class JsMessageHandlers {
    private Context context;

    JsMessageHandlers(Context c) {
        context = c;
    }

    @JavascriptInterface
    public void showNotification(String msg) {
        Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
    }
}
```

Jak je vidět v krátké ukázce, tak i pro hybridní aplikaci pro systém Android stačí vytvořit webovou aplikaci a načíst ji do elementu WebView, přičemž tentokráte pro komunikaci mezi nativní a webovou aplikací použijeme třídu, kterou nám WebView v rámci index.html zpřístupní, popř. použijeme funkci pro spouštění JavaScriptu v rámci WebView.

4.3.3 Ukázka společná pro všechny platformy

Uvedli jsme si, že v rámci prvku WebView otevřeme soubor index.html a podle platformy z něj zavoláme po kliknutí na tlačítko příslušnou funkci, která způsobí zobrazení notifikace. Hybridní aplikace však píšeme z důvodu, abychom nemuseli všechnu funkcionalitu psát pro každou platformu zvlášť. Proto všechnu funkcionalitu napíšeme společně pro všechny platformy a nakonec zavoláme pouze příslušnou funkci podle platformy.

```
<script>
window.addEventListener('load', () => {
  const btn = document.getElementById('showNotification');
  const msg = 'This is notification from inside of WebView';

  btn.addEventListener('click', () => {
    if (window.webkit && window.webkit.messageHandlers) {
      window.webkit.messageHandlers.notificationHandler
        .postMessage(msg);
    } else if (androidNotificationHandlers) {
      androidNotificationHandlers.showNotification(msg);
    } else if (
      window.Notification
      && Notification.permission === 'granted'
    ) {
      new Notification(msg);
    }
  });
});
</script>
```

Kapitola 5

Porovnání technologií, výběr technologie a návrh aplikace

5.1 Shrnutí popsaných technologií

V předchozích třech kapitolách jsme si představili progresivní webové aplikace, Electron a hybridní webové aplikace, popsali jejich principy, implementaci a případně i způsob distribuce. Je nutné podotknout, že neexistuje žádný konkrétní recept na to, jakou technologii bychom měli zvolit při tvorbě multiplatformní aplikace, neboť se pro každou aplikaci může hodit jiná technologie v závislosti na požadavcích, které by měla splňovat.

Nesmíme zapomínat, že tato práce pojednává o vývoji multiplatformních aplikací za pomoci webových technologií HTML, CSS a JavaScriptu, a zmiňuje se pouze o vybraných technologiích, které jsem zvolil dle svého uvážení, protože jsou mi nejblíže a mluví se o nich nejvíce.

Za prvé existuje řada dalších technologií, které využívají více či méně webové technologie a umožňují multiplatformní vývoj, mezi nimiž jsou například React Native či Ionic. Za druhé tu jsou technologie, které sice nejsou postaveny na webových technologiích, ale umožňují taktéž multiplatformní vývoj, mezi nimiž je například Flutter, Xamarin nebo Unity. Za třetí tu je cesta nativních aplikací, která je v řadě případů stále nenahraditelnou a nelze na ni zapomínat.

5.2 Popis aplikace

Abychom mohli porovnat uvedené technologie a zvolit tu nejlepší variantu, tak by bylo vhodné se o daných problémech bavit nad konkrétní aplikací. Pro tuto práci jsem zvolil aplikaci pro evidenci účetních a provozních dat. Co je motivací?

Jakožto OSVČ a student zároveň musím při své práci řešit řadu činností, na které mám nejrůznější programy a tabulky, které mezi sebou nekomunikují. Mezi nimi je evidence klientů, projektů a informací s nimi spojenými, a to včetně evidence odpracovaného času. Dále vytváření faktur a v neposlední řadě pak evidence plateb sociálního a zdravotního pojištění. A mimo to mám na všechno nastavená upozornění v daných aplikacích či kalendáři.

Jsou to v zásadě poměrně jednoduché aktivity, které však nejsou nijak centralizované, a tudíž by mi přišlo dobré vytvořit aplikaci, jež všechny tyto činnosti dokáže obsloužit naráz.

5.3 Požadavky aplikace

Funkční požadavky aplikace jsou:

- Evidence klientů
- Evidence projektů u jednotlivých klientů
- Evidence odpracovaných hodin u jednotlivých projektů
- Nastavení fakturačních údajů
- Evidence provedených plateb za daně, sociální, zdravotní a nemocenské pojištění
- Evidence a vystavování faktur
- Zobrazení přehledu dat
- Upozornění na platbu záloh na dani, sociálním, zdravotním a nemocenském pojištění
- Upozornění na fakturu po splatnosti
- Časovač pro zaznamenávání odpracovaného času

Nefunkční požadavky aplikace jsou:

- Práce v režimu offline a ukládání dat
- Funkčnost napříč platformami

Výše uvedený seznam požadavků je minimální. V případě, že nám nějaká z uvedených technologií nabídne zajímavé API, tak bychom mohli zvážit, zda ho nevyužít a nepřiklonit se na stranu právě té technologie.

5.4 Porovnání požadavků aplikace

Výše uvedené funkční i nefunkční požadavky si jednotlivě projdeme, rozepíšeme a zhodnotíme vhodnost použití konkrétních technologií.

5.4.1 Evidence klientů

V rámci evidence klientů je nezbytné mít možnost si zobrazit seznam všech klientů, detail konkrétního klienta a mít možnost vytvořit nového klienta nebo upravit či odstranit již existujícího klienta. Klienta lze odstranit pouze v momentě, kdy k němu nejsou přiřazena žádná další data, která by způsobila nekonzistenci.

Šikovným doplňkem může být za prvé implementace klávesových zkratek, a to společně s menu ve stavové liště aplikace a za druhé integrace s TouchBarem.

Jedná se o čistě webovou záležitost, takže by PWA mohlo být dostačující technologií, ale vzhledem k uvedeným doplňkům je ještě výhodnější využít Electron, který nám dovolí na desktopových systémech používat uvedené pokročilejší funkce, které pomohou uživateli v efektivnosti při využívání aplikace.

5.4.2 Evidence projektů u jednotlivých klientů

V rámci evidence projektů u jednotlivých klientů je nezbytné mít možnost si zobrazit seznam všech projektů a mít možnost si vytvořit nový projekt nebo upravit či odstranit již existující projekt. Projekt je vždy vázán na klienta, takže není možné vytvořit projekt bez přiřazeného klienta. Projekt lze odstranit pouze v momentě, kdy k němu nejsou přiřazena žádná další data, která by způsobila nekonzistenci.

I v tomto případě může být šikovným doplňkem implementace klávesových zkratek společně s menu ve stavové liště aplikace a s integrací s TouchBarem. Ze stejných důvodů, jako v předchozím bodu, je nejvhodnější využít Electron.

5.4.3 Evidence odpracovaných hodin u jednotlivých projektů

V rámci evidence odpracovaných hodin je nezbytné mít možnost si zobrazit seznam všech záznamů odpracovaných hodin a mít možnost vytvořit nový záznam nebo upravit či odstranit existující záznam. Záznam o odpracovaném čase je vždy vázán na projekt, proto ve výpisu mohu filtrovat záznamy podle projektu, a dále i podle rozmezí dat.

I v tomto případě může být šikovným doplňkem implementace klávesových zkratek společně s menu ve stavové liště aplikace a s integrací s TouchBarem. Ze stejných důvodů, jako v předchozích dvou bodech, je nejvhodnější využit Electron.

5.4.4 Nastavení fakturařních údajů

Pro jednoduchost vytváření faktur je nezbytné mít možnost nastavit fakturační údaje tak, aby při vytváření faktur docházelo k automatickému vyplnění hodnot.

Dostačující technologií pro tento bod je PWA, neboť se jedná o standardní webovou stránku bez pokročilých funkcionalit.

5.4.5 Evidence provedených plateb za daně, sociální, zdravotní a nemocenské pojištění

V rámci evidence zaplacených záloh na dani, sociálním a zdravotním pojištění, je nezbytné mít možnost si zobrazit seznam všech záznamů a mít možnost si vytvořit nový záznam nebo upravit či odstranit existující záznam. Každý typ platby je uveden na zvláštní stránce.

Jako u předchozích bodů sloužících k evidenci, i v tomto případě může být šikovným doplňkem implementace klávesových zkratek společně s menu ve stavové liště aplikace a s integrací s TouchBarem. Ze stejných důvodů je nejvhodnější využít Electron.

