Aplikace pro mobilní monitoring zpracování zakázek

Bc. Jakub Sýkora

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jakub Sýkora**
Osobní číslo:	**A16140**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**prezenční**
Téma práce:	**Aplikace pro mobilní monitoring zpracování zakázek**
Téma anglicky:	**An Application for the Mobile Monitoring of Order Processing**

Zásady pro vypracování:

1. **Prostudujte a popište možnosti tvorby hybridních mobilních aplikací pomocí frameworku Ionic.**
2. **Zaměřte se také na technologie, které Ionic framework využívá, jako je Angular či Apache Cordova.**
3. **Definujte funkční a nefunkční požadavky aplikace pro mobilní monitoring.**
4. **Za použití Ionic frameworku implementujte mobilní aplikaci dle požadavků.**
5. **Navrhněte také webové rozhraní zobrazující výstupy aplikace i v prostředí desktopového počítače.**
6. **Sestavte aplikaci pro platformu Android a otestujte na reálném zařízení.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **YUSUF, Sani. Ionic Framework By Example. Birmingham, Spojené království: Packt Publishing, 2016. ISBN 1782175415.**
2. **PHAN, Hoc. Ionic Framework Cookbook. Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1785287974.**
3. **KHANNA, Rahat. Ionic: Hybrid Mobile App Development. Birmingham, Spojené království: Packt Publishing, 2017. ISBN 1788297814.**
4. **JANSEN, Remo H. Learning TypeScript. Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1783985550.**
5. **DE WOLFF, Ivo Gabe. TypeScript Blueprints. Birmingham, Spojené království: Packt Publishing, 2016. ISBN 1785888773.**
6. **CASTRO, Elizabeth a Bruce HYSLOP. HTML5 a CSS3. Brno: Computer Press, Albatros Media, 2017. ISBN 8025144658.**

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **1. prosince 2017**

Termín odevzdání diplomové práce: **16. května 2018**

Ve Zlíně dne 11. prosince 2017

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.5.2018

.....
podpis diplomanta

ABSTRAKT

Cílem této práce je vytvořit mobilní aplikaci, která by umožňovala sledování a záznam procesu zpracování zakázek. Součástí práce je také webové rozhraní pro zobrazení výstupů aplikace. Teoretická část obsahuje popis technologií, které byly při vývoji použity. Praktická část se zabývá tvorbou aplikace a webové stránky s popisem uživatelského rozhraní.

Klíčová slova: mobilní aplikace, zpracování zakázek, Ionic, Angular, Apache Cordova

ABSTRACT

The target of this work is creation of a mobile application, which would provide means to monitor and register the process of order processing. It also includes a web interface for displaying this process. The theoretical part describes technologies used during the applications development. The practical part deals with creation of application and web page with description of its user interface.

Keywords: mobile application, order processing, Ionic, Angular, Apache Cordova

Chtěl bych poděkovat vedoucímu práce panu Ing. Radku Valovi, Ph.D. za odbornou pomoc. Dále bych chtěl poděkovat své rodině za jejich podporu při studiu.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době je již téměř nutností, aby každá větší společnost, která podniká v oblasti výroby či zpracování, měla informační systém. Část tohoto systému by měla sloužit jako prostředek pro sběr a interpretaci časových údajů při zpracovávání zakázek. Nejlepší varianta, jak implementovat tuto funkcionalitu je vytvoření mobilní aplikace. Hlavní funkcí této aplikace je poskytnout uživateli na pracovišti jednoduchý a rychlý prostředek k zaznamenávání časových informací o zakázkách, které momentálně zpracovává.

Cílem této práce je vytvořit aplikaci pro mobilní monitoring zpracování zakázek. V problematice vývoje mobilních aplikací je v dnešní době velmi populární hybridní přístup. Ten umožňuje velmi efektivní vývoj aplikace na více platforem a dramaticky tak redukuje potřebný čas a náklady. Součástí hybridního vývoje je také oblast webových hybridních mobilních aplikací. Takové aplikace jsou vytvářeny pomocí webových technologií s využitím instance webového prohlížeče (WebView). Velmi populární je v současnosti framework Ionic, který tento přístup využívá.

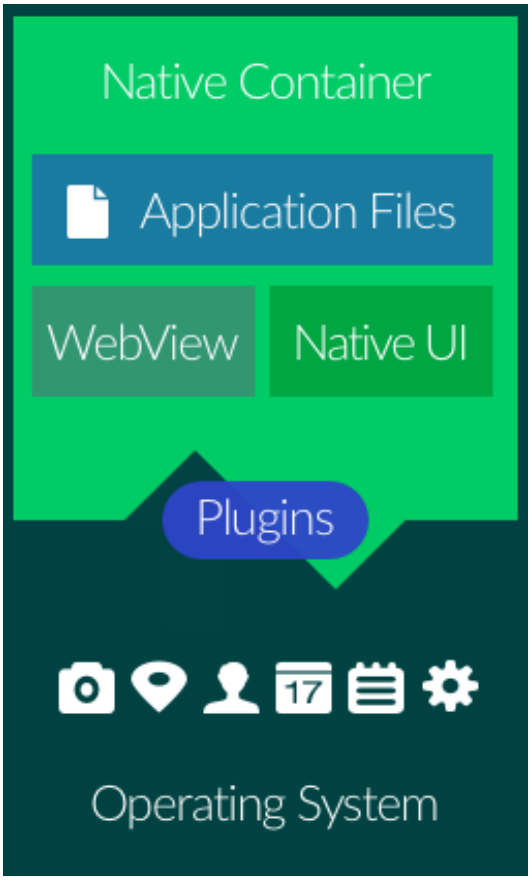
V rámci teoretické části jsou popsány technologie, které byly při vývoji aplikace použity. Jedná se zejména o framework Ionic, Apache Cordova a Angular. Dále je zde zmíněno několik dalších nástrojů z oblasti webových technologií, které jsou v práci využívány. V praktické části jsou nejdříve uvedeny požadavky na aplikaci a diagram případů užití. Dále je zde rozebrána implementace aplikace s popisem uživatelského rozhraní. Součástí práce je také návrh webového rozhraní pro zobrazení výstupů aplikace v prostředí desktopového počítače.

Práce je vytvářena v rámci projektu smluvního výzkumu Fakulty aplikované informatiky. Její výstupy budou integrovány do informačního systému, který fakulta vyvíjí.

I. TEORETICKÁ ČÁST

1 WEBOVÉ HYBRIDNÍ MOBILNÍ APLIKACE

Webové hybridní mobilní aplikace se vyvíjejí podobně jako aplikace webové. Oba přístupy používají webové technologie HTML, CSS a JavaScript. Na rozdíl od webových aplikací, které běží v klasických webových prohlížečích, webové hybridní aplikace používají pro svůj běh instanci webového prohlížeče (WebView). Toto WebView existuje téměř na všech platformách jako součást API pro vývoj mobilních aplikací. Webové hybridní aplikace tedy implementuje koncept, ve kterém běží WebView uvnitř nativní aplikace. To poskytuje hybridním aplikacím možnosti, které klasické webové aplikace postrádají. Nejdůležitější takovou možností je přístup k nativnímu API a hardwarovým funkcím zařízení pomocí pluginů. O tuto komunikaci s operačním systémem se ve většině dnešních hybridních aplikací stará framework Apache Cordova, který bude popsán později. [1]



Obrázek 1 – Schéma webových hybridních mobilních aplikací [2]

Hlavní výhodou hybridního přístupu je možnost multiplatformního vývoje. Při vývoji nativních aplikací na konkrétní platformu je třeba použít specifické nástroje, SDK a progra-

movací jazyk. Například Objective-C nebo Swift pro platformu iOS, Javu nebo Kotlin pro Android a C# pro Windows. Při použití webového hybridním přístupu lze využít znalostí z vývoje webů (HTML, CSS, JavaScript) k vytvoření aplikace, kterou lze sestavit pro všechny uvedené platformy. Tuto aplikaci lze poté distribuovat stejným způsobem jako aplikaci nativní, například prostřednictvím Google Play nebo App Store. Nevýhodou webových hybridních aplikací oproti nativním je jejich vyšší náročnost na výkon zařízení. Například při vývoji her s 3D grafikou nebo složitější práci s multimédii se nativní vývoj jeví jako lepší alternativa. [3]

2 APACHE CORDOVA

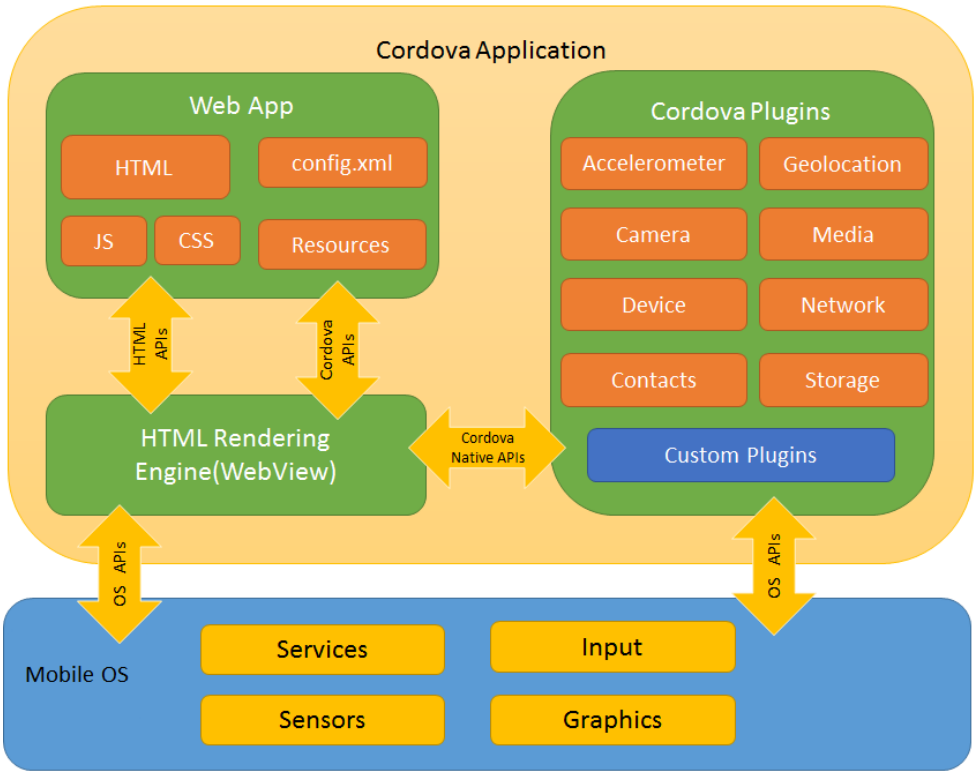
Apache Cordova je open-source framework pro vývoj mobilních aplikací. Byl původně vytvořen společností Nitobi, kterou v roce 2011 koupila firma Adobe Systems a začala framework provozovat pod názvem PhoneGap. Později jej se všemi zdrojovými kódy věnovala organizaci Apache Software Foundation, tím vznikl nový projekt Apache Callback, který byl vzápětí přejmenován na Apache Cordova. [4]

Framework umožňuje použití standardních webových technologií (HTML5, CSS3, JavaScript) k vývoji multiplatformních mobilních aplikací. Je to v podstatě rámec, který obaluje webovou aplikaci a zajišťuje komunikaci s operačním systémem. Dále zprostředkovává nativní API a zajišťuje překlad JavaScriptu do nativního jazyka zařízení.

Jsou zde dva základní směry, kterými lze postupovat při vývoji aplikace pomocí Apache Cordova frameworku. První se zaměřuje na vývoj multiplatformních aplikací pomocí Codova CLI, druhý je pak orientován na vývoj se zaměřením na specifickou platformu.

2.1 Architektura

Aplikace vytvořená pomocí frameworku Apache Cordova se skládá z několika komponent, jejichž uspořádání je zobrazeno na obrázku 2.



Obrázek 2 – Architektura Apache Cordova [5]

WebView je instance webového prohlížeče zařízení, která poskytuje aplikaci uživatelské rozhraní.

Web App je část, která obsahuje všechny zdrojové kódy. Samotná aplikace se implementuje jako webová stránka, jsou zde tedy soubory typu HTML, CSS a JavaScript spolu se všemi ostatními zdroji jako obrázky a média. Důležitým souborem této části je config.xml, který obsahuje informace o aplikaci a specifikuje její parametry.