5.4.6 Evidence a vystavování faktur

Fakturu vystavujeme konkrétnímu klientovi, jehož údaje se automaticky vyplní při zvolení klienta. Dále volíme projekty, kterých se faktura týká a vybíráme všechny záznamy odpracovaných hodin, které se k dané faktuře vztahují. Nakonec volíme položky faktury, data vystavení a splatnosti a údaje o platbě.

Po vystavení faktury musí být možné vygenerovat fakturu a přehled odpracovaného času, a to ve formátu PDF. Zároveň u těch záznamů odpracovaných hodin, které jsme přiřadili této faktuře, dojde k zobrazení příznaku, který uvádí, že daný záznam o odpracovaném čase byl již vyfakturován. V neposlední řadě dojde k zobrazení vystavené faktury v seznamu vystavených faktur, kterou mám možnost upravit či odstranit a také označit jako zaplacenou.

Jako u předchozích bodů sloužících k evidenci, i v tomto případě může být šikovným doplňkem implementace klávesových zkratek společně s menu ve stavové liště aplikace a s integrací s TouchBarem. Ze stejných důvodů je nejvhodnější Electron.

5.4.7 Zobrazení přehledu dat

Pro zobrazení měsíčních a ročních přehledů vystavených a zaplacených faktur je postačující technologií PWA.

5.4.8 Upozornění na platbu záloh na dani, sociálním, zdravotním a nemocenském pojištění

Upozornění na platbu záloh na dani, sociálního, zdravotního a nemocenského pojištění, se nastaví ve speciální sekci pro upozornění, kde si uživatel bude moci nejen nastavit, zda chce dané upozornění obdržet či nikoliv, ale kdy chce upozornění zobrazit. Upozornění se v první verzi aplikace budou zobrazovat ten den, na kdy máme upozornění nastavené, a to bez ohledu na evidovanou platbu.

Tady je situace složitější, neboť musíme využívat API pro notifikace. V případě podpory Notifications API nastává problém obecně u iOS a iPadOS, dále pak u prohlížečů Internet Firefox, Opera Mini či Samsung Internet,

jejichž celosvětové použití je mezi 1% až 3%. Tyto prohlížeče totiž nepodporují Notifications API.

Je potřeba zvážit, jak jsou pro nás notifikace podstatné. Kritickou platformou je pro nás iOS, resp. iPadOS, na kterém žádný z prohlížečů nepodporuje toto API. U ostatních systémů lze totiž funkčnosti docílit použitím podporovaného prohlížeče.

Řešením je hybridní aplikace, která by suplovala chybějící funkcionalitu notifikací, a to alespoň na systémech iOS a iPadOS.

5.4.9 Upozornění na fakturu po splatnosti

Upozornění na fakturu po splatnosti se, stejně jako v předchozím bodě, nastaví ve speciální sekci pro upozornění, kde si uživatel bude moci nastavit, zda chce dané upozornění obdržet či nikoliv. Problém s podporou je totožný s předchozím bodem.

5.4.10 Časovač pro zaznamenávání odpracovaného času

Časovač pro zaznamenávání odpracovaného času je rozšířením evidence odpracovaného času pro efektivnější práci s aplikací. Jedná se o plovoucí tlačítko v rámci aplikace, které umožňuje stisknutím spustit časovač, který nám po celou dobu ukazuje aktuálně odpracovaný čas, přičemž při ukončení časovače zadáme projekt, ke kterému se odpracovaný čas vztahuje.

Šikovným doplňkem může být za prvé implementace klávesových zkratek, a to společně s menu ve stavové liště aplikace a za druhé integrace s TouchBarem, kde uživatel kromě spuštění a zastavení časovače, uvidí aktuální čas a možnost přiřadit konkrétní projekt. Třetím doplňkem může být integrace s prvkem Tray v oznamovací oblasti hlavního panelu, který by taktéž mohl uživateli ukazovat odpracovaný čas a možnost spustit či zastavit časovač.

Bez doplňků je dostačující technologií PWA, ale pro implementaci doplňků je nezbytné využít Electron. Na mobilních zařízeních by mohlo být pěkné implementovat widget, který by fungoval stejně jako časovač, což by vedlo k využití hybridní aplikace.

5.4.11 Práce v režimu offline a ukládání dat

Používání aplikace v režimu offline se dá docílit využíváním služby běžící na pozadí (anglicky Service workeru) v kombinaci s Cache API, které zajišťuje ukládání požadavků do mezipaměti prohlížeče. V závislosti na připojení k internetu pak můžeme používat uložená data, deaktivovat některé funkcionality aplikace, které v offline režimu nejsou dostupné a reagovat na změny připojení.

Vše tohle je dostupné v běžných prohlížečích a dostačující technologií je pro tento bod PWA.

5.4.12 Funkčnost napříč platformami

V celé této práci nám jde o to, aby aplikace fungovala napříč platformami a měli jsme tak možnost pomocí jednoho společného kódu obsloužit všechny hlavní používané platformy.

5.5 Volba technologie aplikace

Jak ukazuje detailní rozbor, tak nelze jednoznačně určit, jaká technologie je vhodná a jaká ne. Různé problémy se dají řešit nejlépe v různých technologiích, a proto si potřebujeme nastavit priority a očekávání tak, abychom dokázali zvolit co nejoptimálnější technologii.

Naše aplikace pro evidenci účetních a provozních dat musí být určitě dostupná na všech platformách, tedy na desktopu, tabletu i telefonu. Z detailního rozboru jednotlivých požadavků je však vidět, že se podpora API liší v závislosti na systému. Musíme si tedy uvědomit, jak tuto aplikaci bude uživatel nejspíše využívat.

Z povahy aplikace je vidět, že se jedná převážně o vedení účetních a provozních dat. Bezesporu mohou nastat situace, kdy budeme aplikaci používat na mobilním telefonu.

Ačkoliv by aplikace měla být použitelná na mobilním zařízení, tak se stále jedná o agendu, kterou člověk dělá převážně při práci, pro kterou poměrně často používá desktop. A to ať už se jedná o samotný výkon práce, tak pro placení záloh na dani, sociálním a zdravotním pojištění nebo vystavování a rozesílání faktur, desktop stále při výkonu těchto činností dominuje, i když ho v řadě činností začínají nahrazovat tablety.

Z detailního rozboru jednotlivých požadavků nám vyšlo, že nejčastěji dostačující technologií je PWA, která je společná pro všechny platformy. V rozboru se nám však často objevuje i Electron. Electron není nikde vysloveně nutný, protože všechna základní funkcionalita je providitelná právě pomocí PWA, ale Electron poskytuje řadu zajímavých a užitečných API, která by se v této aplikaci dala využít. A jak jsme zmínili v předchozím odstavci, tak očekáváme největší využití na desktopových systémech, takže by se nám tyto doplňky mohly hodit.

O hybridních aplikacích jsme se zmiňovali ve dvou případech, a to z důvodu prozatimního nepodporování Notifications API na platformách iOS a iPadOS a v případě možného widgetu pro časovač odpracované práce. Jak jsme se však zmínili, tak primární využití očekáváme na desktopových systémech, a proto by tvorba hybridní aplikace pouze pro potřeby widgetu byla příliš nákladná, protože bychom museli vytvořit aplikaci pro Android, iOS a iPadOS a přínos widgetu v tomto případě není takový.

Zajímavější využití hybridní aplikace by mohlo být pro suplování chybějícího Notifications API na iOS a iPadOS. Dle statistických dat portálu StatCouter [20] bylo ke konci roku 2019 iOS nainstalováno na 23% a Android na 76% všech telefonů na světě. U tabletů je rozložení trhu opačné, neboť zde iPadOS společně s iOS dosahují 63% a Android 37%. Pokud se omezíme na Českou

republiku, tak tam bylo iOS nainstalováno na 20% a Android na 80% všech telefonů na světě. U tabletů je rozložení trhu přesně napůl, tedy půlka trhu využívá iPadOS společně s iOS a druhá půlka trhu Android. Ostatní mobilní operační systémy nestojí za zmínku, neboť se jejich využití pohybuje hluboko pod hranicí 1%.

Pokud bychom se podívali na statistiky rozložení trhu společně pro telefony a tablety, tak se celosvětově u iOS a iPadOS dostáváme na jednu čtvrtinu trhu, v případě České republiky k jedné pětině trhu. Pokud zanedbáme ostatní systémy, tak zbytek trhu obsazuje Android.