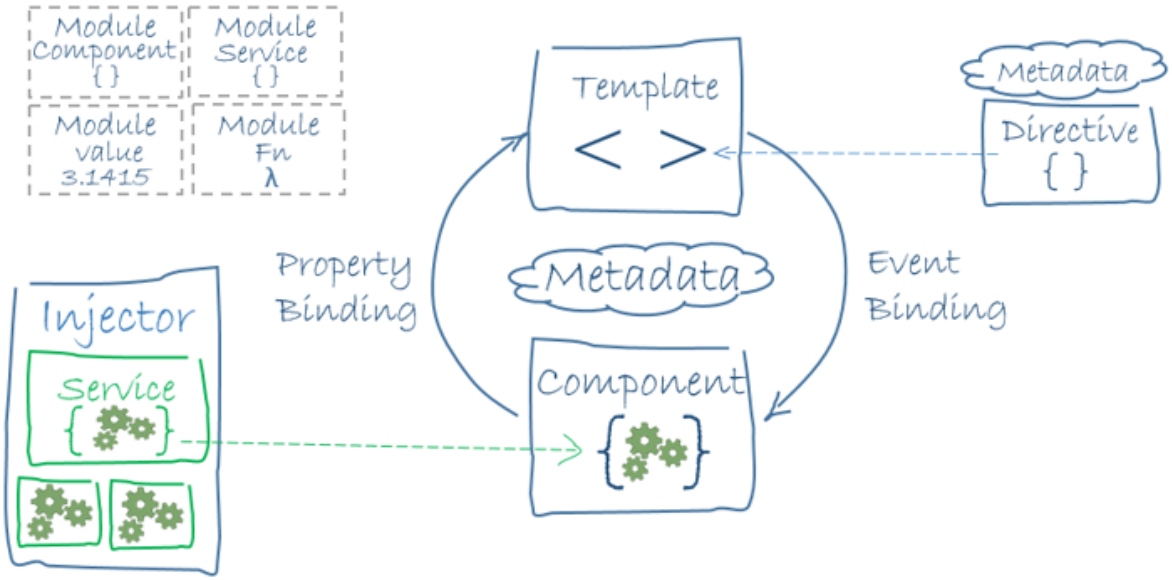
Pluginy jsou významnou částí aplikace. Poskytují rozhraní pro komunikaci mezi Cordova a nativními komponenty a umožňují volání nativního kódu z JavaScriptu. Cordova obsahuje balíček základních pluginů, které poskytují přístup k funkcím mobilního zařízení (kamera, kontakty, baterie). Je zde také možnost psaní vlastních pluginů nebo vyhledání a použití pluginů třetích stran, které mohou zpřístupňovat funkcionalitu více specifických nativních komponent, které nemusí být přístupné na všech platformách. [6]

3 ANGULAR

Angular je framework, který slouží pro vývoj klientských webových aplikací. O jeho vývoj a údržbu se stará skupina osob a korporací v čele se společností Google a jejím Angular týmem. Jeho vývoj byl poprvé oznámen v roce 2014 na ng-Europe konferenci pod názvem Angular 2.0. Předchozí verze Angular 1.x jsou nyní označovány jako AngularJS, verze 2.0 a vyšší jako Angular. Finální verze Angularu 2.0 tedy vyšla oficiálně v září roku 2016. V prosinci téhož roku byla oznámena další verze Angular 4.0, jejíž finální verze vyšla v květnu roku 2017 a je zpětně kompatibilní s verzí 2.0. V listopadu roku 2017 vyšla verze Angular 5.0. Vydání další verze je naplánováno na konec března roku 2018 pod názvem Angular 6.0 (nyní v testovací fázi). Další verze jsou naplánovány na přelom září/říjen 2018 (Angular 7.0) a březen/květen 2019 (Angular 8.0). Společnost Google prohlásila, že plánuje vydávat každý rok dvě nové verze Angularu s tím, že by měly být vždy zpětně kompatibilní s verzemi předchozími. [7]

Framework se používá pro vývoj aplikací v jazyce HTML a JavaScript, případně jazycích, které se kompilují do JavaScriptu – například TypeScript. Skládá se z několika knihoven (jádrových a volitelných).

3.1 Architektura



Obrázek 3 – Architektura frameworku Angular [8]

3.1.1 Moduly

Aplikace vyvíjené pomocí Angular frameworku se skládají z modulů, samotný framework má vlastní modulární systém, který se nazývá *NgModules*. Existují dva typy modulů – root a feature. Každá Angular aplikace musí obsahovat root modul, feature moduly jsou volitelné. Samotný modul je třída, která je označená dekorátorem *@NgModule*. Tento dekorátor pomocí souboru metadat definuje vlastnosti modulu. [9]

3.1.2 Komponenty

Základním stavebním prvkem Angular aplikace jsou komponenty. Tyto komponenty jsou sdružovány ve stromové hierarchii, kde kořenem je komponenta samotné aplikace. Komponenty jsou kompozitní, to znamená, že se komponenta může skládat z více dílčích komponent. Vzhledem ke stromové hierarchii se tedy při spuštění aplikace načte kořenová komponenta, která dále rekurzivně renderuje své dílčí komponenty (potomky). Každá komponenta se skládá ze tří částí – dekorátor, pohled a kontrolér. [10]

3.1.2.1 Dekorátor

Dekorátor komponenty se značí *@Component* a obsahuje dva hlavní prvky – selektor a šablonu. Selektor slouží jako identifikátor komponenty v HTML šabloně. V kontextu HTML se jedná v podstatě o nový tag se svou vlastní funkcionalitou.

3.1.2.2 Pohled

Pohled komponenty reprezentuje HTML šablona, kterou lze specifikovat dvěma způsoby. Za pomoci speciální syntaxe ji lze zapsat přímo do dekorátoru komponenty, nebo do dekorátoru vložit URL odkaz na separátní HTML soubor.

3.1.2.3 Kontrolér

Poslední částí komponenty je třída kontroléru, která se stará o její logiku. Tato třída je spojená s šablonou pohledu prostřednictvím rozhraní metod a vlastností. O sdílení a synchronizaci dat mezi šablonou a třídou kontroléru se stará tzv. data binding. Pomocí speciální syntaxe lze v šabloně specifikovat prvky a způsob, jakým budou spojeny s prvky třídy dané komponenty.

3.1.3 Služby

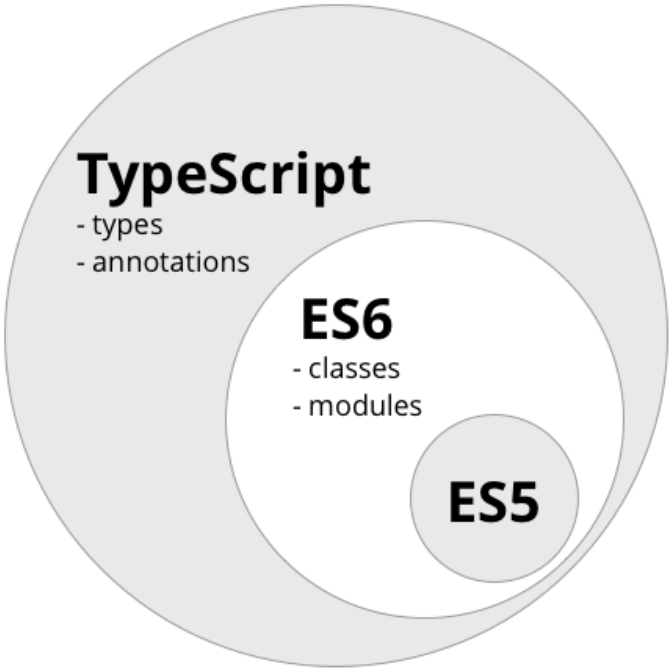
Důležitou částí Angular aplikace jsou služby. Komponenta by měla být navžena tak, aby umožňovala data binding skrze vlastnosti a metody. Další netriviální úkoly by měly být implementovány pomocí služeb. Služba je většinou třída, která zajišťuje přesně definovanou funkcionalitu. [11]

3.1.4 Dependency injection

Funkce služeb jsou komponentě zprostředkovány pomocí dependency injection. Nejjednoduší formou dependency injection je importování souboru s kódem, který chceme použít. Ve většině případů se však dependency injection specifikuje pomocí parametrů konstruktoru. Vždy, když Angular vytváří novou komponentu, podívá se na typy parametrů konstruktoru a na základě těchto typů poté požádá injector a příslušné služby. Injector udržuje kolekci instancí používaných služeb. Pokud požadovaná služba není v kolekci, injector ji přidá. V momentě, kdy se v kolekci nachází všechny požadované služby, Angular zavolá konstruktor komponenty s těmito službami jako argumenty. [12]

4 TYPESCRIPT

TypeScript je programovací jazyk vytvořený společností Microsoft v roce 2012. Nyní se o jeho údržbu stará společně s firmou Google. TypeScript je rozšířením jazyka ES6 (ECMAScript 6), který vychází z jazyka ES5 (ECMAScript 5), známého jako JavaScript. Celá hierarchie je znázorněna na obrázku níže. [13]



Obrázek 4 – Vztah jazyků Typescript, ES6 a ES5 [14]

Hlavní výhodou TypeScriptu je, jak už samotný název napovídá, jeho typovací systém. V JavaScriptu se dosud definovaly proměnné pouze pomocí klíčového slova *var*. V TypeScriptu můžeme dále specifikovat typ proměnné (boolean, number, string, ...). To umožňuje využívat výhody statického typování, kde se typy proměnných kontrolují při kompilaci, což u klasického JavaScriptu nelze (kompilátor nemá). Další prvek, kterým se TypeScript liší od JavaScriptu, je systém tříd. Zatímco JavaScript funguje na principu prototypů, TypeScript implementuje klasický systém tříd, jak jej známe z jiných objektově orientovaných jazyků. To samozřejmě zahrnuje i princip dědičnosti, kde můžeme specifikovat rodičovskou třídu pomocí klíčového slova *extends*. Třídy typicky obsahují metody, konstruktor a vlastnosti. Dále existuje spousta dalších funkcí a vlastností, které TypeScript nabízí – rozhraní, generika, import/export modulů, dekorátory a další. V dnešní době ještě webové prohlížeče nepodporují všechny funkce TypeScriptu, většinou tedy probíhá transpilace do JavaScriptu, který je plně podporován téměř všemi prohlížeči. [13]

4.1 Základní typy

Jak již bylo zmíněno, hlavním prvkem, kterým se TypeScript liší od JavaScriptu, je typovací systém. Výhodou TypeScriptu je možnost přiřazení typů proměnným, vlastnostem a funkcím. TypeScript obsahuje předdefinovanou sadu základních typů. Některé tyto typy byly převzaty z JavaScriptu a některé jsou specifické pro TypeScript. [15]

- Boolean – tomuto typu je přiřazena jedna ze dvou hodnot – true, false.
- Number – reprezentuje číslo, které může být zapsané v desítkové, šestnáctkové, osmičkové nebo dvojkové soustavě.
- String – obsahuje textový řetězec. Řetězce v TypeScriptu musí být zapsané v jednoduchých nebo dvojitých uvozovkách.
- Null a Undefined – v JavaScriptu představují speciální hodnoty, v TypeScriptu jsou to také typy, kde každý může obsahovat pouze jednu hodnotu (hodnota null pro typ null, hodnota undefined pro typ undefined). Hodnoty null a undefined mohou být přiřazeny proměnným s jakýmkoliv typem. Toto je však zdrojem častých chyb při běhu aplikace, proto TypeScript kompilátor podporuje možnost přísnější kontroly těchto případů, kde hodnoty null a undefined mohou být přiřazeny pouze proměnným typu null, undefined a any.
- Array – obsahuje pole hodnot. Typ pole specifikuje typ hodnot v poli.
- Tuple – heterogenní pole, které je deklarované jako pole typů konečné délky. Používá se v případě, kdy je potřeba z funkce vrátit více hodnot.
- Enum – obsahuje sadu hodnot. Každá hodnota v této sadě má své jméno a číselnou reprezentaci, která bez explicitního přiřazení začíná hodnotou 0.
- Any – pokud je proměnná deklarovaná s typem any, je možné do ní přiřadit hodnoty různých typů. Při kompilaci neprobíhá typová kontrola.
- Void – je většinou používán jako návratová hodnota funkcí, které nevracejí žádnou hodnotu. Může být také použit jako typ při deklaraci proměnných. V tom případě může proměnná nabývat hodnot null nebo undefined.
- Union – používá se pro reprezentaci hodnot, které mohou nabývat více typů. Při deklaraci jsou tyto typy oddělené znakem "|".[16]

4.2 Funkce

Ve funkcích TypeScriptu lze na rozdíl od funkcí v JavaScriptu přidat informaci o typu jejich argumentů a typu návratové hodnoty.