Rozhodnout v tomto případě o tom, zda se nám vyplatí vytvořit hybridní aplikaci pro iOS a iPadOS, je velmi těžké. Víme tedy, že se použití iOS a iPadOS, na rozdíl od Androidu, pohybuje okolo 25% a nepodporují zatím Notifications API a nemáme tedy možnost z PWA vyvolat notifikaci, můžeme zobrazovat upozornění pouze v rámci hybridní aplikace. Zároveň jsme uvedli, že očekáváme primární použití na desktopu.

Z těchto důvodů bych si osobně, jakožto autor aplikace, dovolil nevyužít technologii hybridních aplikací. Sice se jedná o nemalé procento uživatelů, které oberu o funkcionalitu, ale musíme si uvědomit, že musíme vytvořit nativní aplikaci, kterou musíme dále udržovat a navíc se musíme naučit novou technologii a programovací jazyk. Dalším problémem je, že Apple v jednom svých prohlášení [19] uvádí, že aplikace, jejichž hlavní funkcionalita bude napsaná právě v HTML, CSS a JavaScriptu a bude zobrazována v rámci prvku WebView, bude s největší pravděpodobností odmítnuta a nebude možné ji nahrát do AppStoru.

Co se týče technologií PWA a Electron, tak z předchozího je patrné, že je PWA technologií dostačující. Electron nám však dokáže nabídnout takové doplňkové funkce, které by mohly pomoci v efektivním využívání aplikace a zlepšení uživatelského zážitku, a proto v tomto konkrétním případě zvolíme kombinaci obou technologií.

Výhoda Electronu oproti hybridní aplikaci je taková, že v případě Electronu používáme technologie, které už známe z webového vývoje. Navíc Electron sám o sobě je prohlížečem, takže na hotovou PWA aplikace pouze navěsíme tu funkcionalitu Electronu, kterou chceme a i když budeme mít aplikaci pro desktopová zařízení, tak její vývoj a údržba bude, v porovnání s hybridní aplikací, minimální.

Z výše uvedeného tedy vyplývá, že je vhodné primárně implementovat aplikaci jako PWA a případně ji zabalit do Electronu, který může poskytnout řadu zajímavých API, o kterých jsme se zmiňovali výše.

Kapitola 6

Implementace aplikace

6.1 Úvod do implementace aplikace

V prvních třech kapitolách jsme si představili PWA, Electron a hybridní aplikace a v předchozí kapitole jsme si definovali popis aplikace, její funkční a nefunkční požadavky a pro implementaci aplikace zvolili PWA a Electron.

V této kapitole si přiblížíme samotnou implementaci naší aplikace, zvolené programovací jazyky, technologie a knihovny, které byly při implementaci použity a zmíníme se o detailech implementace vybraných částech aplikace.

Vzhledem k tomu, že tato kapitola odráží reálnou implementaci zmiňované aplikace, tak je doporučeno pro detailní porozumění nahlížet do zdrojových kódů aplikace. Nebudeme se však zabývat kompletním popisem toho, jak byla aplikace implementována, ale pouze těmi částmi, které jsou pro probíranou problematiku zajímavé.

6.2 Zvolené programovací jazyky, technologie a knihovny

6.2.1 Serverová část

Do této chvíle jsme se bavili pouze o aplikaci. I když není serverová část zmíněna v zadání, pro vytvoření funkčního celku je její implementace nezbytná, a proto se o ni ve stručnosti zmíníme. Serverovou část budeme v rámci aplikace označovat taktéž jako webové API.

Serverovou část jsem se rozhodl implementovat v programovacím jazyce PHP. Prvním důvodem je moje osobní zkušenost s tímto programovacím jazykem a druhým důvodem je to, že se PHP na serverové části webových aplikací využívá velmi často, má velkou komunitu a spoustu knihoven, které je možné využít. Serverová část kromě PHP využívá databázového systému PostgreSQL.

Jakožto architekturu rozhraní API jsem zvolil REST [21], což je jedna z nejrozšířenějších architektur pro tvorbu rozhraní webového API.

Mezi nejdůležitější knihovny, jež jsou v serverové části použity, jsou Symfony, Doctrine, API Platform a Lexik JWT Authentication.

6.2.2 Aplikační část

Základním stavebním kamenem aplikační části jsou již dříve zmiňované jazyky HTML, CSS a JavaScript. Kromě standardního CSS využívá aplikační část preprocesoru Sass [22], který rozšiřuje CSS o řadu věcí jako jsou proměnné, vnořování, dědění, mixins atd. JavaScriptová část využívá standardu EcmaScript 2020.

Aplikační část, na rozdíl od té serverové, využívá širokou škálu závislostí, a to jak vývojových, tak produkčních.

Vývojové závislosti

Úplným základem pro vývoj, testování a sestavování aplikace je běhové prostředí Node.js [23], označováno taktéž jako Node, a balíčkovací systém NPM (anglicky Node Package Manager) [24].

Na těchto dvou technologiích je postaven Webpack [25]. Webpack je tzv. bundler. Ten zpracovává veškerý JavaScriptový kód projektu a společně s dalšími médii, jako jsou styly a obrázky, vytváří výsledný JavaScriptový balíček spustitelný v konkrétním běhovém prostředí. V našem případě na webu, případně v rámci Electron aplikace.

Vzhledem k tomu, že standard EcmaScript 2020 není v rámci Node.js ani webových prohlížečích podporován, tak je součástí Webpack konfigurace nástroj Babel [26]. Babel podle své konfigurace překládá kód do starších verzí JavaScriptu, přidává do kódu polyfily a provádí další úpravy kódu podle toho, pro jakou cílovou verzi JavaScriptu ho nakonfigurujeme.

Pro potřeby PWA aplikace a jejího fungování bez přípojení k internetu aplikace využívá Offline Plugin [27] pro Webpack. Ten automaticky vytváří a registruje soubor služby běžící na pozadí (anglicky Service Worker) na základě konfiguračního souboru, což výrazně zjednodušuje jinak poměrně komplikovanou práci s touto službou.

Pro sestavení Electron aplikace využíváme nástroj Electron Builder [28], který sestavuje a balíčkuje spustitelné soubory pro jednotlivé operační systémy.

Naše aplikace využívá další řadu vývojových nástrojů a závislostí, jež je zbytečné tu zmiňovat. V případě zájmu doporučuji nahlédnout do zdrojových kódů samotné aplikace.

Produkční závislosti

Produkční závislosti, nebo také aplikační závislosti, jsou tím nejdůležitějším prvkem, který formuje architekturu a směr, jakým se při samotné implementaci vydáváme. Z toho důvodu si ty hlavní představíme detailněji.

React

React [29] je knihovna pro tvorbu uživatelského rozhraní. Základem Reactu jsou znovupoužitelné komponenty, jež zapouzdřují konkrétní funkcionalitu a jejich skládáním vzniká výsledné uživatelské rozhraní aplikace. Komponenty

si udržují vlastní stav (anglicky state) a zvenčí přijímají vlastnosti (anglicky props).

Komunikace napříč komponentami probíhá pomocí jednosměrného datového toku (anglicky One-Way Data Flow). Datový tok v rámci hierarchie probíhá tedy z nejvyšší do nejnižší komponenty a komunikace opačným směrem probíhá pomocí zpětných volání (anglicky callback function).

React využívá principu virtuálního DOMu, který udržuje stav React komponent v paměti a synchronizuje se s DOMem prohlížeče.

React Router

React Router [30] je knihovna pro dynamické směrování klientských aplikací napsaných v Reactu. React Router využívá komponentového přístupu, tudíž je směrování definováno pomocí hierarchie navigačních komponent. Při samotném směrování nedochází k přechodu mezi stránkami s přenačtením stránek, React Router však stav stránky prohlížeči podsune.

Samotné směrování má několik variant, přičemž v aplikaci jsou využity pouze dvě z nich. Browser Router, který využívá standardního směrování, který známe z adresních řádků prohlížečů, kdy se stránky hierarchicky oddělují lomítkem a Hash Router, který zapisuje cestu za hash.

Vzhledem k tomu, že Electron využívá fyzického souborové systému, tak se v jeho případě používá zmíněný Hash Router.

Redux, Immutable, Reselect

Redux [31] je knihovna pro udržování a správu stavu aplikace, která využívá architektury Flux [32].

Jak Redux funguje? Uživatelské rozhraní vyvolá akci (anglicky Action), která je odeslána do reduktoru (anglicky Reducer). Reduktor provede změnu úložiště (anglicky Store). Z úložiště poskytujeme stav (anglicky State), který předáváme komponentám uživatelského rozhraní.

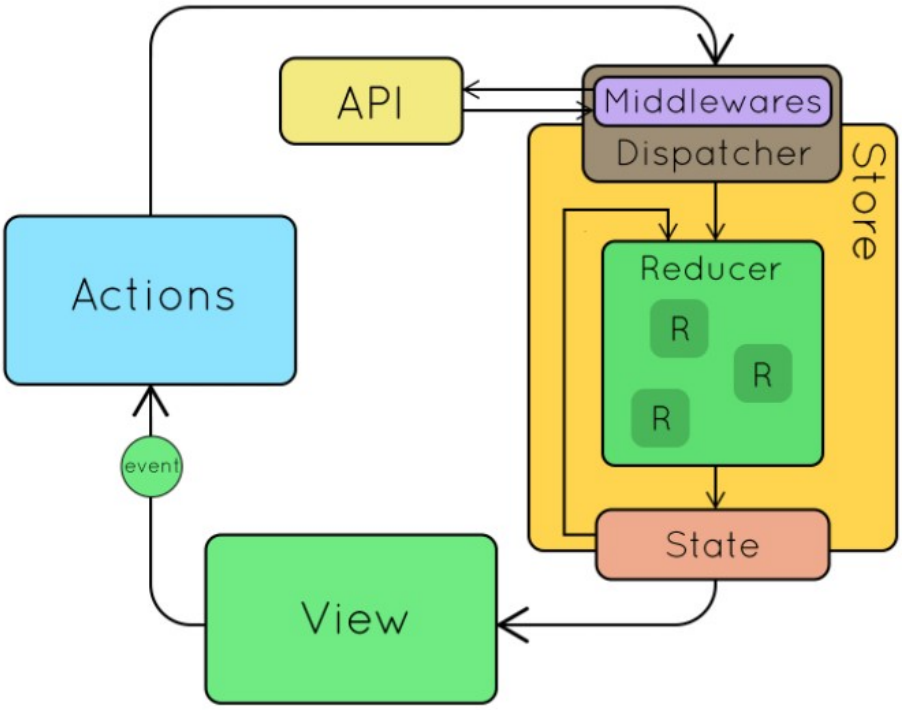