4.2.1 Argumenty funkcí

Způsob předávání argumentů funkci je jedna z hlavních oblastí, ve kterých se TypeScript od JavaScriptu liší. JavaScript implementuje velmi volný způsob zacházení s těmito argumenty. Funkce může být deklarována s libovolným počtem vstupních parametrů. V okamžiku volání ji může být předán libovolný počet parametrů. Pokud je předáno větší množství argumentů, než je uvedeno v deklaraci funkce, přebytečné argumenty se ignorují. V opačném případě, kdy je počet předávaných argumentů menší, než je uvedeno v deklaraci funkce, je zbytek argumentů nastaven na hodnotu undefined. TypeScript implementuje více striktní politiku v zacházení s argumenty. Počet předávaných argumentů se musí rovnat počtu argumentů ve formální deklaraci funkce. Nedodržení tohoto pravidla vyústí v chybu při kompilaci. [17]

4.3 Rozhraní

Stejně jako u klasických objektově orientovaných jazyků i TypeScript poskytuje vývojáři koncept rozhraní. To má dvě základní využití. První je typický příklad z jiných objektově orientovaných jazyků, kde třídy mohou implementovat rozhraní pomocí klíčového slova implements. V rámci druhého způsobu využití mohou rozhraní definovat strukturu objektů. [18]

4.4 Třídy

Koncept tříd je základním prvkem objektově orientovaných jazyků. TypeScript umožňuje použití abstraktních tříd, které nemohou být přímo instanciovány. Takové třídy obsahují abstraktní metody, které musí být nejdříve implementovány v odvozených třídách. Dále je zde podporován princip dědičnosti a modifikátory přístupu (public, private, protected). Třídy mohou obsahovat konstruktor, který se volá při tvorbě nových instancí. [19]

4.5 Moduly

Od roku 2015 (ES6) je v JavaScriptu využíván koncept modulů. Stejný koncept je implementován také v TypeScriptu. Každý modul pracuje v rámci svého prostoru. To znamená,

že proměnné, funkce a třídy deklarované v modulu nejsou viditelné v rámci jiných modulů, pokud nejsou explixitně exportovány použitím klíčového slova *export.*Aby mohl modul přistupovat k prostředkům jiného modulu, musí být importován pomocí klíčového slova *import*. [20]

Každý soubor, ve kterém je použit tento koncept importování/exportování, považujeme za modul. Ostatní soubory, které import/export neobsahují, se označují jako skripty. Obsah těchto skriptů je poté k dispozici v globálním měřítku.

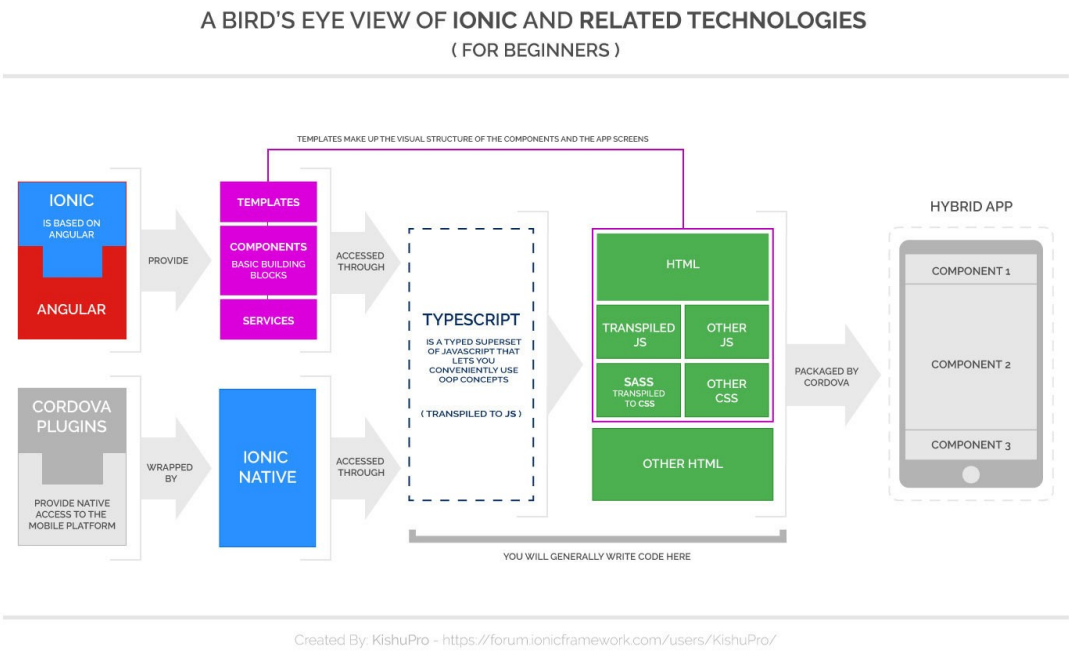
4.5.1 Module loader

Importování modulů obstarává tzv. module loader, který je zodpovědný za lokalizaci a přiřazení všech dependencies modulu před jeho spuštěním. Typickými příklady těchto module loaderů jsou například require.js pro webové aplikace nebo CommonJS pro Node.js. [20]

5 IONIC

Ionic je open-source SDK pro vývoj hybridních mobilních aplikací. Byl vytvořen v roce 2013 společností Drifty Co., která se specializuje na vývoj softwarových nástrojů pro tvorbu mobilních a webových aplikací. Původní verze byla postavená na frameworcích AngularJS a Apache Cordova. Nyní je Ionic dostupný ve verzi 3.9.2 s podporou Angularu 5.0.0. [21]

Ionic umožňuje vývoj multiplatformních mobilních aplikací pomocí webových technologií a jazyků (HTML, CSS, JavaScript, Angular, TypeScript). Obsahuje kolekce komponent, které simulují vzhled, chování a funkcionalitu nativních aplikací na platformách jako Android a iOS. Dále podporuje Universal Windows Platform pro vývoj aplikací na operační systém Windows 10.



Obrázek 5 – Schéma frameworku Ionic a dalších technologií [22]

5.1 Ionic Native

Jak již bylo zmíněno, Ionic úzce spolupracuje s frameworkem Apache Cordova. Ten poskytuje přístup k nativním funkcím zařízení pomocí JavaScriptu a umožňuje webové aplikaci běžet jako nativní na mobilních platformách. V Ionic aplikaci lze použít přes 200 Cordova pluginů, které jsou používány k přístupu k nativnímu rozhraní daných platforem a zpřístupňují jejich funkcionalitu (geolokace, kamera, gyroskop, ...). Tyto Cordova pluginy

jsou do Ionic aplikace integrovány prostřednictvím knihovny Ionic Native. Ionic Native je ES6/TypeScript implementací dnešních ES5 Cordova pluginů, umožňuje jejich import a použití v TypeScriptu. [23]

Pro vývoj aplikací ve frameworku Ionic je potřeba mít nainstalovanou sadu nástrojů.

5.2 Ionic CLI

Ionic CLI (Command-Line Interface) je hlavním nástrojem, který se při vývoji Ionic aplikací používá. Je plně zpětně kompatibilní s projekty, které byly vytvořeny v předchozích verzích Ionicu. Umožňuje nám vytvářet nové projekty s možností použití předdefinovaných šablon. Pomocí Ionic CLI dále spouštíme, emulujeme a sestavujeme vytvořené projekty a generujeme resources. Poskytuje nám možnost přidávat platformy, generovat stránky a providery. Tyto funkce společně s mnoha dalšími také šetří čas a ulehčují vývojáři tvorbu aplikace generováním kódu, který by jinak musel být psán manuálně. [24]

5.3 Node.js

Node.js je asynchronní běhové prostředí pro JavaScript, které je řízené událostmi. Bylo vytvořeno v roce 2009 Ryanem Dahlem a na jeho vývoji se podílí firma Joyent. Základem Node.js je interpret V8 od firmy Google, který vyniká svou rychlostí a je snadno zabudovatelný do jiných aplikací. Samotný Node.js je tedy postavený na enginu V8, který rozšiřuje o funkcionalitu, která poskytuje prováděným scriptům přístup k síťovým funkcím a souborům. To umožňuje použití Node.js pro běh JavaScriptu jak na desktopových počítačích, tak i na serverech. Existují dva typy vydávaných verzí Node.js – stabilní verze LTS a stávající verze s nejnovějšími prvky a funkcemi. Je doporučováno používat nejnovější stabilní verze. [25]

Node.js se v kontextu frameworku Ionic používá jako běhové prostředí pro Ionic CLI, které je napsané v JavaScriptu.

5.3.1 npm

npm je největší softwarový registr na světě, jehož první verze byla vytvořena roku 2010. Každý týden zaznamenává přibližně 3 miliardy stažení. Obsahuje více než 600 000 JavaScriptových balíčků (stavební bloky kódu). Skládá se ze 3 hlavních částí:

- webová stránka
- rozhraní příkazové řádky

- registr

Webová stránka slouží k hledání balíčků a správě profilů. Lze zde nastavovat parametry sdílených balíčků a umožnuje spravovat jejich přístup (veřejné, soukromé). Rozhraní příkazové řádky je hlavním nástrojem pro interakci s npm a přístupem do registru. Samotný registr je rozsáhlá databáze softwaru psaného v JavaScriptu spolu s přiloženými metadaty. [26]

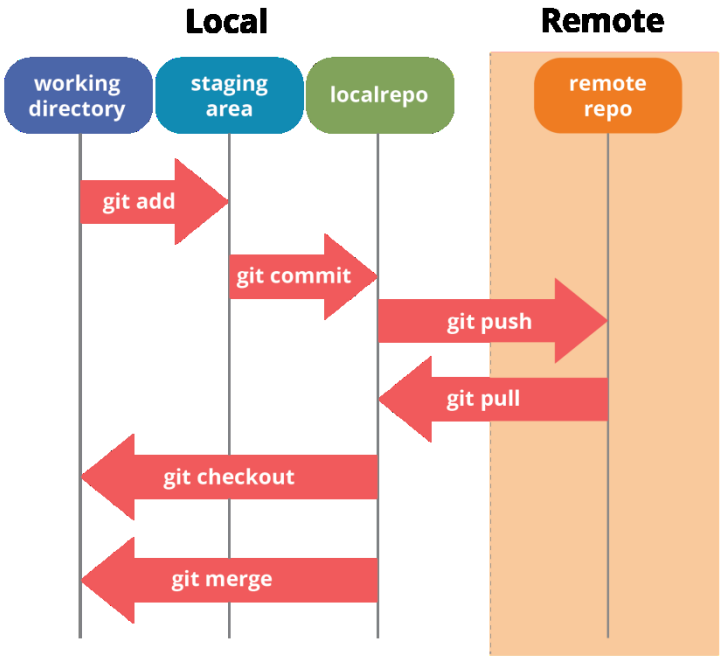
npm je výchozí manažer balíčků (modulů) pro prostředí Node.js.

5.4 Git

Git je open source systém správy verzí. Byl vytvořen roku 2005 Linusem Torvaldsem jako nástroj pro vývoj Linux kernelu. Je velmi rychlý, škálovatelný a umožňuje distribuovaný vývoj. Jeho hlavní předností je model vývojových větví, které mohou být navzájem nezávislé, lze s nimi pracovat separátně a později je slučovat. Git je systém distribuovaný, to znamená, že každý vývojář má k dispozici kopii historie celého vývoje. Git má svůj systém souborů a obsahuje dvě základní datové struktury: index a objektovou databázi. Index je proměnlivý a slouží k ukládání informací pracovního adresáře. Objektová databáze slouží k ukládání samotných dat a obsahuje 4 typy objektů:

- blob (binary large object) – je obsah souboru, nemá žádné jiné metadata, jeho interní název je hash jeho obsahu.
- tree – je ekvivalent adresáře, obsahuje názvy souborů a reference na jejich obsah.
- commit – spojuje tree objekty do historie. Ta obsahuje informace o názvu objektu tree (nejvyšší v hierarchii adresářů), logovací zprávy a jména rodičovských commitů.
- tag – je kontejner, který obsahuje referenci na jiný objekt společně s jeho metadaty.

Ionic používá Git při tvorbě nových projektů, konkrétně k získání zdrojových kódů předdefinovaných šablon projektů. Obrázek níže ilustruje běžné operace, které jsou používány při práci s Gitem. Také je zde vidět jdnoduchá struktura propojení mezi jednotlivými fázemi a repozitáři. [27]



Obrázek 6 – Schéma Git operací [28]

5.5 Struktura Ionic projektu

Pro vytvoření nového Ionic projektu se používá Ionic CLI, které po vykonání příkazu vygeneruje soubory (šablona získaná z Gitu) v předem definované struktuře. Jak již bylo zmíněno dříve, Ionic je postavený na frameworku Angular, tudíž i struktura Ionic projektu se velmi podobá struktuře webových aplikací vytvořených v Angularu. Tato struktura je popsána níže, některým klíčovým položkám bude věnováno více prostoru později.

- hooks
- node_modules
- platforms
- plugins
- resources
- src
- www
- config.xml
- ionic.config.json
- package.json
- tsconfig.json

– tslint.json

Celková struktura se může lišit v závislosti na konkrétní implementaci projektu. Například větší projekty v pokročilém stádiu vývoje mohou mít tuto strukturu více rozšířenou. Pro demonstraci a vysvětlení významu jednotlivých položek se však budu odkazovat na výše uvedené schéma. [23, 24]

- plugins – obsahuje všechny pluginy, které byly nainstalovány.
- node_modules – tato složka se automaticky vygeneruje při instalaci dependencies pomocí npm (uvedené v souboru package.json).
- platforms – složka, která obsahuje nativní projekty daných platforem (generují se pomocí příkazu v Ionic CLI).
- resources – obsahuje soubory, které aplikace pro běh vyžaduje, po vytvoření projektu jsou zde verze ikon a splashscreenů.
- src – zde probíhá velká část procesu vývoje. Je zde uložený zdrojový kód aplikace.
- hooks – tato složka obsahuje skripty, které jsou spuštěny při určitých událostech (po přidání platformy, před začátkem emulace, před spuštění aplikace, po přidání pluginu, ...)
- www – tato složka obsahuje sestavený Ionic kód napsaný ve složce src. Všechen kód z této složky je použit ve WebView aplikace.
- config.xml – tento soubor se skládá ze všech meta informací, které jsou potřeba pro konverzi aplikace do instalačních balíčků konkrétních platforem. Existují dva typy konfigurací, které zde můžeme specifikovat. První je globální, která platí pro všechny platformy a zařízení. Druhá je specifická pro konkrétní platformy.
- ionic.config.json – tento soubor obsahuje konfigurační informace samotného Ionicu, například název aplikace a její app_id.
- package.json – soubor, ve kterém jsou uvedeny všechny dependencies projektu. Také obsahuje metadata projektu jako název, popis, verze, licence a další.
- tsconfig.json – soubor, který obsahuje nastavení, parametry a konfiguraci TypeScriptu. Specifikuje způsob, jakým bude kód TypeScriptu kompilován do JavaScriptového kódu
- tslint.json – obsahuje seznam všech TypeScriptových lint pravidel a preferencí.

V následující části bude více rozebrána složka src a její struktura. Nachází se v ní zdrojový kód aplikace a při vývoji zde probíhá nejvíce práce.

- src
  - app
  - assets
  - pages
  - theme
  - index.html
  - manifest.json
  - service-worker.js

Stejně jako v přechozím případě, může být i tato struktura rozšířena o další položky v závislosti na typu a potřebách vytvářené aplikace. Typickým příkladem jsou složky services nebo providers a pipes, které mohou být vygenerovány příslušným příkazem v Ionic CLI. [23, 24]

- app – tato složka obsahuje veškeré globální konfigurace. Je zde soubor se styly, ve kterém se definují pravidla, která nejsou specifická pro konkrétní komponenty, ale aplikují se globálně. Také zde probíhá registrace nově vytvořených komponent a providerů, je zde správa modulů a import pluginů, které aplikace používá.
- assets – složka, která obsahuje všechny statické zdroje (typicky obrázky nebo i používané json data).
- pages – obsahuje všechny stránky aplikace. Každá stránka je komponenta a jak již bylo dříve zmíněno, Ionic je postavený na Angularu, to znamená, že každá stránka vytvořená v Ionicu musí obsahovat základní prvky a charakteristiky Angular komponenty. Složka pages tedy obsahuje složky stránek, kde každá obsahuje soubory typu .html, .ts a .scss. TypeScript soubor obsahuje dekorátor se selektorem a URL adresu šablony, která se nachází v HTML souboru. Dále také obsahuje třídu kontroléru, která se stará o logiku a data binding. Soubor se styly definuje pravidla pro vizuální stránku HTML šablony.
- theme – obsahuje soubor variables.scss, ve kterém jsou definovány Ionic Sass proměnné. Pomocí těchto proměnných lze rychle provádět změny stylů napříč celou aplikací. Tyto proměnné lze také definovat zvlášť pro každou platformu.
- index.html – tento soubor je hlavním vstupním bodem celé aplikace. Jeho úkolem je připravit styly a skripty a zajistit spuštění aplikace.
- manifest.json – soubor s metadaty.

- service-worker.js – konfigurační soubor.

5.5.1 Soubor app.module.ts

Tento soubor deklaruje základní kořenový modul aplikace. Obsahuje pouze jednu třídu AppModule, která je prázdná. Vetšina kódu je obsažena v dekorátoru *@NgModule*, který tuto třídu rozšiřuje. Tento dekorátor obsahuje několik vlastností, které budou blíže popsány v následujících kapitolách. V těchto kapitolách se vždy při popisu odkazuji na zdrojový kód níže. [24, 29]

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { ErrorHandler, NgModule } from '@angular/core';
3. import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-
 angular';
4. import { SplashScreen } from '@ionic-native/splash-screen';
5. import { StatusBar } from '@ionic-native/status-bar';
6. import { MyApp } from './app.component';
7. import { HomePage } from '../pages/home/home';
8. import { MainPage } from '../pages/main/main';
9. import { IonicStorageModule } from '@ionic/storage';
10.
11. @NgModule({
12.   declarations: [
13.     MyApp,
14.     HomePage,
15.     MainPage
16.   ],
17.   imports: [
18.     BrowserModule,
19.     IonicModule.forRoot(MyApp),
20.     IonicStorageModule.forRoot()
21.   ],
22.   bootstrap: [IonicApp],
23.   entryComponents: [
24.     MyApp,
25.     HomePage,
26.     MainPage
27.   ],
28.   providers: [
29.     StatusBar,
```

```
30.     SplashScreen,
31.     {provide: ErrorHandler, useClass: IonicErrorHandler}
32.   ]
33. })
34. export class AppModule {}
```

Zdrojový kód 1 – Ukázka obsahu souboru app.module.ts

5.5.1.1 Declarations

V tomto poli je potřeba zahrnout všechny komponenty a direktivy, které se v rámci aplikace budou používat. V opačném případě nebude Angular schopný tyto prvky rozeznat.

5.5.1.2 Imports

V této sekci je třeba uvést všechny moduly, jejichž funkcionalita bude v aplikaci využívána. V uvedeném příkladu je importován modul IonicModule, který dále nastavuje kořenovou komponentu MyApp. Tento modul obsahuje všechny komponenty a direktivy Ionic frameworku, které se jeho importováním zpřístupní a mohou být použity v rámci celé aplikace.

5.5.1.3 Bootstrap

Tato sekce specifikuje komponenty, které budou při spuštění aplikace vytvořeny. Obvykle je zde uvedena pouze jedna komponenta. V rámci uvedeného příkladu se jedná o komponentu IonicApp. Tato komponenta je kořenovou komponentou Ionicu a je zodpovědná za načtení komponenty MyApp.

5.5.1.4 EntryComponents

V této sekci musí být uvedeny všechny komponenty, které se načítají imperativně. To typicky zahrnuje komponenty stránek, o jejichž načítání se stará navigační kontrolér.

5.5.1.5 Providers

V sekci Providers se uvádějí všechny služby, jejichž funkcionalita se komponentám zpřístupňuje pomocí dependency injection. Tyto služby poté mohou využívat vschny komponenty v aplikaci.

5.5.2 Soubor app.component.ts

Tento soubor obsahuje komponentu, která je při spuštění aplikace načtena jako první. V dekorátoru je uveden soubor, který specifikuje její HTML šablonu. Dále je zde přiřazena hodnota proměnné rootPage, která se používá jako parametr navigační komponenty v HTML šabloně. To způsobí, že stránka, která je přiřazena proměnné rootPage, bude zobrazena po spuštění aplikace. [29]

V konstruktoru této komponenty jsou uvedeny operace, které se provádí při inicializaci aplikace. V uvedeném příkladu je to volání některých Ionic pluginů.

```
1. import { Component } from '@angular/core';
2. import { Platform } from 'ionic-angular';
3. import { StatusBar } from '@ionic-native/status-bar';
4. import { SplashScreen } from '@ionic-native/splash-screen';
5.
6. import { HomePage } from '../pages/home/home';
7. @Component({
8.   templateUrl: 'app.html'
9. })
10. export class MyApp {
11.   rootPage:any = HomePage;
12.
13.   constructor(platform: Platform, statusBar: StatusBar,
 splashScreen: SplashScreen) {
14.    platform.ready().then(() => {
15.       statusBar.styleDefault();
16.       splashScreen.hide();
17.     });
18.   }
19. }
```

Zdrojový kód 2 – Ukázka obsahu souboru app.component.ts

5.6 Komponenty

Komponenty jsou základním stavebním prvkem každé Ionic aplikace. Slouží k vytvoření uživatelského rozhraní. Vzhled a funkcionalita každé komponenty se může lišit v závislosti na platformě, na které aplikace poběží. Obecně by komponenty měly simulovat nativní uživatelské rozhraní dané platformy, v ideálním případě by mělo toto rozhraní být uživatelem nerozeznatelné od rozhraní nativní aplikace. Všechny komponenty jsou zdokumento-

vané a jejich použití s uvedením praktického příkladu je k nalezení na stránkách Ionic frameworku. Ionic v současné době obsahuje 28 komponent, které jsou rozděleny do dvou základních kategorií. [23, 30]

5.6.1 Šablonové komponenty

Jsou to komponenty, které se skládají pouze z HTML a CSS kódu. Šablonových komponent je celkově 20. Níže je uvedeno několik příkladů nejpoužívanějších šablonových komponent.

- Card
- Grid
- Button
- Icon

5.6.2 Dynamické komponenty

Jsou to komponenty, které obsahují interaktivní, dynamickou funkcionalitu. Dynamických komponent je 8. Typickým příkladem dynamické komponenty může být například Alert.

5.7 Navigace

Jako navigace se v Ionicu nazývá proces přepínání stránek. Není zde zapotřebí URL jako v klasickém webovém konceptu, který implementovaly první verze Ionicu. Nyní jsou stránky vkládány a odebírány do navigačního zásobníku. Tento přístup přesněji simuluje navigaci v nativních aplikacích. Základní prvek, který implementuje tuto funkcionalitu, je třída NavController. [24]

5.7.1 Třída NavController

Je to třída pro navigační kontrolér komponenty jako Nav a Tab. Tyto komponenty jsou používány k navigaci v aplikaci a lze si je představit jako zásobník, do kterého lze přidávat nebo z něj odebírat stránky. Tento zásobník reprezentuje historii navštívených stránek, kde se na jeho vrcholu nachází stránka, která je právě zobrazená. Nová stránka se vykreslí jejím vložením na vrchol zásobníku. Odebrání současné stránky ze zásobníku znamená navigaci na stránku předešlou. Navigační komponenta Nav bývá v aplikacích definována pomocí tagu <*ion-nav*> s parametrem root, ve kterém se nachází první vykreslená stránka (kořenová stránka). Abychom tuto instanci navigační komponenty mohli použít i

v ostatních stránkách (komponentách), je třeba ji získat pomocí injekce NavControlleru. Po získání této instance je možné pracovat s navigačním zásobníkem. Pro navigaci na novou stránku je třeba zavolat metodu push(), navigace na stránku předchozí se provádí zavoláním metody *pop()*. Při navigaci na novou stránku je možné zasílat data ve fromě parametrů. Tyto parametry se poté získají v navigované stránce pomocí injekce objektu NavParams. Proces navigace na další stránku je znázorněn ve zdrojovém kódu níže. [31]

```
1. import { Component } from '@angular/core';
2. import { NavController } from 'ionic-angular';
3. import { MainPage } from '../main/main';
4.
5. @Component({
6.   selector: 'page-home',
7.   templateUrl: 'home.html'
8. })
9. export class HomePage {
10.
11.   constructor(public navCtrl: NavController) {
12.   }
13.
14.   pushPage(){
15.     this.navCtrl.push(MainPage, {
16.       firstname: "Adam",
17.       surname: "Veselý"
18.     });
19.   }
20. }
```

Zdrojový kód 3 – Ukázka použití třídy NavController

Ve zdrojovém kódu je vidět obsah TypeScript souboru stránky Home. Navigaci zde obstarává metoda *pushPage*(), která může být spuštěna například v rámci obslužné události stisknutí tlačítka. Po zavolání této metody se nad instancí navigační komponenty, která je importována a poté injektována v konstruktoru, zavolá metoda *push* s parametrem stránky, kterou chceme vykreslit, a případnými parametry.

5.8 Ukládání dat

Při vývoji mobilních aplikací se často vyskytne potřeba ukládat data do zařízení. K tomu účelu poskytuje Ionic službu Storage, která je při vytvoření projektu součástí projektové

šablony a lze ji tedy přímo používat. Data se ukládají do páru klíč/hodnota, kde hodnota každého páru může být reprezentována daty jakéhokoliv typu. Pokud je hodnota JavaScriptový objekt, je před uložením serializován do JSON stringu. Při zpětném získávání hodnoty z uložiště je JSON string deserializován zpět do JavaScriptového objektu. Ionic Storage poskytuje rozhraní pro přístup k různým systémům, které se k ukládání dat používají (SQLite, IndexedDB, WebSQL, localstorage). Použitý systém závisí na jeho dostupnosti na konkrétní platformě. V současné době se jeví jako nejlepší volba systém SQLite, který je podporován na platformách Android a iOS. Je dostupný ve formě pluginu, který lze do projektu nainstalovat. Po instalaci je nutné jej přidat do seznamu importů aplikace. [23]

5.8.1 Třída Storage

Třída Storage poskytuje základní funkcionalitu pro komunikaci se systémem pro ukládání dat. Po přidání do pole importů aplikace je možné instanci této třídy injektovat do ostatních komponent. Níže jsem popsal základní metody, které lze volat nad instancí třídy Storage. [24]

- get(klíč) – na základě klíče najde v uložišti příslušnou hodnotu a vrátí ji.
- set(klíč, hodnota) – uloží hodnotu do páru s příslušným klíčem.
- remove(klíč) – odstraní pár klíč/hodnota podle zadaného klíče.
- clear() – smaže všechny položky uložiště.
- keys() – vrátí všechny klíče z uložiště.
- length() – vrátí počet klíčů v uložišti.
- forEach(callback) – zavolá callback funkci pro každý pár klíč/hodnota v uložišti.

V příkladu níže je ukázka kódu, který demonstruje přidání dat do uložiště a jejich zpětné získání.