Aplikace tento princip však rozšiřuje pomocí dalších knihoven jako např. Immutable, Thunk a Reselect. Knihovna Immutable zajišťuje neměnnost dat celého úložiště tak, aby nemohlo dojít k nechtěné modifikaci dat a navíc dokáže velmi efektivně pracovat se samotnými daty. Thunk rozšiřuje Redux o možnost odesílat akce, které nejsou definovány jako standardní Redux objekty, ale umožňuje odesílat i akce definované jako funkce, jejichž výsledky jsou teprve poté zpracovány reduktorem. O zmíněnou logiku se stará Thunk middleware. Tento princip umožňuje asynchronní volání, např. při volání webového API. Knihovna Reselect pak slouží pro vyzvedávání dat z úložiště Reduxu a vzhledem k tomu, že využívá principů memoizace [33], tak pracuje s daty taktéž velmi efektivně.

Redux API middleware

Redux API middleware [34] je rozšiřujícím middlewarem výše zmiňovaného Reduxu, který slouží pro komunikaci s webovým API.

Místo běžné akce stačí odeslat RSAA akci (anglicky Redux Standard API-calling action), která je definována v rámci Redux API middleware. Middleware tuto akci zachytí, provede požadavek na webové API a zavolá reduktor s FSA akcí (anglicky Flux Standard Actions), kterou můžeme zpracovat dle potřeby. FSA akce jsou tří typů: čekající požadavek, úspěšná odpověď, neúspěšná odpověď.

Schéma Reduxu se zapojeným Redux API middlewarem můžeme vidět na následujícím obrázku.



Obrázek 6.1: Schéma Reduxu se zapojeným Redux API middleware [2]

Material UI

Material UI je knihovna komponent využívající Material Design od společnosti Google. Celá aplikace využívá této knihovny a to ať už se jedná o prvky pro rozložení obrazovky, navigační či formulářové prvky.

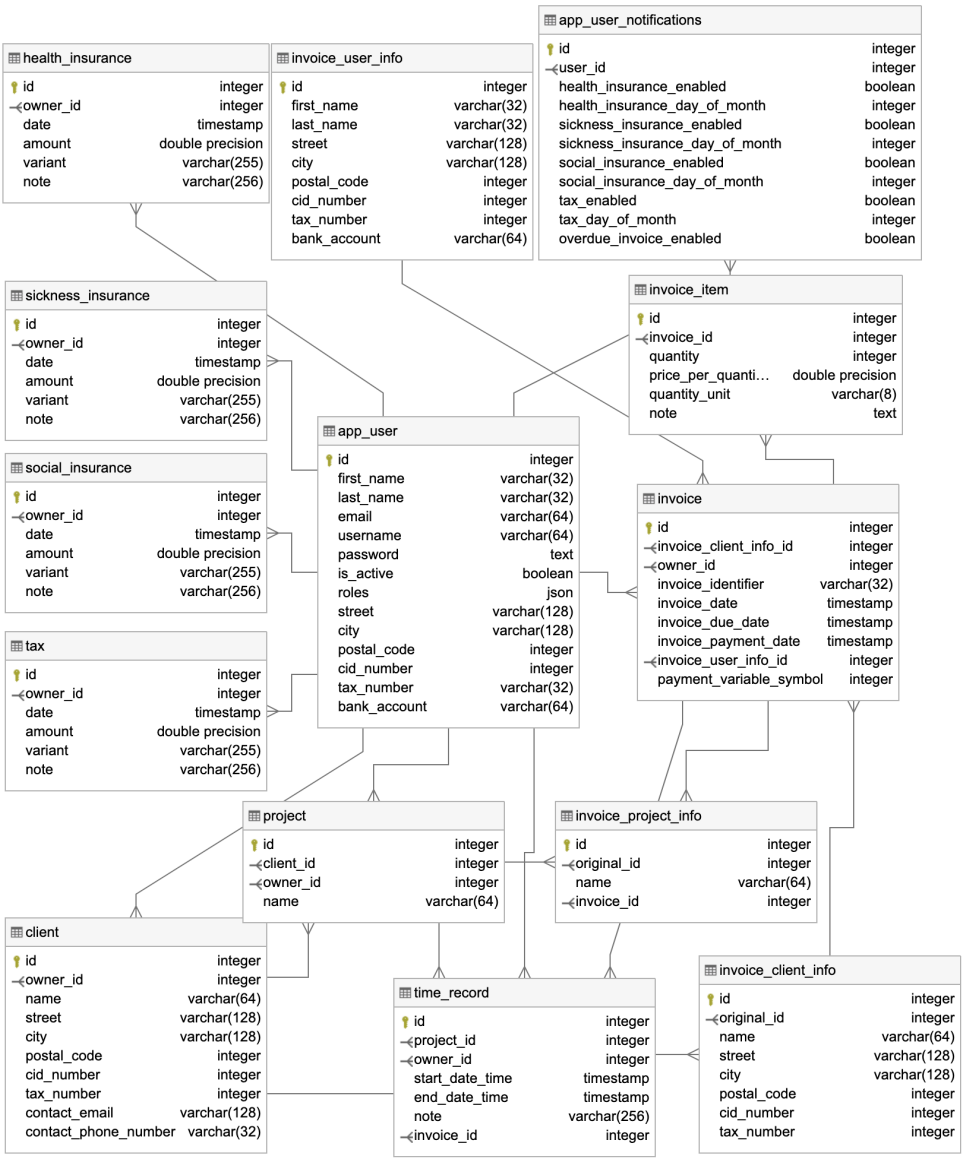
Ostatní knihovny

Kromě výše uvedených knihoven aplikace využívá řadu dalších knihoven, které však nemají dopad na celkovou architekturu aplikace. Jedná se kupříkladu o knihovny pro generování PDF, grafů či funkce pro validování dat.

6.3 Databázové schéma

Dle funkčních a nefunkčních požadavků aplikace byl navržen diagram databáze, na jehož základě později v rámci webového API vznikly jak třídy, tak samotné databázové schéma.

Databázové schéma je samovysvětlující, a tudíž bude uvedeno bez dalšího popisu jednotlivých tříd.



Obrázek 6.2: Databázové schéma

6.4 Obrazovky aplikace

Na základě popisu aplikace a jejích funkčních a nefunkčních požadavcích, jež byly uvedeny v předchozích kapitolách, spolu s databázových schématem uvedeným v předchozí podkapitole, byste mohli mít o samotné aplikaci již nějaké představy. Níže si tedy představíme konkrétní obrazovky, abychom v navazujících podkapitolách mohli rozebírat konkrétní pasáže.

6.4.1 Přihlášení

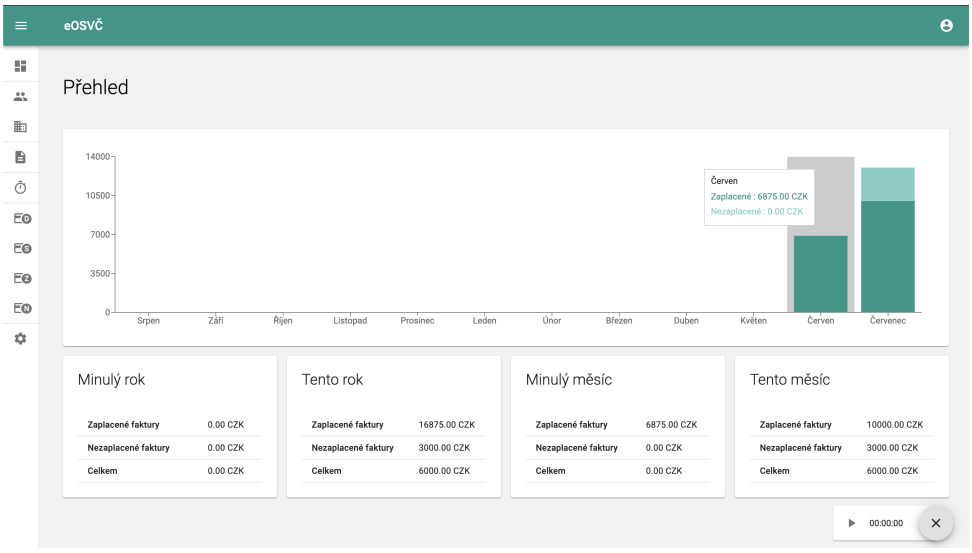
Stránka Přihlášení umožňuje přihlášení se do aplikace, popř. přechod na stránku Vytvoření účtu.

6.4.2 Vytvoření účtu

Stránka Vytvoření účtu umožňuje vytvořit nový účet, a to ve dvou krocích. Zadání přihlašovacích a osobních údajů v prvním kroku. V druhém kroku pak zadání fakturačních údajů pro pozdější potřeby vystavování faktur. Jak při úspěšném vytvoření účtu, tak při odchodu ze stránky, se vždy dostanete na stránku Přihlášení.

6.4.3 Přehled

Stránka Přehled je hlavní stránkou celé aplikace, na níž jste přesměrováni po úspěšném přihlášení. Stránka zobrazuje přehledy o vystavených a zaplacených fakturách, a to ve dvou zobrazeních.



Obrázek 6.3: Stránka Přehled

Prvním zobrazením je sloupcový graf, který ukazuje výši vystavených a zaplacených faktur, a to pro posledních 12 měsíců. Druhým zobrazením

jsou kartičky s totožnými údaji, zobrazují však souhrny za aktuální měsíc, předchozí měsíc, aktuální rok a předchozí rok.

Jak po kliknutí na sloupec grafu, tak po kliknutí na detail kartičky, jste přesměrování na stránku Faktury s přednastaveným filtrem.

6.4.4 Klienti

Stránka Klienti poskytuje tabulkový výpis klientů řazený abecedně podle jména klienta. Stisknutím tlačítka přidat otevřete modálové okno pro vytvoření nového klienta. V jednotlivých řádcích tabulky pak naleznete tlačítka na zobrazení modálové okna umožňující úpravu klienta a tlačítko pro odstranění klienta.

Smazání klienta je možné pouze v případě, kdy na něm nejsou závislé další entity, jako třeba projekty, faktury apod.

6.4.5 Projekty

Stránka Projekty poskytuje tabulkový výpis projektů řazený abecedně podle názvu projektu. Akce pro přidání, úpravu a smazání jsou totožná se stránkou Klienti.

Projekt je možné přidat v momentě, kdy je vytvořený alespoň jeden klient, ke kterému lze nový projekt přiřadit. Smazání projektu je možné pouze v případě, kdy na něm nejsou závislé další entity, jako třeba faktura, záznam odpracovaného času apod.

6.4.6 Faktury

Stránka Faktury poskytuje tabulkový výpis faktur řazený sestupně podle data vystavení faktury. Stisknutím tlačítka přidat přejdete na stránku Přidat fakturu. V jednotlivých řádcích tabulky pak najdete tlačítka pro zobrazení detailu a editace faktury, přičemž každé z tlačítek vyvolá přechod na zvláštní stránku. Tlačítko pro smazaní faktury smaže fakturu okamžitě.

Po stisknutí tlačítka filtr dojde k zobrazení filtru, pomocí něhož je možné filtrovat výpis na základě data vystavení faktury, případně dle klienta, jemuž byla faktura vystavena.

Fakturu je možné přidat v momentě, kdy je vytvořený alespoň jeden projekt a klient, ke kterým lze novou fakturu přiřadit.

6.4.7 Přidat fakturu, Upravit fakturu

Stránky Přidat fakturu a Upravit fakturu jsou totožné. Jak z názvu vyplývá, tak slouží pro vytváření a úpravu faktur.

Při otevření stránky dojde k automatickému předvyplnění údajů dodavatele, tedy údajů, které uživatel zadal při vytvoření účtu.

Fakturu vystavujeme vždy pro konkrétního odběratele, po jehož vybrání dojde k předvyplnění jeho údajů. Zároveň se nám načtou projekty spojené

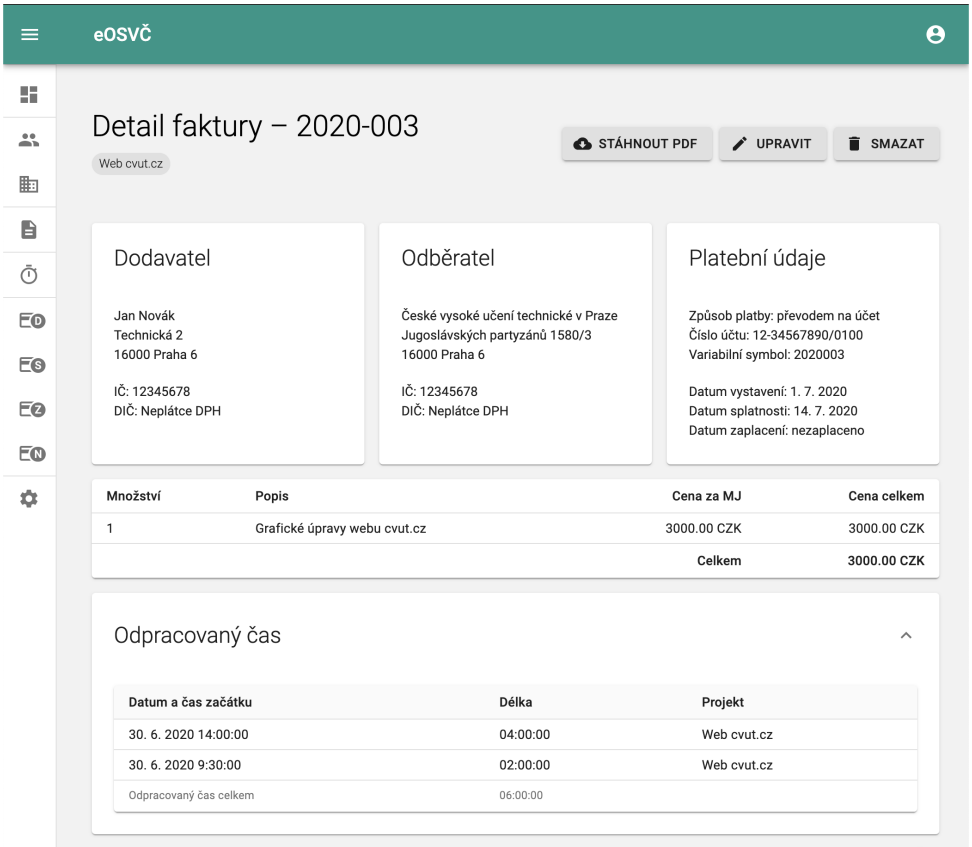
s vybraným klientem, přičemž je nezbytné vybrat alespoň jeden projekt, ke kterému se faktura vztahuje.

V závislosti na vybraných projektech dojde k načtení všech záznamů odpracovaného času, které ještě nejsou přiřazené k žádné faktuře. Na základě vybraných záznamů dojde k vypočítání celkového odpracovaného času. Po vytvoření faktury na stránce Odpracovaný čas nebude možné již měnit ty záznamy, které jsou k nějaké faktuře již přiřazené, ale přibude k nim odkaz na konkrétní fakturu, v rámci které byly hodiny fakturovány.

Mimo tyto údaje je nutné přidat alespoň jednu položku faktury a doplnit ty platební údaje, které nebyly automaticky doplněny.

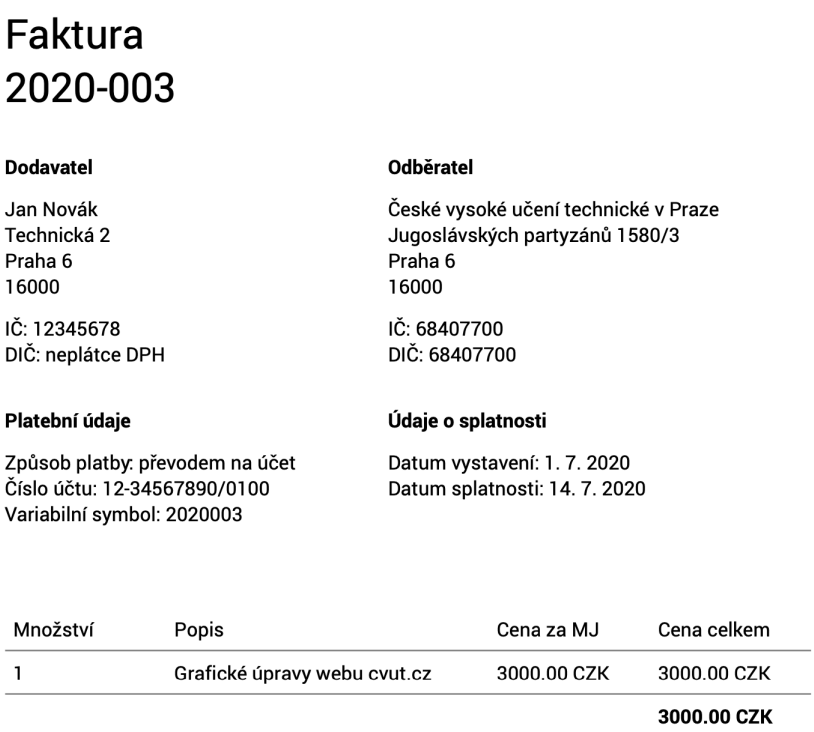
6.4.8 Detail faktury

Stránka Detail faktury obsahuje stejná data, jež jsou vyplňována na stránce Přidat fakturu a Upravit fakturu s tím rozdílem, že je nelze upravovat.



Obrázek 6.4: Stránka Detail faktury

Kromě tlačítka pro přechod na stránku Úprava faktury a tlačítka pro smazání faktury, je zde přítomné i tlačítko Stáhnout PDF. Po jeho stisknutí dojde na základě dostupných údajů k vygenerování PDF, které bude obsahovat fakturu a v případě, že jsou k faktuře přiřazeny nějaké záznamy odpracovaného času, tak i přehled záznamů odpracovaného času.



Faktura
2020-003

Dodavatel

Jan Novák
Technická 2
Praha 6
16000

IČ: 12345678
DIČ: neplátce DPH

Odběratel

České vysoké učení technické v Praze
Jugoslávských partyzánů 1580/3
Praha 6
16000

IČ: 68407700
DIČ: 68407700

Platební údaje

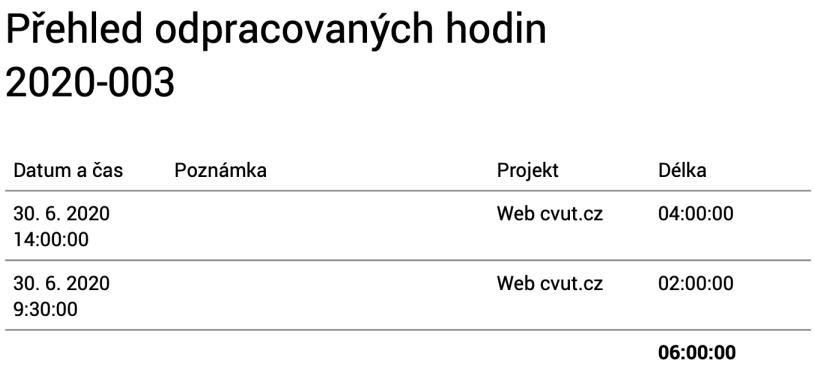
Způsob platby: převodem na účet
Číslo účtu: 12-34567890/0100
Variabilní symbol: 2020003

Údaje o splatnosti

Datum vystavení: 1. 7. 2020
Datum splatnosti: 14. 7. 2020

Množství	Popis	Cena za MJ	Cena celkem
1	Grafické úpravy webu cvut.cz	3000.00 CZK	3000.00 CZK
			3000.00 CZK

Obrázek 6.5: Vygenerované PDF faktury



Přehled odpracovaných hodin
2020-003

Datum a čas	Poznámka	Projekt	Délka
30. 6. 2020 14:00:00		Web cvut.cz	04:00:00