```
1. import { Component } from '@angular/core';
2. import { NavController } from 'ionic-angular';
3. import { Storage } from '@ionic/storage';
4.
5. @Component({
6.   selector: 'page-main',
7.   templateUrl: 'main.html'
8. })
9. export class MainPage {
10.
```

```
11.  constructor(private storage: Storage, public navCtrl: NavCon-
 troller) {
12.   }
13.
14.   storageSet(){
15.     this.storage.set('jméno', 'Adam');
16.   }
17.
18.   storageGet(){
19.     this.storage.get('jméno').then((val) => {
20.       console.log('Vaše jméno je: ', val);
21.     });
22.   }
23. }
```

Zdrojový kód 4 – Ukázka použití třídy Storage

Na řádku 3 je vidět import třídy Storage do komponenty. Dále je v konstruktoru vidět injekce instance této třídy. Pro demonstraci přidání a získání dat z uložiště slouží metody *storageSet()* a *storageGet()*. V metodě *storageSet()* se nad instancí třídy Storage volá metoda *set* s parametry klíč a hodnota. Po zavolání této metody je tedy v lokálním uložišti zařízení pár definovaný poskytnutými parametry. V metodě *storageGet()* se nad instancí třídy Storage volá metoda *get* s parametrem klíč. Hodnota příslušného klíče je poté uložena do proměnné *val* a zalogována do konzole. [24, 32]

II. PRAKTICKÁ ČÁST

6 POŽADAVKY NA APLIKACI

V následující kapitole budou specifikovány požadavky na aplikaci pro mobilní monitoring zpracování zakázek. Popsána bude také funkcionalita této aplikace pomocí diagramu případů užití.

6.1 Funkční požadavky

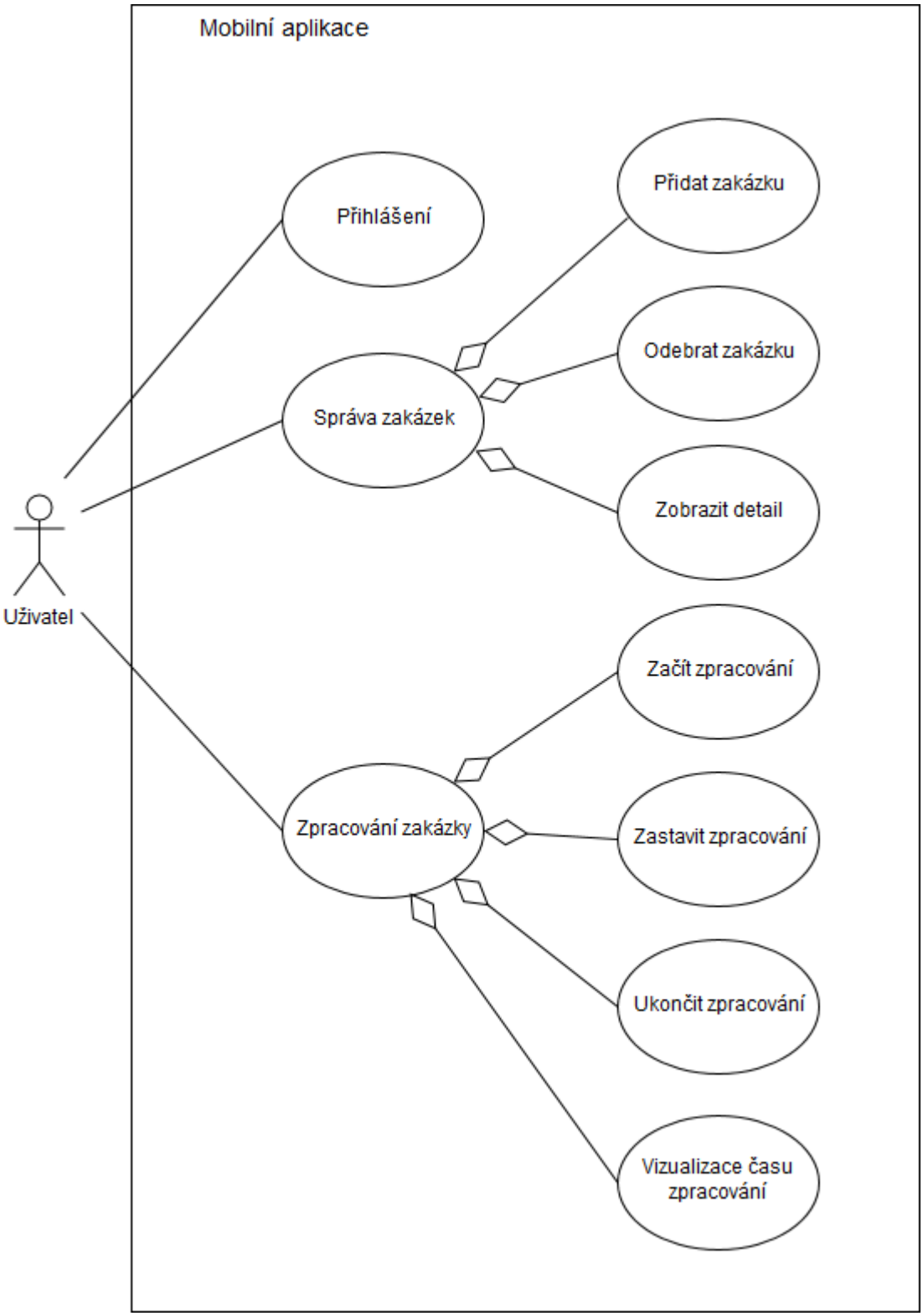
- aplikace bude umožňovat identifikaci a přihlášení uživatele na základě jeho unikátního čísla.
- aplikace bude zobrazovat uživateli seznam jeho zpracovávaných zakázek.
- aplikace poskytne uživateli možnost přidání zpracovávané zakázky prostřednictvím skenování QR kódu (každá zakázka bude mít svůj unikátní QR kód, který bude obsahovat základní informace potřebné k jejímu zpracování).
- aplikace poskytne uživateli možnost odebrání zakázky z jeho seznamu právě zpracovávaných zakázek.
- aplikace umožní zobrazení detailu zakázky s dodatečnými informacemi.
- aplikace umožní zobrazení časových údajů zpracování zakázky.
- aplikace bude umožňovat spuštění zpracování zakázky.
- aplikace bude umožňovat pozastavení zpracování zakázky.
- aplikace bude umožňovat obnovení zpracování zakázky.
- aplikace bude umožňovat ukončení zpracování zakázky.
- aplikace poskytne uživateli možnost odstranit časové údaje zpracování zakázky.
- aplikace umožní uživateli zobrazení grafické interpretace časových údajů zpracovávaných zakázek.
- aplikace bude zprostředkovávat informace o zpracování zakázek informačnímu systému prostřednictvím webových služeb.

6.2 Nefunkční požadavky

- aplikace bude multiplatformní (sestavitelná na více platforem)
- aplikace bude používat rozhraní webových služeb pro komunikaci s informačním systémem

6.3 Diagram případů užití

V této kapitole je funkcionalita aplikace popsána diagramem případů užití, který je znázorněn na obrázku níže.



Obrázek 7 – Diagram případů užití

Aplikace bude po přihlášení uživatele umožnovat správu zpracovávaných zakázek. To zahrnuje přidání, odebrání a zobrazení detailu s dalšími parametry zakázky. U každé zakázky bude v rámci jejího zpracování možné provádět několik operací. Uživatel bude mít možnost zahájit a zastavit zpracování zakázky. Tyto operace bude možno provádět opakovaně až do ukončení zpracování této zakázky. Aplikace bude také umožňovat uživateli zobrazit časy zpracování zakázky ve formě grafu.

7 WEBOVÁ STRÁNKA

Webové rozhraní slouží pro zobrazení výstupů mobilní aplikace v prostředí desktopového počítače. Stejně jako mobilní aplikace bude i tento web komunikovat s informačním systémem pomocí webových služeb. Webové rozhraní je postaveno na frameworku Angular. Aby bylo možné tento framework při vývoji použít, je třeba mít nainstalovanou sadu nástrojů.

- Node.js – pro instalaci je třeba navštívit sekci Downloads na oficiálních sránkách. Node.js je dostupný ve dvou verzích – LTS (stabilní verze) a Current (nejnovější verze). Je doporučeno stahovat stávající LTS verzi. Po stažení je třeba spustit instalační balíček a řídit se zobrazenými instrukcemi.
- npm – jako výchozí manažer balíčků se nainstaluje společně s Node.js.
- Angular CLI – je nástroj pro vytváření, vývoj a údržbu Angular aplikací. Instaluje se pomocí npm manažeru prostřednictvím příkazu *npm install -g @angular/cli*.

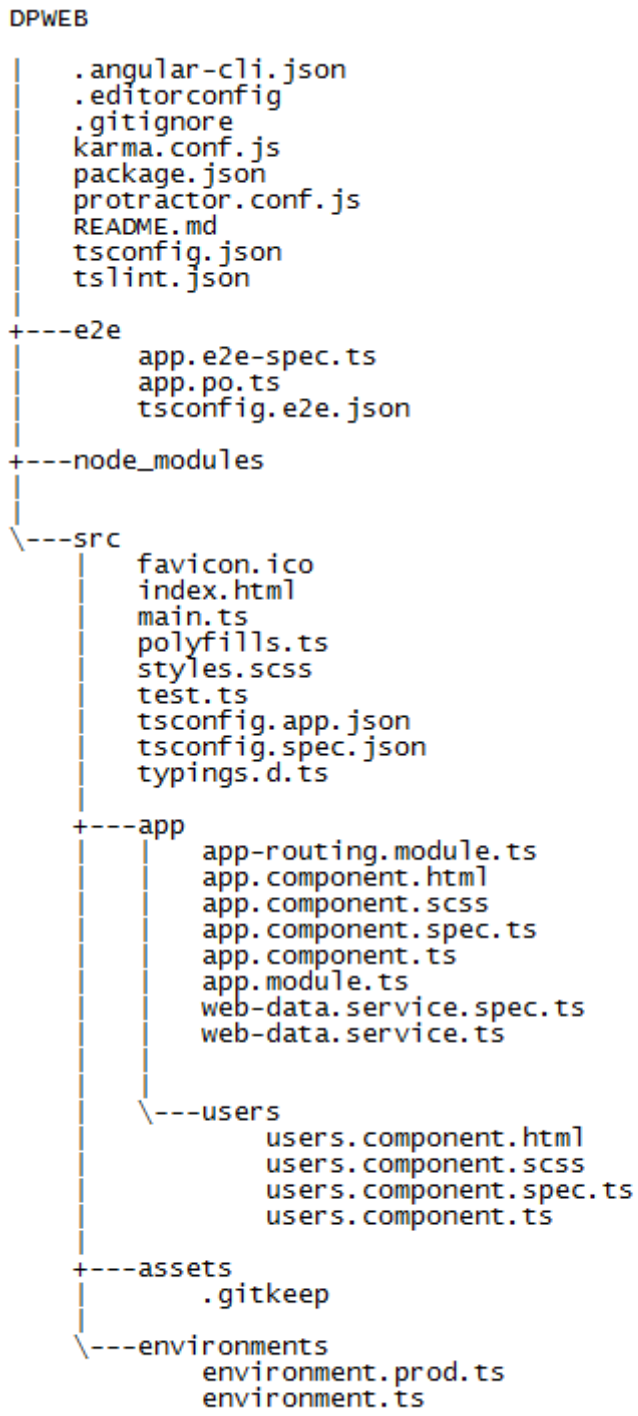
Po nainstalování Angular CLI je příkazem *ng new dpWeb* vytvořen nový projekt. Tím se také vytvoří základní adresářová struktura a vygenerují konfigurační soubory.

Angular CLI také poskytuje možnost generovat komponenty, což lze v případě mojí aplikace použít pro generování stránky. Příkazem *ng generate component users* se tedy v adresáři src vytvoří podadresář users, ve kterém se vygenerují soubory komponenty. Na základě struktury Angular komponenty to jsou user.component.ts, user.component.scss, user.component.html, user.component.spec.ts.

Dalším prvkem, který lze pomocí Angular CLI generovat, jsou služby. Ve webové aplikaci je třeba získávat data pomocí webových služeb. Tato funkcionalita je zapouzdřená ve službě webData, která se vygeneruje pomocí příkazu *ng generate service webData*. Tento příkaz vytvoří v adresáři app soubory web-data.service.ts a web-data.service.spec.ts, ve kterých je definována logika této služby.

7.1 Struktura projektu

Celková adresářová struktura vygenerovaného projektu s komponenty a službami je uvedena na obrázku níže.



Obrázek 8 – Adresářová struktura projektu webové stránky

Význam většiny těchto souborů a složek byl rozebrán v kapitole teoretické části 5.5 Struktura Ionic projektu. Ionic projekt je postaven na Angular frameworku, tudíž i jeho adresářová struktura bude velmi podobná projektu vytvořeném v Angularu. Z toho důvodu již nebude dále popisován význam konfiguračních souborů.

7.1.1 Stránka Users

Po přidání komponenty pomocí Angular CLI se v adresářové struktuře vytvořila složka users, ve které se vygenerovaly soubory pro tuto komponentu. Stránka Users se stejně jako většina Angular komponent skládá ze souboru se šablonou (users.component.html), třídou kontroléru (users.component.ts) a souboru se styly (users.component.scss).

7.1.1.1 Soubor šablony

V souboru šablony se definuje rozložení elementů na stránce. Nachází se zde jednoduchý formulář, který uživatel používá při hledání pracovníků a jejich zakázek. V následujícím příkladu je zobrazen HTML kód tohoto formuláře s vysvětlením základních principů jeho funkcionality a propojení s třídou kontroléru.