30. 6. 2020 9:30:00		Web cvut.cz	02:00:00
			06:00:00

Obrázek 6.6: Vygenerované PDF odpracovaného času k faktuře

6.4.9 Odpracovaný čas

Stránka Odpracovaný čas poskytuje tabulkový výpis záznamů odpracovaného času řazený sestupně podle času začátku. Stisknutím tlačítka přidat otevřete modálové okno pro vytvoření nového záznamu odpracovaného času ke konkrétnímu projektu. V jednotlivých řádcích tabulky pak naleznete tlačítka na zobrazení modálové okna umožňující úpravu záznamu či jeho smazání.

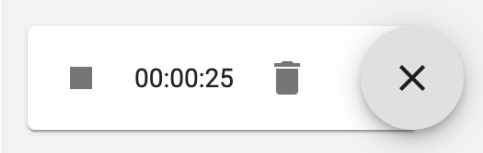
V případě, kdy je záznam odpracovaného času součástí vytvořené faktury,

tak místo tlačítek pro úprava a smazání záznamu je zobrazeno tlačítko pro přechod na detail příslušné faktury.

Součástí stránky je filtr, který umožňuje filtrovat výpisy podle data a podle projektu. Ve výchozím nastavení filtr zobrazuje záznamy za posledních 30 dní, a to z důvodu, aby nedošlo k přetížení webového API a ke zbytečně velkým přenosům dat.

Součástí funkcionality odpracovaného času jsou i stopky, jež nejsou zobrazené pouze na stránce odpracovaného času, ale na všech stránkách aplikace. Stopky po spuštění začínají počítat odpracovaný čas, a to i v případě, kdy úplně zavřete okno prohlížeče nebo Electron aplikace. Poté je možné stopky resetovat nebo je zastavit a pomocí modálového okna pro přidání záznamu odpracovaného času měřený čas uložit.

Electron aplikace poskytuje další podporu pro obsluhu tohoto časovače, o níž se zmíníme později.



Obrázek 6.7: Časovač

6.4.10 Daně, Sociální pojištění, Zdravotní pojištění, Nemocenské pojištění

Každá ze stránek Daně, Sociální pojištění, Zdravotní pojištění a Nemocenské pojištění eviduje jiné typy plateb, ale jejich rozložení, včetně políček formulářů, je stejné.

Data jsou řazena sestupně podle data platby. Stisknutím tlačítka přidat otevřete modálové okno pro vytvoření nové platby. V jednotlivých řádcích tabulky pak naleznete tlačítka na zobrazení modálové okna umožňující úpravu platby a tlačítko na odstranění platby.

Tyto stránky v současné verzi aplikace slouží pouze k evidenci plateb, žádná další funkcionalita není na tyto data navázána.

6.4.11 Nastavení

Stránka Nastavení umožňuje momentálně nastavovat několik upozornění. Prvním upozorněním je faktura po splatnosti, jež je možné vypnout nebo zapnout. Další čtyři upozornění na platbu záloh na dani, sociálním, zdravotním a nemocenském pojištění je možno buď vypnout nebo zapnout a nastavit konkrétní den, kdy se má na danou věc upozornit.

Upozornění na fakturu po splatnosti je vyvoláno vždy po spuštění aplikace a dále v pravidelných intervalech. Upozornění na platbu záloh se děje stejně, ale pouze v konkrétně nastavený den. To, zda je nějaká platba daný měsíc zaevidována či nikoliv, nehraje na zobrazení upozornění roli. Logika plateb záloh je totiž netriviální.

6.4.12 Účet

V uživatelském menu v záhlaví webu je možné přejít na stránku Účet. Na stránce Účet je možné upravit osobní, kontaktní a fakturační údaje, které se zadávají při vytváření účtu a je zde mimo jiné možné změnit i heslo k účtu.

6.5 Implementace klíčových částí aplikace

Dostáváme se ke klíčové podkapitole. V prvních dvou kapitolách jsme získali teoretické znalosti v oblasti technologií PWA a Electron, v minulé kapitole jsme se dozvěděli o popisu aplikace, jejich funkčních a nefunkčních požadavcích a v aktuální kapitole jsme se dozvěděli o použitých programovacích jazycích, technologiích a knihovnách a dokonce o podobě konkrétních obrazovek.

V této podkapitole se již budeme věnovat pouze vybraných klíčových částí aplikace.

6.5.1 Komunikace s webovým API

Nejvíce klíčovou částí aplikace je komunikace s webovým API. Komunikace je založena na standardu REST [21] (anglicky Representational State Transfer), který využívá protokol HTTP.

Jednotlivé operace jsou založeny na kombinaci HTTP metod GET, POST, PUT, DELETE, a URI identifikátoru (anglicky Uniform Resource Identifier) identifikujícího jednotlivé zdroje. Ukážeme si příklady kombinací URL a HTTP metod volání webového API:

- /clients [GET] - Získá výpis klientů
- /clients [POST] - Vytvoří nového klienta (data klienta jsou součástí těla požadavku ve formátu JSON)
- /clients/1 [GET] - Získá klienta s ID=1
- /clients/1 [PUT] - Upraví klienta s ID=1 (data klienta jsou součástí těla požadavku ve formátu JSON)
- /clients/1 [DELETE] - Odstraní klienta s ID=1

Implementace webového API počítá vždy s tím, že jsou data v těle uvedena ve formátu JSON.

Vzhledem k dříve uvedenému databázovému schématu je patrné, že všechny třídy mají vazbu na třídu reprezentujícího uživatele. Proto implementace webového API počítá s tím, že se URI identifikátor přihlášeného uživatele neposílá. Webové API si přihlášeného uživatele doplní samo.

Webové API informaci o přihlášeném uživateli získá z JWT tokenu (anglicky JSON Web Token) [35], který je posílán v HTTP hlavičce Authorization. Ten aplikace do všech HTTP požadavků vkládá pomocí zvláštního middlewaru, který JWT token získá z lokálního úložiště prohlížeče (anglicky Local Storage).

Pro to, aby měla aplikace k dispozici JWT token, je zapotřebí zavolat příslušný koncový bod webového API. Ten se volá při přihlášení do aplikace, kdy posílá webovému API uživatelské jméno a heslo, a ten v případě úspěšného ověření zašle nově vygenerovaný JWT token.

Samotnou komunikaci s webovým API v rámci aplikace zaštiťuje knihovna Redux API middleware s knihovnou Redux a dalšími dříve zmíněnými knihovnami. Pro detailní popis doporučuji projít zdrojové kódy aplikace.

6.5.2 Offline režim

Ačkoliv se očekává, že při práci s aplikací budete připojeni k Internetu, je i tak možné částečně používat aplikaci v režimu bez připojení k Internetu. K dosažení tohoto chování využívá aplikace dvě služby, a to Service Worker API a Cache API. Aplikace dále využívá Offline Plugin pro Webpack, který stojí nad těmito službami.

V rámci PWA využíváme Offline plugin pro obsluhu veškerých statických souborů, tedy HTML, CSS a JavaScript souborů, dále pak obrázků, ikon a všech dalších souborů, jež jsou pro běh aplikace nezbytné. Electron aplikace tohoto pluginu nevyužívá, protože jsou všechny statické soubory součástí spustitelného souboru.

Offline plugin na základě své konfigurace vygeneruje soubor služby běžící na pozadí sw.js, který se při spuštění aplikace zaregistruje a ihned stáhne příslušné soubory do cache. Z ní je pak aplikace později načítána. Offline plugin generuje do kódu kromě samotné funkcionality také otisky souborů podle kterých rozhoduje, zda ze serveru stahovat novější soubory či nikoliv.

Druhou úrovní je pak cachování síťových požadavků na webové API. V rámci Electron aplikace se tato služba běžící na pozadí registruje manuálně, v rámci PWA je Webpack sloučí dohromady s předchozí zmiňovaným souborem sw.js.

Cachování síťových požadavků tvoří jádro offline módu. Všechny síťové požadavky na webové API chodí skrz službu běžící na pozadí a po úspěšném provedení požadavku si jeho odpověď uloží do cache. Dokud je připojení k internetu k dispozici, tak se do cache požadavky pouze ukládají, ale aplikace využívá odpovědi poslané z webového API. Pouze v momentě, kdy je připojení k internetu ztraceno, se služba běžící na pozadí pokusí načíst data z cache, a buď data vrátí nebo vrátí chybu.

Cachování se děje pouze u požadavků s HTTP metodou GET, tedy pouze u operací pro získání výpisu dat. Operace pro přidávání, úpravu a mazání jsou v offline režimu nedostupné. Z toho vyplývá, že jsou data v cache uložena pouze v průběhu sezení a je nezbytné být do aplikace přihlášen. Odhlášením se pak obsah cache se síťovými požadavky na webové API vyprázdní.

Níže je uveden obsah služby běžící na pozadí, která se stará o cachování síťových požadavků webového API. Konfigurace Offline pluginu je rozsáhlá, a proto doporučuji nahlédnout do zdrojových kódů aplikace.

```
import { API_URL } from '../config/config';

self.addEventListener('fetch', (event) => {
  if (
    !event.request.url.startsWith(`${API_URL}/`)
    || event.request.method !== 'GET'
  ) {
    return;
  }

  event.respondWith(
    caches
      .match(event.request)
      .then((cachedResponse) => {
        if (cachedResponse && !navigator.onLine) {
          return cachedResponse;
        }

        return fetch(event.request)
          .then((response) => caches.open(
            'webpack-offline:e-osvc-api'
          )
            .then((cache) => {
              cache.put(event.request, response.clone());
              return response;
            }))
          .catch(() => cachedResponse);
      }),
  );
});
```

6.5.3 Notifikace

Notifikace se využívají v rámci upozornění na fakturu po splatnosti a platbu pravidelných záloh. Konkrétně se využívá Notifications API, v rámci Electronu označované taktéž jako HTML 5 Notifications API.

I když je na tuto funkcionalitu vhodnějším kandidátem Push API (taktéž označováno jako Push Notifications), tak byla v této verzi aplikace, především z důvodu obtížné implementaci na straně API, které není hlavním požadavkem této bakalářské práce, zvolena varianta Notifications API. Budoucím vylepšením je tedy bezesporu přechod na Push API.

V rámci Electron aplikace jsou vyvolávány notifikace v rámci vykreslovacího procesu, a tudíž se používá stejné API jako v případě PWA.

Obě uvedená API bohužel nejsou podporována ve všech často používaných prohlížečích. Jedná se o prohlížeč Internet Explorer, který však není mezi podporovanými prohlížeči touto aplikací a prohlížeč Safari na systémech iOS a iPadOS.

Dalším problémem je i samotné vytváření notifikací. Kvůli měnícímu se Notifications API je vytváření rozdílné například i v rámci prohlížeče Google Chrome, kdy desktopová verze a Android verze vyžadují jiné konstrukty. Řešení uvedeného problému je dostupné v následující ukázce zdrojového kódu.

```
export const showNotification = (message) => {
  if (!('Notification' in window)) {
    return null;
  }

  const showNotificationImpl = () => {
    const data = {
      body: message,
      icon: '/images/favicon-96x96.png',
    };

    try {
      return navigator.serviceWorker.ready.then((reg) => {
        if ('showNotification' in reg) {
          return reg.showNotification('eOSVČ', data);
        }

        return new Notification('eOSVČ', data);
      });
    } catch (e) {
      return new Notification('eOSVČ', data);
    }
  };

  if (Notification.permission === 'granted') {
    return showNotificationImpl();
  }

  if (Notification.permission !== 'denied') {
    return Notification.requestPermission().then((perm) => {
      if (perm === 'granted') {
        return showNotificationImpl();
      }

      return null;
    });
  }

  return null;
};
```

6.5.4 Electron aplikace – Základní principy

Electron aplikace, resp. její část, která běží ve vykreslovacím procesu, má oproti základní webové aplikaci, tedy i PWA aplikaci, jinou konfiguraci Webpacku, a to především z důvodu, že Electron aplikace má jiný vstupní bod než základní webová aplikace. Důvodem je především rozdílná registrace služby běžící na pozadí. Zatímco Webpack konfigurace PWA aplikace vyžaduje pro svůj běh Offline plugin, Electron aplikace tento plugin nevyžaduje a registruje pouze část služby běžící na pozadí, která se stará o cachování síťových požadavků webového API.

Implementace hlavního vykreslovacího procesu Electron aplikace se nijak zvláště neliší od teorie a ukázek zdrojových kódů, které byly součástí kapitoly věnující se Electronu, a proto zde nebudou uvedeny.

Oproti implementaci webové aplikace, tedy PWA aplikace, má Electron aplikace jiný vstupní HTML soubor, který neobsahuje konstrukty, jež jsou nezbytné pro běh PWA aplikace a importuje jiný JavaScriptový soubor samotné aplikace. Důvodem je výše uvedená rozdílná konfigurace Webpacku.

Electron naopak vyžaduje soubor hlavního vlákna Electronové aplikace electron.js, který se stará o obsluhu hlavní vlákna, vykreslení okna aplikace a obsluhu dalších API.

Tím, že je celá aplikace koncipována tak, že je kód samotné webové aplikace společný jak pro PWA, tak pro Electron, tak zde nastává problém, že do samotné webové aplikace potřebujeme občas přidat část kódu, která je spustitelná pouze v rámci Electronu. Toho docílíme dvěma kroky.

První krokem je konfigurace Webpacku. V rámci konfigurace Webpacku inicializujeme proměnnou `IS_ELECTRON`, jež nám říká, jestli se jedná o sestavení pro Electron či nikoliv. Tato globální proměnná je poté k dispozici v kódu samotné aplikace a pomocí jednoduché podmínky můžeme provádět určité operace v závislosti na této hodnotě.

Druhým krokem je dynamické importování modulů. Pokud bychom totiž nevyužili dynamického importování, tak by se moduly specifické pro Electron vložily i do sestavení PWA aplikace, a ta by skončila s chybou.

Kombinací výše uvedených kroků Webpack při procesování souborů dojde k tomu, že se v kódu nachází mrtvý kód, který se do výsledného sestavení webové aplikace vůbec nevloží.

Příkladem je např. registrace služby pro obsluhu časovače, která komunikuje se systémem pomocí Tray API a TouchBar API:

```
if (IS_ELECTRON) {
  import('./services/timerService').then((timerService) => {
    timerService.registerTimerService(store);
    return timerService;
  });
}
```

6.5.5 Electron aplikace – Komunikace mezi procesy

V návaznosti na ukázku registrace služby pro obsluhu časovače si detailněji rozebereme komunikaci mezi vykreslovacím a hlavním procesem Electronu a volání pokročilých API.

Vykreslovací proces

Kód služby pro obsluhu časovače, jež jsme v minulé podkapitole zavolali, vypadá následovně:

```
import { ipcRenderer } from 'electron';
import {
  resetTimer,
  selectTimer,
  setTimer,
} from '../resources/timeRecord';
import { getTimeDifferenceString } from './dateTimeService';

export const registerTimerService = async (store) => {
  ipcRenderer.on('startTimer', () => {
    store.dispatch(setTimer(Date.now()));
  });

  ipcRenderer.on('resetTimer', () => {
    store.dispatch(resetTimer());
  });

  return setInterval(() => {
    const timer = selectTimer(store.getState());

    if (timer) {
      ipcRenderer.invoke(
        'setTimer',
        getTimeDifferenceString(timer, Date.now())
      );
    }
  }, 1000);
};
```

Služba přijímá proměnnou `store`, která je objektem knihovny Redux, který poskytuje funkci `dispatch`, pomocí níž je možné vyvolávat akce v rámci Reduxu a funkci `getState`, jež vrací stav úložiště.

Pomocí funkce `ipcRenderer.on` zaregistrujeme posluchač pro příjem zpráv z hlavního vykreslovacího procesu. V této ukázce se jedná o zprávy typu `startTimer` a `resetTimer` a funkce, jež se o zpracování daných zpráv starají. V rámci nich pak voláme akce, které se starají o spuštění a resetování stavu časovače v rámci aplikace.

Opačným směrem pak komunikuje kód uvnitř funkce `setInterval`, který je volán v intervalu 1 sekundy. Každou sekundu se kontroluje zda časovač běží či nikoliv, a pokud ano, tak pomocí funkce `ipcRenderer.invoke` pošle hlavnímu procesu zprávu typu `setTimer` s aktuálním časem časovače.

Hlavní proces

V hlavním procesu aplikace je práce se zprávami obdobná.

```
ipcMain.handle('setTimer', (event, timer) => {
  touchBarService.handleSetTimer(mainWindow, electronStore, timer);
  trayService.handleSetTimer(mainWindow, electronStore, timer);
});

ipcMain.handle('resetTimer', () => {
  touchBarService.handleResetTimer(mainWindow, electronStore);
  trayService.handleResetTimer(mainWindow, electronStore);
});
```

Pomocí funkce `ipcMain.handle` zaregistrujeme poslouchač pro příjem zpráv z vykreslovacího procesu. V této ukázce se jedná o zprávy typu `setTimer` a `resetTimer` a funkce, jež se o zpracování daných zpráv starají. V rámci nich pak voláme služby, které se starají v tomto případě o komunikaci se systémovými Tray API a TouchBar API.

Pouze v případě posílání zprávy z hlavního procesu do vykreslovacího procesu je situace mírně odlišná. V tomto případě voláme funkci pro odeslání zprávy nad instancí konkrétního okna třídy `BrowserWindow`.

Funkce pro volání takové zprávy pak vypadá následovně:

```
mainWindow.webContents.send('resetTimer');
```

6.5.6 Electron aplikace – Práce s pokročilými API

V návaznosti na předchozí podkapitoly si ukážeme koncept práce s pokročilými API v rámci hlavního procesu. Vzhledem k tomu, že jsme si jednotlivá API pro komunikaci se systémem představovali ve zvláštní kapitole věnující se Electronu, tak představím pouze obsluhu TouchBar API, protože obsluha dalších API by se prováděla obdobným způsobem.

Ve vstupním souboru hlavního procesu electron.js je definován objekt uschovávající instanci třídy TouchBar, instance jednotlivých komponent a další data, která v rámci služby využíváme.