```
1. <form>
2.   <input type="number" class="txt" name="item" placeholder="ID"
 [(ngModel)]="userId">
3.   <input type="submit" class="btn" value="Submit"
 (click)="findUser()">
4. </form>
```

Zdrojový kód 5 – Formulář v souboru šablony stránky Users

První element je vstup pro identifikační číslo pracovníka. Je zde použitý princip data bindingu frameworku Angular, kdy je hodnota vstupního elementu synchronizována s property *userId* v třídě kontroléru. To je docíleno použitím syntaxe *[(ngModel)]*, která zajistí obousměrný data binding. Další vstupní prvek je tlačítko. Zde je použit princip jednosměrného bindingu *(click)*, který využívá metodu třídy kontroléru *findUser()* jako obslužnou rutinu při vyvolání události kliknutí.

Po nalezení pracovníka se načte seznam jeho zpracovávaných zakázek. Tento proces se implementuje pomocí Angular direktivy *ngFor*, která je použita v HTML kódu na příkladu níže.

```
1. <p class="life-container" *ngFor="let order of ordersNGraph; let i
 = index" (click)="addItemY(i)">
2.   {{ order.name }}
3. </p>
```

Zdrojový kód 6 – Seznam zpracovávaných zakázek v šabloně stránky Users

Tato direktiva funguje podobně jako klasický cyklus v programovacích jazycích. V tomto případě projde všechny položky pole *ordersNGraph* (property třídy kontroléru). Tyto po-

ložky reprezentují objednávky uživatele. Pro každou položku vytvoří nový HTML element a dovnitř vloží parametr *name*, což odpovídá názvu zakázky.

Na konci souboru je specifikovaný HTML element, který na webové stránce reprezentuje generovaný graf.

7.1.1.2 Soubor třídy kontroléru

Zde je obsažena logika celé komponenty. Nejdříve jsou uvedeny všechny importy. To zahrnuje jádrové komponenty Angularu, které jsou nutné pro běh webové aplikace. Dále jsou importovány animace, které budou později definovány v dekorátoru. Nakonec jsou zde uvedeny externí knihovny, které poskytují možnost generování grafů.

Jako další následuje definice dekorátoru @Component, která je zobrazena na příkladu níže. Poté bude vysvětlen obsah tohoto dekorátoru.

```
1. @Component({
2.   selector: 'app-users',
3.   templateUrl: './users.component.html',
4.   styleUrls: ['./users.component.scss'],
5.   animations: [
6.     trigger('orders', [
7.       transition('* => *', [
8.         query(':enter', style({ opacity: 0 }), {optional: true}),
9.         query(':enter', stagger('300ms', [
10.           animate('.4s ease-in', keyframes([
11.             style({opacity: 0, transform: 'translateX(-75%)',
 offset: 0}),
12.             style({opacity: .5, transform: 'translateX(35px)',
 offset: 0.3}),
13.             style({opacity: 1, transform: 'translateX(0)',
 offset: 1.0}),
14.           ]))]), {optional: true})
15.           ,
16.         query(':leave', stagger('300ms', [
17.           animate('.4s ease-out', keyframes([
18.             style({opacity: 1, transform: 'translateX(0)', off-
 set: 0}),
19.             style({opacity: .5, transform: 'translateX(35px)',
 offset: 0.3}),
20.             style({opacity: 0, transform: 'translateX(-75%)',
 offset: 1.0}),
21.           ]))]), {optional: true})
```

```
22.         ])
23.       ])
24.     ]
25. })
```

Zdrojový kód 7 – Dekorátor komponenty Users

Obsah dekorátoru komponenty Users zahrnuje základní prvky, které by měla mít každá Angular komponenta. Jedná se o selektor, který je uvedený na druhém řádku. Následuje odkaz na soubor HTML šablony a také odkaz na soubor, který definuje styly. Ve webové aplikaci jsou použity animace. Jejich definice tedy musí být taktéž uvedena v dekorátoru.

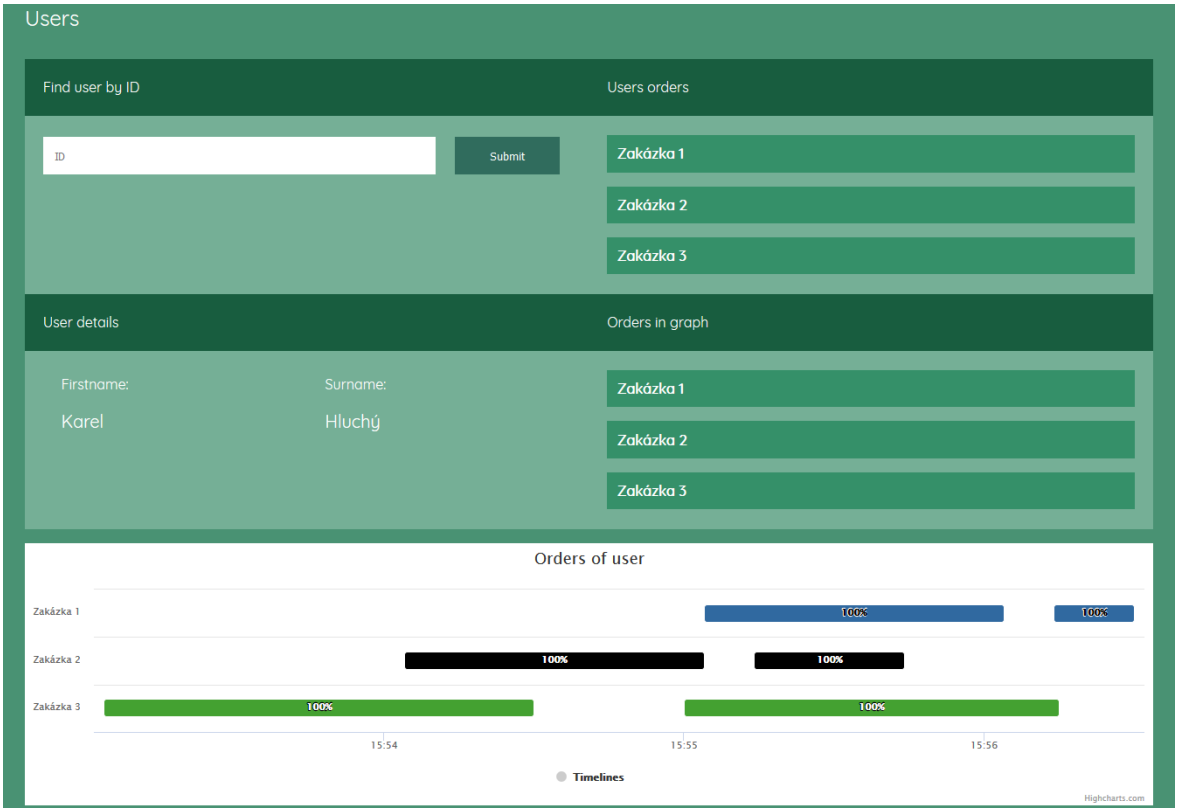
Animace jsou použity při přidávání a odebírání položek ze seznamu zakázek, které se zobrazují v grafu. V rámci animací je definovaný tzv. trigger, na který se odkazuji v souboru HTML šablony. V těle tohoto triggeru jsou vytvořeny animace pro přídání a odebrání položky. Samotné animace jsou poté definovány jako posloupnost přechodů mezi jednotlivými stavy.

7.1.1.3 Soubor stylů

V souborech stylů je definována grafická stránka elementů uživatelského rozhraní. Tyto soubory jsou dva. V souboru styles.scss jsou definovány globální pravidla, které se aplikují na elementy v rámci všech komponent. Soubor users.component.scss definuje pravidla pouze pro elementy komponenty Users.

7.2 Uživatelské rozhraní

V této kapitole bude popsáno uživatelské rozhraní a způsob, jakým uživatel interaguje s webovou stránkou.



Obrázek 9 – Uživatelské rozhraní webové stránky

Webová stránka umožňuje uživateli hledat pracovníky podle jejich identifikačního čísla a zobrazovat jejich zpracovávané zakázky.

Uživatel začne vložením identifikačního čísla pracovníka do formuláře v horní části stránky. Po kliknutí na tlačítko se skrze webové služby získají informace o příslušném pracovníkovi a jeho zpracovávaných objednávkách. Informace o pracovníkovi se objeví v sekci pod formulářem. Seznam jeho objednávek se poté objeví v pravé horní části stránky. Z tohoto seznamu je poté možné přidávat jednotlivé objednávky do dalšího seznamu, který je umístěn pod ním. Zakázky v tomto seznamu jsou poté vykresleny do grafu v dolní části stránky. Pokud bude chtít uživatel zakázku z grafu odstranit, klikne na její záznam v seznamu.

8 MOBILNÍ APLIKACE

Tato kapitola se bude věnovat popisu mobilní aplikace. Jak již vyplývá z názvu diplomové práce, aplikace slouží pro mobilní monitoring zpracování zakázek. Její vývoj je přizpůsobený požadavkům, které byly specifikovány v kapitole 6.

Základní principy celého vývoje vycházejí z filozofie webových hybridních mobilních aplikací. Ty jsou vyvíjeny pomocí webových technologií a pro běh využívají instanci webového prohlížeče WebView. Toto WebView je součástí API pro vývoj aplikací téměř na všech platformách. V současné době je tento způsob vývoje velmi používaný hlavně kvůli jednodušší údržbě a nižším nákladům na vývoj. Velmi populárním nástrojem pro vývoj webových hybridních mobilních aplikací je framework Ionic, který je použit i pro vývoj aplikace v rámci této diplomové práce.

8.1 Instalace prerekvizit

Pro využití frameworku Ionic k vývoji aplikace je nejdříve nutné mít nainstalovanou sadu nástrojů.

Nejdříve je třeba nainstalovat Java Development Kit (JDK). Nejlepší způsob je navštívit oficiální stránku a podle typu operačního systému vybrat příslušnou verzi. Po stažení stačí spustit instalační soubor a řídit se pokyny. Stejným způsobem se nainstaluje i běhové prostředí Node.js a společně s ním i manažer balíčků npm.

Dále je třeba nainstalovat SDK platformy, pro kterou se software vyvíjí. V případě této práce je součástí zadání sestavit aplikaci pro Android. Instalace Android SDK lze provést více způsoby. Pokud je na zařízení nainstalováno Android Studio, tento SDK je jeho součástí. V opačném případě je nutné jej manuálně stáhnout a nakonfigurovat prostřednictvím systémových proměnných. Tento proces zde ale vysvětlován nebude.

8.2 Instalace frameworku Ionic

Framework Ionic se nainstaluje pomocí příkazu *npm install -g ionic* v příkazové řádce. Tento příkaz nainstaluje také Ionic CLI, které bude hlavním nástrojem při vývoji a sestavování aplikace.

8.3 Operace v Ionic CLI

Jako hlavní nástroj pro vývoj aplikace v Ionic frameworku poskytuje Ionic CLI rozhraní pro tvorbu, sestavování a správu projektů. V této kapitole budou popsány základní operace, které byly při vývoji aplikace použity.

8.3.1 Založení projektu

Po nainstalování Ionicu je nyní možné použít Ionic CLI k vytvoření projektu. To se provede příkazem *ionic start dpa blank*. Spuštěním tohoto příkazu se z Git repozitáře Ionicu stáhne předpřipravená šablona projektu. Těchto šablon je více a každá má v sobě zakomponované jiné počáteční funkce. Pomocí parametru příkazu tedy existuje možnost vybrat si počáteční šablonu.

8.3.2 Generování kódu pomocí Ionic CLI

Podobně jako u Angularu i Ionic framework poskytuje možnost generování kódu projektu. Jedná se typicky o stránky, komponenty, providery, pipes a direktivy. V projektu je použito generování stránek a providerů. Příkaz použítý pro vygenerování providera je *ionic generate provider WebData*. Spuštěním tohoto příkazu se tedy vygenerují přislušné soubory a složka, čímž se změní adresářová struktura projektu.

8.3.3 Přidání platformy

Aplikace musí být podle zadání sestavena na platformu Android. Pro testovací účely byla také použita platforma Browser. Tyto platformy je tedy nutné přidat do projektu pomocí příkazu *ionic cordova platform add android*, podobně jě tomu pro platformu Browser.

8.3.4 Sestavení projektu

Projekt se v Ionic frameworku sestavuje pro každou platformu zvlášť. Příkazem *ionic cordova build android* se projekt sestaví pro platformu Android. Podobně by se postupovalo při sestavování projektu na jiné platformy.

8.3.5 Spuštění projektu

Spuštění projektu může probíhat v různých formách. Základním nástrojem pro vývoj a testování je lokální server, který se spustí příkazem *ionic serve*. Tento příkaz aplikaci spustí v prostředí webového prohlížeče. Pokud však v aplikaci používáme nativní pluginy nebo jiné hardwarové prostředky mobilního zařízení, nebude možné je v prostředí prohlížeče

správně implementovat. To může vyústit v absenci některých funkcí, v horším případě aplikace spadne úplně. Výhodou tohoto přístupu je okamžitá projekce změn provedených v kódu do prostředí webového prohlížeče.

Další alternativou jak provádět testování aplikace je použitím platformy Browser, která nabízí více možností než předchozí varianta. Aplikace je ovšem také spuštěna v prostředí webového prohlížeče. Při každém spuštění je třeba aplikaci vždy znovu sestavit. Spuštění proběhne příkazem *ionic cordova run browser*.

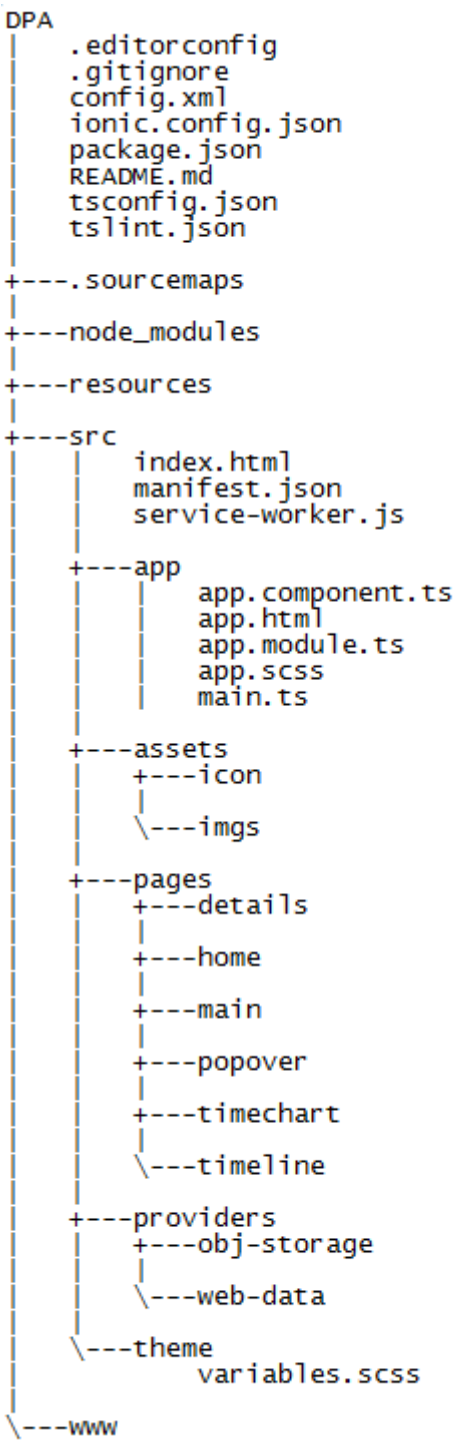
Pro spuštění aplikace na mobilním zařízení je třeba provést konfigurační změny. V rámci vývojářského módu je potřeba povolit debugging pomocí USB. Dále je nutné nainstalovat nástroj ADB na PC. Tento nástroj umožňuje komunikaci se zařízením platformy Android. Příkazem *adb devices* se poté zjistí, která zařízení jsou k PC připojena. Poté, co úspěšně proběhne konfigurace zařízení, se příkazem *ionic cordova run android* aplikace nahraje a spustí.

Podobným způsobem lze aplikaci spustit v rámci emulátoru. Ten simuluje virtuální stroj se systémem Android na lokálním PC. Stejně jako ostatní metody, které aplikaci spustí na PC, i zde nastává problém při použití některých hardwarových prostředků specifických pro mobilní zařízení.

Instalaci aplikace je také možno provést manuálně. K tomu je třeba provést změny v nastavení zařízení. Konkrétně je nutné povolit instalaci aplikací z neznámých zdrojů v sekci nastavení bezpečnosti. Poté se na zařízení zkopíruje instalační balíček, který vznikne sestavením aplikace (přípona .apk), a po následném spuštění se aplikace nainstaluje.

8.4 Struktura projektu

Na obrázku níže je zobrazena celková adresářová struktura projektu.



Obrázek 10 – Adresářová struktura projektu mobilní aplikace

Význam většiny těchto souborů a složek byl rozebrán v kapitole teoretické části 5.5 Struktura Ionic projektu. Proto bude v následující části věnována pozornost především složce src, ve které probíhala většina vývoje.

8.4.1 Složka App

V rámci této složky budou popsány soubory app.module.ts a app.component.ts. Soubory s HTML šablonou a styly obsahují původní vygenerovaný obsah Ionic projektu, který nebyl dále modifikován.

8.4.1.1 app.module.ts

Je to soubor, ve kterém jsou uvedeny všechny importy, které aplikace využívá. Kromě jádrových knihoven Ionicu a stránek s providery jsou to:

- Dialogs – poskytuje možnost vyvolání dialogových oken.
- QRScanner – plugin, který je použit při skenování QR kódů.
- IonicStorageModule – umožňuje ukládat data do paměti zařízení.
- Network – zpřístupňuje síťové funkce.
- HttpClientModule – umožňuje tvorbu HTTP dotazů.
- ChartModule – použit pro generování grafu.

Tento soubor dále obsahuje prázdnou třídu AppModule s dekorátorem @NgModule, který tuto třídu rozšiřuje. Zmíněné importy se poté uvádějí v rámci vlastností tohoto dekorátoru. Funkce jednotlivých vlastností byly popsány v teoretické části.

8.4.1.2 app.component.ts

V rámci tohoto souboru jsem definoval operace, které probíhají v kontextu celé aplikace. Jedná se zejména o sledování stavu sítě, na jehož změnu poté aplikace reaguje. V ukázce kódu je znázorněna reakce aplikace na připojení zařízení k síti.