```
const electronStore = {
  touchBar: {
    data: { isStarted: false },
    instance: null,
    parts: {/* Definice vynechána */},
  },
};
```

Registraci samotné služby pak provedeme po vytvoření okna vykreslovacího procesu a předáme ji zmiňovanou proměnnou `electronStore` a instanci okna `mainWindow`.

```
import * as touchBarService from './electron/touchBarService';

const createWindow = () => {
  mainWindow = new BrowserWindow({
    height: 960,
    webPreferences: { nodeIntegration: true },
    width: 1600,
  });
  mainWindow.loadFile('./public/index.electron.html');

  touchBarService.init(mainWindow, electronStore);
};

app.whenReady().then(createWindow);
```

V rámci inicializace samotné třídy vytvoříme instanci jednotlivých tlačítek a popisků, které uložíme do příslušného místa proměnné `electronStore`. Na základě těchto částí vytvoříme instanci samotného TouchBaru, kterou taktéž uložíme do proměnné `electronStore` a přiřadíme ji instanci konkrétního okna aplikace. Celý kód inicializace vypadá následovně.

```
import { TouchBar } from 'electron';
import { getIcon } from './iconService';

const { TouchBarLabel, TouchBarButton } = TouchBar;

export const init = (mainWindow, electronStore) => {
  electronStore.touchBar.parts.startButton = new TouchBarButton({
    click: () => mainWindow.webContents.send('startTimer'),
    icon: getIcon('start_icon'),
  });
  electronStore.touchBar.parts.stopButton = new TouchBarButton({
    click: () => {
      mainWindow.show();

      return mainWindow.webContents.send('stopTimer');
    },
    icon: getIcon('stop_icon'),
  });
  electronStore.touchBar.parts.removeButton = new TouchBarButton({
    click: () => mainWindow.webContents.send('resetTimer'),
    icon: getIcon('remove_icon'),
  });
```

```
  electronStore.touchBar.parts.timer = new TouchBarLabel({
    label: '',
  });

  electronStore.touchBar.instance = new TouchBar({
    items: [
      electronStore.touchBar.parts.startButton,
    ],
  });

  mainWindow.setTouchBar(electronStore.touchBar.instance);
};
```

Jak lze vidět, práce s TouchBar API se od ukázky v rámci kapitoly věnující se Electronu nijak nezměnila, pouze byla vhodně zasazena do implementace samotné aplikace.

Když se vrátíme do podkapitoly věnující se komunikaci mezi procesy, tak tam nalezneme registraci posluchače zpráv chodících z vykreslovacího procesu, jejichž obsluhující funkce volá funkce služby pro práci s TouchBarem. Definice těchto funkcí je následovná.

```
export const handleSetTimer = (mainWindow, electronStore, timer) => {
  electronStore.touchBar.parts.timer.label = timer;

  if (!electronStore.touchBar.data.isStarted) {
    electronStore.touchBar.instance = new TouchBar({
      items: [
        electronStore.touchBar.parts.timer,
        electronStore.touchBar.parts.stopButton,
        electronStore.touchBar.parts.removeButton,
      ],
    });
    mainWindow.setTouchBar(electronStore.touchBar.instance);
    electronStore.touchBar.data.isStarted = true;
  }
};

export const handleResetTimer = (mainWindow, electronStore) => {
  electronStore.touchBar.instance = new TouchBar({
    items: [
      electronStore.touchBar.parts.startButton,
    ],
  });
  mainWindow.setTouchBar(electronStore.touchBar.instance);
  electronStore.touchBar.data.isStarted = false;
};
```

Uvedené funkce upravují zobrazení TouchBaru v momentě, kdy se spuštění nebo zastavení časovače provede v rámci aplikace a ne pomocí tlačítek na TouchBaru.

Vzhledem k tomu, že neexistuje doporučený způsob, jak strukturovat aplikace psané v Electronu, tak toto je jeden z možných způsobů, kterého je možno využít.

6.5.7 Práce s úložištěm a zabezpečení dat

V rámci webové aplikace se využívá práce s Web Storage API, konkrétně s variantou Local Storage, která ukládá data až doby, kdy je zase smažeme. Data se nesmažou ani při vypnutí prohlížeče, PWA či Electron aplikace.

Do Local Storage si aplikace ukládá dva typy dat. Prvním jsou aplikační upozornění, které potřebujeme zachovat i při opětovném načtení prohlížeče. Konkrétně se jedná o zprávy úspěchu nebo chybě při komunikaci s webovým API.

Druhý typem dat je JWT token, který slouží k ověření totožnosti vůči webovému API. Jak JWT token funguje již bylo zmíněno v rámci kapitoly o komunikaci s webovým API.

Důvodů, proč se neužívá varianta úložiště Session Storage nebo HTTP-Only Cookie, je několik. Při využití Local Storage však dochází ke snížení bezpečnosti, o čemž se ještě zmíníme.

Důvodem, proč aplikace nevyužívá Session Storage je ten, že potřebujeme mít token k dispozici i po zavření a opětovném otevření aplikace, abychom v případě offline módu mohli aplikaci využívat.

HTTP-Only Cookie se pak nevyužívá kvůli tomu, že na straně webového API využíváme již zmiňovaných PHP knihoven API Platform a Lexik JWT Authentication, které HTTP-Only Cookie v základu nevyužívají a dokumentují zasílání JWT tokenu v rámci hlavičky Authorization, jak jsme si již řekli dříve. Za druhé JWT token umí přenášet data, která využíváme v rámci aplikace, jako např. ID uživatele.

Důsledkem používání Local Storage oproti HTTP-Only Cookie je zvýšení rizika XSS útoku. XSS útok (anglicky Cross-site scripting) využívá slabých míst v aplikaci, kdy aplikace spustí škodlivý kód, který ukradne data (např. JWT token). Aplikace je však naprogramována tak, že escapuje veškerá vstupní i výstupní data. Škodlivý kód by tedy musel být součástí aplikačních knihoven, a proto jsou v aplikaci využívány známé knihovny, abychom pravděpodobnost takové chyby minimalizovali. V případě použití HTTP-Only Cookie by tento problém odpadl, neboť k takovému záznamu nemá JavaScript přístup.

Pro celkové zvýšení bezpečnosti a naplnění požadavku progresivních webových aplikacích, využívá jak aplikace, tak webové API, šifrovaného připojení přes HTTPS, aby v rámci komunikace nemohlo dojít k odposlechu dat.

Samotné webové API pak využívá mechanismů na zabezpečení jednotlivých koncových bodů tak, aby uživatel mohl pracovat pouze s entitami jež mu náleží. Webové API využívá k hashování hesel algoritmu bcrypt [36], jež v sobě obsahuje kryptografickou sůl a iterováním dochází ke zvýšení odolnosti vůči útokům hrubou silou.

6.6 Publikace aplikace

Webové API vyžaduje pro svůj běh webový server s PHP 7.4 a databázový systém PostgreSQL 11. Pro potřeby servírování základní webové aplikace však stačí pouze základní webový server, jež je schopný servírovat soubory HTML, CSS, JS, případně další přílohy jako třeba obrázky. Aplikace je z pohledu uživatele tedy dostupná pomocí webového prohlížeče pomocí URL.

PWA aplikace je taktéž dostupná v rámci webového prohlížeče pomocí URL. Díky tlačítku Přidat na plochu je možné PWA aplikaci nainstalovat na plochu zařízení. Detailní popis instalace je zmíněn v kapitole zabývající se progresivními webovými aplikacemi.

Situace u Electron aplikace je rozdílná. V rámci bakalářské práce jsou dodány instalační balíčky aplikace. Instalační balíčky aplikace využívají automatických aktualizací knihovny Electron Builder, které si stahují nové verze z GitHub repozitáře.

V rámci hlavního vlákna Electron aplikace zavolá následující funkci.

```
import { autoUpdater } from 'electron-updater';
autoUpdater.checkForUpdates();
```

Ta zkontroluje, zda je k dispozici nová verze. Pokud ano, tak vyvolá událost `update-available`, v opačném případě `update-not-available`.

Vzhledem k tomu, že je výchozím nastavením aktivované automatické stahování nové verze aplikace, tak je po dokončení stahování vyvolána událost `update-downloaded`, v případě chyby `error`.

Po stažení souboru je zobrazeno dialogové okno, které se uživatele zeptá, zda se může spustit instalace aplikace či nikoliv.

```
autoUpdater.on('update-downloaded', (info) => {
  const clickedButtonIndex = dialog.showMessageBoxSync({
    type: 'info',
    title: 'Nová verze aplikace',
    message: `Nová verze ${info.version} je k dispozici.
        Chcete ji nainstalovat?`,
    buttons: ['Ano', 'Ne']
  });

  if (clickedButtonIndex === 0) {
    autoUpdater.quitAndInstall(true, true);
  }
});
```

Tento mechanismus automatických aktualizací však momentálně není funkční v systému Windows, a to kvůli chybě v Electron Builder, která špatně zapisuje data do registrů systému Windows.

Tento problém se dá zatím opravit pouze manuálně, a to pomocí odstranění hodnoty `UninstallString` z následujícího `HKEY_CURRENT_USER` registru.

```
\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

6.7 Podpora aplikace

Podpora aplikace se dělí podle její varianty. Uvedeme si zvlášť podporu pro základní webovou aplikaci, PWA a Electron.

6.7.1 Podpora webové aplikace

Podpora prohlížečů se dělí mezi desktopové a mobilní systémy. V obou případech se však zabýváme minimálně podporou posledních dvou verzí prohlížeců, nebo jejichž využití je větší než 0.5%.

V rámci systémů Windows, Linux a MacOS se tedy bavíme o prohlížečích Google Chrome 79 a vyšší, Mozilla Firefox 74 a vyšší, Opera 65 a vyšší. Dále pak Microsoft Edge 18, a pak Microsoft Edge 79 a vyšší. V případě MacOS Safari 12.1 a vyšší. Prohlížeč Internet Explorer již není mezi podporovanými prohlížeči.

U systému Android je zajištěna podpora u Google Chrome 80 a Mozilla Firefox 68. V případě systémů iOS a iPadOS se jedná o Safari 12.2 a vyšší a Google Chrome 84. V případě systémů iOS a iPadOS však nejsou podporované upozornění, a to v žádném z prohlížečů.

Mimo uvedené prohlížeče aplikace funguje i v jiným prohlížečích, ale jejich podpora není zajištěna a aplikace v nich není otestována.

6.7.2 Podpora PWA aplikace

Podpora PWA aplikace se taktéž dělí mezi desktopové a mobilní systémy. Vzhledem k tomu, že PWA aplikace běží v rámci prohlížeče, tak se na jejich fungování vztahuje obdobná podpora s tím rozdílem, že PWA aplikace fungují pouze v rámci některých prohlížečů.

Na desktopových systémech Windows, Linux a MacOS funguje PWA aplikace pouze v rámci prohlížeče Google Chrome. V případě systému Android se bavíme o podpoře Google Chrome a Mozilla Firefox. V případě iOS a iPadOS se jedná pouze o Safari, a to se stejnými omezeními, tedy bez podpory notifikací.