```
1.  let connectSubscription = network.onConnect().subscribe(() => {
2.    console.log('network connected!');
3.    this.dialogs.alert('Síť nalezena, pracujete v online režimu')
4.      .then(() => {
5.        setTimeout(() => {
6.            this.synchronize();
7.        }, 3000);
8.      });
9.  });
```

Zdrojový kód 8 – Obsluha události připojení k síti

Při nalezení sítě se událost zaloguje do konzole a zobrazí se dialogové okno, které uživateli tuto událost oznámí. Poté se po krátké pauze spustí proces synchronizace.

8.4.2 Složka Pages

Složka Pages obsahuje soubory komponent, které představují stránky aplikace. Každá stránka je zde reprezentována složkou se soubory typu .ts, .html, .scss. Tyto složky jsou následující:

- home – stránka, kde se uživatel přihlašuje.
- main – stránka se seznamem zakázek, které uživatel zpracovává.
- details – stránka s detailem zákázky.
- timeline – stránka se zpracováním zakázky.
- timechart – stránka s grafem časů zpracování.
- popover – stránka vysouvacího menu.

8.4.3 Složka Providers

Projekt obsahuje celkem dva providery, kteří jsou obsaženi ve složce Providers. Základní šablona těchto providerů je vygenerována pomocí příkazu Ionic CLI. Každý z nich poskytuje vlastní funkcionalitu, která je pomocí injekce využívána v aplikaci.

8.4.3.1 WebData provider

Je definován v souboru web-data.ts. Poskytuje metody pro získávání a posílání informací informačnímu systému. V ukázce zdrojového kódu je metoda pro posílání zakázky.

```
1.  postUser(order) {
2.    const httpOptions = {
3.      headers: new HttpHeaders({
4.        'Content-Type':  'application/json',
5.      })
6.    };
7.    return new Promise((resolve, reject) => {
8.
 this.http.post('https://app.wista.cz/api/v1/mon/users/orders',
 JSON.stringify(order), httpOptions)
9.         .subscribe(res => {
10.          resolve(res);
11.        }, (err) => {
12.          reject(err);
13.          console.log(err);
14.          this.dialogs.alert('Error - zakázka')
15.          .then(() => console.log('Post error'));
```