6.7.3 Podpora Electron aplikace

Podpora Electron aplikace je zaručena na systémech Windows 10 a MacOS 10.14 a vyšší, na kterých proběhlo testování.

Dle oficiální dokumentace Electronu by měla být aplikace spustitelná i na systémech Windows 7, na MacOS 10.11 a vyšší a na systémech Ubuntu 12.04, Fedora 21 a Debian 8, ale zmiňovaná aplikace na nich nebyla testována, a proto nejsou oficiálně podporovány.

Kapitola 7
Testování

7.1 Cíl uživatelského testování

Cílem uživatelského testování bylo zjistit, jak se reální uživatelé chovají v prostředí aplikace a odhalit chyby a problémová místa při používání aplikace. Na základě testování byly chyby a nejasnosti v aplikaci opraveny.

7.2 Cílová skupina

Jak bylo řečeno na začátku práce, tak cílovou skupinou jsou osoby samostatně výdělečně činné působící na území České republiky, jež ze své praxe znají všechny náležitosti, které jsou v rámci aplikace řešené.

Z důvodu malého počtu osob samostatně výdělečně činných, kteří se byli schopni zúčastnit testování, se testování zúčastnili i jedinci, kteří nejsou osobami samostatně výdělečně činnými, ale jejich rutina je jim blízká.

7.3 Průběh testování

Vzhledem k epidemiologické situaci v čase vypracovávání práce testování proběhlo vzdáleně. Obrazové a zvukové spojení s uživateli bylo zajištěno přes komunikační aplikace Skype, Google Meet či FaceTime.

Na začátku sezení byli uživatelé seznámeni s účelem aplikace a samotného testování, při čemž jim byl zaslán PDF dokument s jednotlivými scénáři. Uživatelé postupně procházeli aplikací na základě daných scénářů. Při samotném průchodu aplikací bylo sledováno jejich počínání a v případě neočekávaných kroků bylo vše poznamenáno. Na konci samotného testování byla od uživatele zjištěna zpětná vazba a případné nápady na další rozšíření aplikace.

Na základě výsledků uživatelského testování byly v aplikaci opraveny chyby a vylepšeny některé části aplikace, jež byly pro uživatele matoucí.

7.4 Testovací scénáře

Testovací scénářů bylo celkem sedm, ale každý ze scénářů byl určen pro jiného uživatele.

První tři scénáře byly společné pro všechny testované uživatele. Scénáře se zabývaly kompletním průchodem aplikace, a to od samotné registrace, přes vytváření klientů, projektů, zaznamenávání odpracovaného času, až po vystavování faktur a testování upozornění. Průchod se odehrával ve webových prohlížečích na desktopových systémech.

Další tři scénáře se týkaly Electron aplikace a lišily se v závislosti na platformě, kterou má uživatel k dispozici. Buď probíhalo testování na Windows nebo na MacOS. V případě MacOS se scénář lišil v závislosti na tom, zda uživatel vlastní MacBook Pro s TouchBarem či nikoliv. V rámci těchto scénářů se testovala schopnost nainstalovat aplikaci, přihlásit se do ni a používat časovač buď pomocí ikony v rámci oznamovacího centra nebo pomocí TouchBaru.

Poslední scénář byl zaměřen na průchod systémy Android, iOS a iPadOS. V rámci něho se testovala schopnost nainstalovat aplikaci na plochu a použít ji v režimu bez připojení k internetu.

7.5 Výsledky testování

Testování bylo přínosné, neboť odhalilo řadu menších chyb, které se v aplikaci nacházely. Mimo jiné zpětná vazba uživatelů vedla k několika vylepšením samotné aplikace.

7.6 Porovnání s existujícími řešeními

Nejčastější aplikací, kterou dotazování OSVČ používají, je Fakturoid. Jedná se o webovou aplikaci, která je v oblasti evidence faktur velmi pokročilá. Oproti mé aplikaci však běží pouze v prohlížeči a nemá možnost evidovat projekty, odpracovaný čas, jednotlivé platby či na ně upozorňovat. Bezplatně je možné vystavovat faktury pouze pěti klientům.

Pro zaznamenávání odpracovaného času nejčastěji uživatelé používají zahraniční nástroj Toogl, jež má dostupnou aplikaci pro všechny platformy, umí zaznamenávat čas k jednotlivých klientům a projektům a generovat pokročilé reporty do PDF. Bohužel žádnou další funkcionalitu neposkytuje.

Poslední aplikací, kterou však nikdo z dotazovaných nepoužívá, je iDoklad. iDoklad je nástroj pro kompletní finanční správu, který má k dispozici i mobilní aplikace. I tak však neumí evidovat odpracovaný čas, platby pojistného či upozorňovat na platbu záloh. Navíc je zdarma pouze po dobu 60 dní.

Pro potřeby OSVČ tedy neexistuje žádná aplikace, která by spojovala všechnu funkcionalitu do jedné aplikace. V případě dalšího vývoje by mohla mít velký potenciál.

Kapitola 8

Závěr

8.1 Shrnutí

V rámci celé této práce jsme si představili tři technologie pro vývoj multiplatformních aplikací, a to progresivní webové aplikace, Electron a hybridní aplikace.

V jednotlivých kapitolách jsme si představili principy daných technologií a na praktických ukázkách ukázali, jak takovou aplikaci lze vyvíjet a distribuovat. Posléze jsme si detailněji nadefinovali požadavky, které by aplikace měla umět. Nad jednotlivými požadavky jsme pak zvažovali užití konkrétních technologií. Na základě nich jsme pro implementaci aplikace zvolili technologii progresivních webových aplikací a Electronu.

Dalším krokem byla implementace aplikace pro evidenci účetních a provozních dat a webového API. Programovací jazyky, technologie a knihovny použité při implementaci byly posléze detailně rozebrány v rámci této práce, stejně tak jako databázové schéma či jednotlivé obrazovky aplikace. Podrobněji byly pak rozebrány klíčové části aplikace, možnost publikace aplikace a její podpora.

V závěru práce bylo popsáno uživatelské testování aplikace a porovnání s již existujícími řešeními.

8.2 Přínos práce

Po přečtení práce by čtenář měl mít přehled o možnostech tvorby multiplatformních aplikací za pomoci webových technologií HTML, CSS a Javascriptu a měl by být schopen takovou aplikaci napsat.

Osobním přínosem této práce pro mě bylo detailní seznámení se s danými technologiemi a schopnost takovou aplikaci implementovat a publikovat.

8.3 Budoucí rozvoj aplikace

Implementovaná aplikace sloužila především pro demonstraci zvolených technologií. V průběhu implementace i testování se však ukázalo, že i základní verze aplikace může být pro osoby samostatně výdělečně činné přínosná.

Do budoucna by se tak mohlo přemýšlet kupříkladu o implementaci sekce pro evidenci nákladů, počítání zisku na základě faktur a nákladů či podrobnější evidence projektů s možností nahrávat přílohy a poznámky.

Z pohledu uživatelského zážitku by se pak mohlo jednat například o přidání podpory klávesových zkratek či vylepšení funkcionality v režimu offline.

Literatura

[1] E. contributors. (2019) The how and what of javascript desktop applications. [Online]. Available: https://medium.com/developers-writing/building-a-desktop-application-with-electron-204203eeb658

[2] R. Sharma. (2018) Abc of redux. [Online]. Available: https://dev.to/radiumsharma06/abc-of-redux-5461

[3] MDN. (2019) Progressive web apps. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

[4] ——. (2019) Progressive web app advantages. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Advantages

[5] ——. (2019) Web app manifest. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Manifest

[6] W3C. (2019) Web app manifest. [Online]. Available: https://www.w3.org/TR/appmanifest/

[7] MDN. (2019) Progressive enhancement. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement

[8] ——. (2019) Using service workers. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

[9] ——. (2019) Cache. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Cache

[10] ——. (2019) Using the notifications api. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API

[11] A. Inc. (2016) Configuring web applications. [Online]. Available: https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html

[12] P. LePage. (2019) Add to home screen. [Online]. Available: https://developers.google.com/web/fundamentals/app-install-banners

[13] E. contributors. (2019) About electron. [Online]. Available: https://electronjs.org/docs/tutorial/about

[14] M. Michálek. (2018) package.json: Vývojářský manifest pro každý projekt. [Online]. Available: https://www.vzhurudolu.cz/prirucka/package-json

[15] (2019) electron-builder. [Online]. Available: https://www.electron.build

[16] E. contributors. (2019) Api electron. [Online]. Available: https://electronjs.org/docs/api

[17] (2019) Webview. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView

[18] A. Inc. (2019) Wkwebview. [Online]. Available: https://developer.apple.com/documentation/webkit/wkwebview

[19] ——. (2019) App updates for html5 apps. [Online]. Available: https://developer.apple.com/news/?id=09062019b

[20] statcounter.com. (2019) Statcounter global stats. [Online]. Available: https://gs.statcounter.com/

[21] restfulapi.net. (2020) Rest architectural constraints. [Online]. Available: https://restfulapi.net/rest-architectural-constraints/

[22] sass lang.com. (2020) Sass basics. [Online]. Available: https://sass-lang.com/guide/

[23] nodejs.org. (2020) About node.js. [Online]. Available: https://nodejs.org/en/about/

[24] ——. (2011) What is npm. [Online]. Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

[25] webpack.js.org. (2020) Concepts, webpack. [Online]. Available: https://webpack.js.org/concepts/

[26] ——. (2020) What is babel? [Online]. Available: https://babeljs.io/docs/en/

[27] A. Stolyar. (2018) offline-plugin for webpack. [Online]. Available: https://github.com/NekR/offline-plugin

[28] E. B. contributors. (2020) offline-plugin for webpack. [Online]. Available: https://www.electron.build

[29] S. Buna. (2019) All the fundamental react.js concepts, jammed into this single medium article. [Online]. Available: https://medium.com/edge-coders/all-the-fundamental-react-js-concepts-jammed-into-this-single-medium-article-c83f9b53eac2

[30] R. T. contributors. (2020) Primary components. [Online]. Available: https://reactrouter.com/web/guides/primary-components

[31] D. Abramov and the Redux documentation authors. (2020) Getting started with redux. [Online]. Available: https://redux.js.org/introduction/getting-started

[32] Y. Tay. (2019) In-depth overview, flux. [Online]. Available: https://facebook.github.io/flux/docs/in-depth-overview

[33] Codesmith. (2018) Understanding javascript memoization in 3 minutes. [Online]. Available: https://codeburst.io/understanding-memoization-in-3-minutes-2e58daf33a19

[34] A. Garcia-Raboso. (2019) redux-api-middleware. [Online]. Available: https://github.com/agraboso/redux-api-middleware

[35] A. Inc. (2020) Introduction to json web tokens. [Online]. Available: https://jwt.io/introduction/

[36] D. Arias. (2018) Hashing in action: Understanding bcrypt. [Online]. Available: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/