```
16.         });
17.     });
18.   }
```

Zdrojový kód 9 – Metoda posílání zakázky providera WebData

Metoda přijímá jeden parametr, který představuje objekt zakázky. Nejdříve se specifikuje parametr hlavičky HTTP dotazu, konkrétně je nutné, aby data byla označena jako application/json. Poté se na řádku 8 použije metoda POST a dotaz se vytvoří pomocí dříve definovaného objektu hlavičky a parametru metody, který se použije jako tělo. Následují dvě obslužné metody, který pokryjí stavy úspěšného a neúspěšného zpracování.

8.4.3.2 ObjStorage provider

Tento provider se stará o správu dat v lokálním uložišti zařízení. Je definován v souboru obj-storage.ts a poskytuje metody pro přístup, získání a uložení dat. Funkcionalitu těchto operací zprostředkovává třída Storage, která je na začátku souboru importována a poté skrze konstruktor injektována. Data se vždy ukládají v páru klíč/hodnota. V kontextu aplikace má každý uživatel svůj vlastní záznam, jehož klíč odpovídá uživatelovu identifikačnímu číslu.

8.4.4 Složka Theme

V této složce je soubor variables.scss, který obsahuje definici všech proměnných v kontextu stylů. K těmto proměnným poté může být přistupováno v rámci celé aplikace. Hlavní prvek, který byl pomocí proměnných definován, jsou barvy.

8.5 Ukládání dat

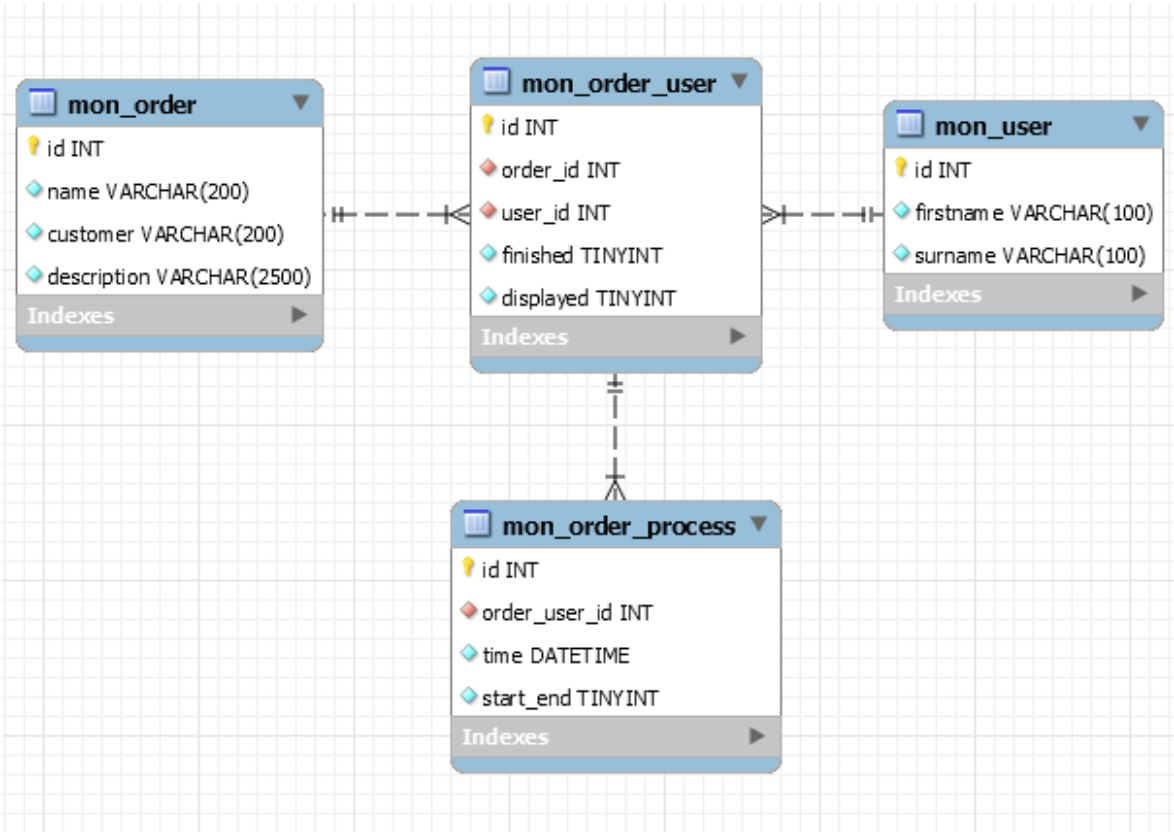
Při používání mobilní aplikace je nutné zajistit ukládání informací o zpracovávaných zakázkách. To se provádí využitím dvou přístupů. První přístup využívá lokální uložiště zařízení pomocí pluginu SQLite. Druhý přístup využívá databázovou strukturu informačního systému, se kterým mobilní aplikace komunikuje pomocí rozhraní webových služeb.

8.5.1 Pamět mobilního zařízení

Ukládání do paměti lokálního zařízení zprostředkovává služba Ionic Storage. Jako systém pro ukládání dat byl zvolen SQLite. Do projektu ho lze přidat pomocí příkazu *ionic cordova plugin add cordova-sqlite-storage*. Princip ukládání dat a třída Storage byla popsána v kapitole 5.8 Ukládání dat.

8.5.2 Databáze na straně informačního systému

Informace o zpracování zakázek jsou také poskytnuty informačnímu systému prostřednictvím webových služeb. Tyto data jsou uložena v databázové struktuře, která je znázorněna na obrázku níže.



Obrázek 11 – Struktura tabulek na straně informačního systému

Celková struktura je navržena tak, aby uživatel mohl zpracovávat více zakázek a naopak aby jedna zakázka mohla být zpracovávána více uživateli. Dále je popsán význam jednotlivých tabulek:

- mon_order – tabulka, která reprezentuje zpracovávanou zakázku. Obsahuje atributy, které reprezentují základní informace, jež uživatel při zpracování potřebuje.
- mon_user – tabulka, která reprezentuje uživatele. Obsahuje základní informace o uživateli.
- mon_order_user – tabulka, která reprezentuje záznam zpracování. Obsahuje identifikaci uživatele a zakázky společně s dalšími informacemi o záznamu zpracování.
- mon_order_process – tabulka, která reprezentuje časovou informaci o záznamu zpracování. To může být začátek nebo zastavení zpracovávání zakázky. Dále je tu

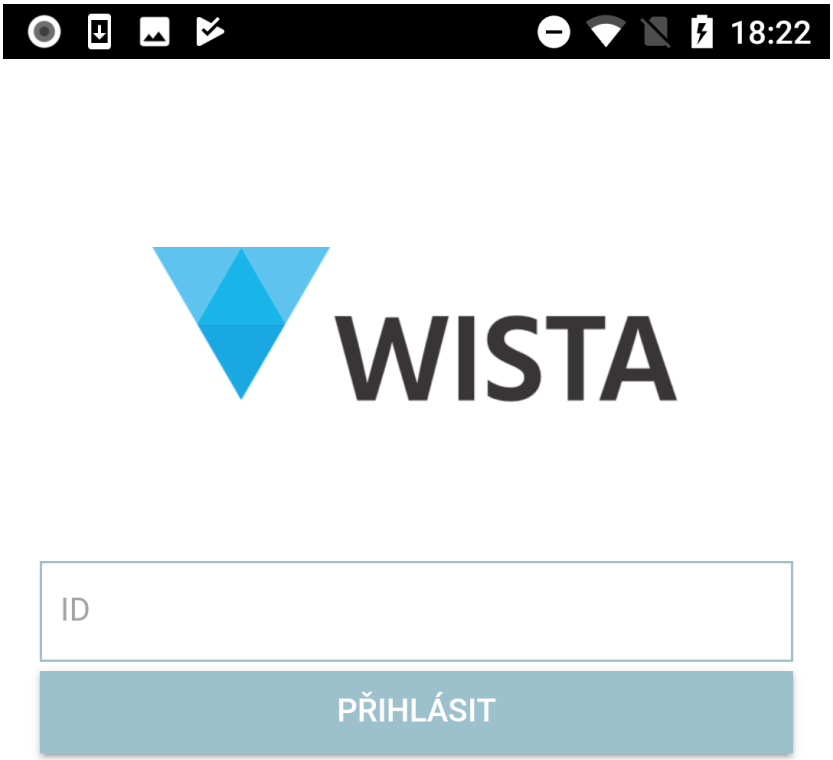
identifikace konkrétního záznamu zpracování, ke kterému je časová informace vztažena.

8.6 Uživatelské rozhraní

Elementy uživatelského rozhraní aplikace, která je vyvíjena ve frameworku Ionic, se snaží simulovat vzhled nativních komponent vývojového API konkrétních platforem. S tím souvisí i simulace animací a celkového pocitu při práci s aplikací. V rámci této práce je mobilní aplikace sestavena pro platformu Android, od toho se také odvíjí celkový vzhled aplikace.

V této kapitole bude popsáno uživatelské rozhraní a způsob, jakým uživatel s aplikací interaguje. Popis bude proveden pro každou stránku uživatelského rozhraní.

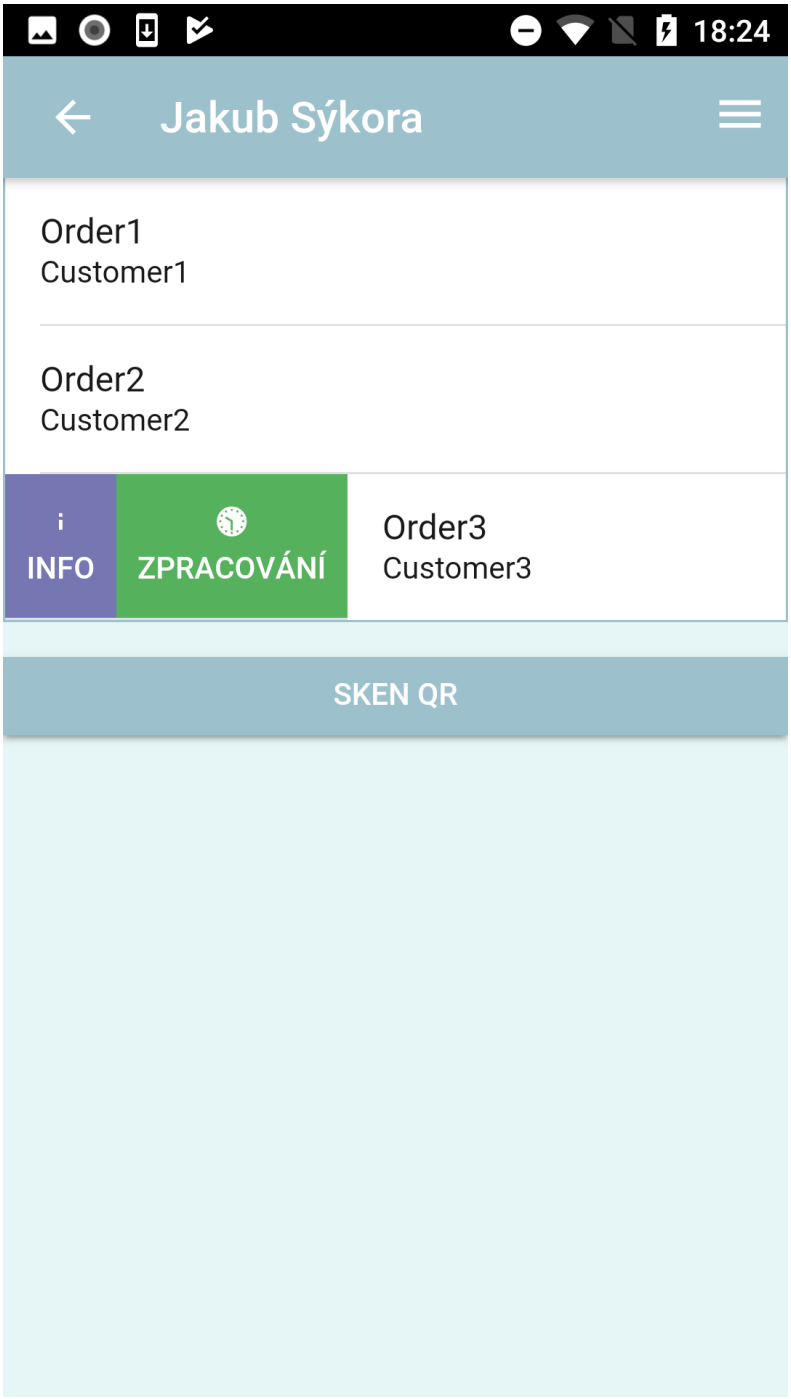
8.6.1 Stránka přihlášení



Obrázek 12 – Stránka přihlášení

Po spuštění aplikace se jako první zobrazí stránka přihlášení. Je zde vidět jednoduchý formulář se vstupním polem a tlačítkem. Pro přihlášení je nutné zadat identifikační číslo uživatele do vstupního pole a poté potvrdit tlačítkem Přihlásit.

8.6.2 Hlavní stránka



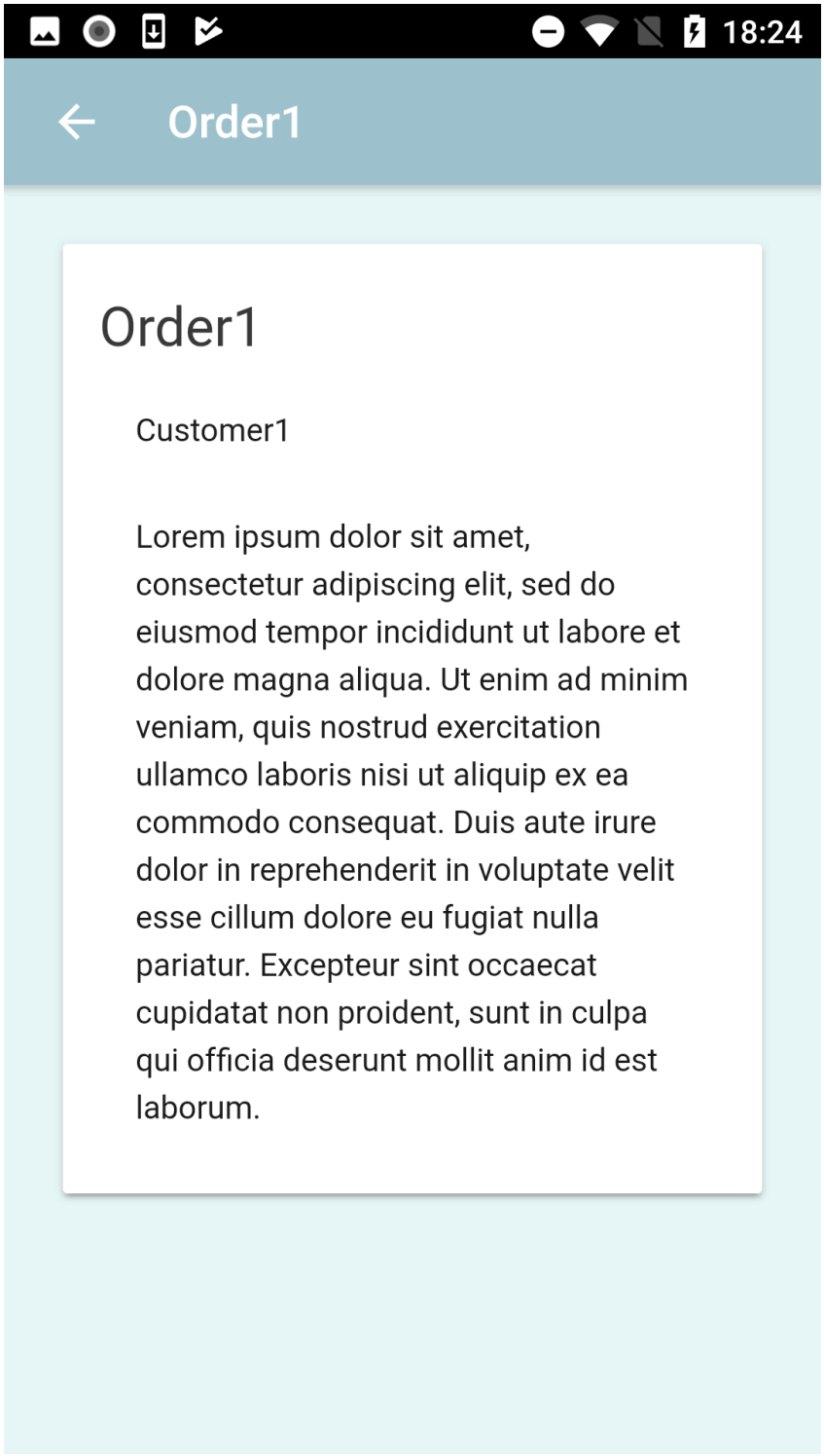
Obrázek 13 – Hlavní stránka

Po přihlášení uživatele se zobrazí hlavní stránka, která obsahuje jeho zpracovávané zakázky. Zakázky lze do seznamu přidávat skenováním QR kódů. Každá zakázka má svůj unikátní QR kód, po jehož naskenování se přidá do seznamu zpracovávaných zakázek. U každé zakázky jsou k dispozici tři tlačítka. Přejetím přes zakázku doprava se na levé straně

zobrazí dvě tlačítka Info a Process. Přejetím doleva se na pravé straně zobrazí tlačítko Delete.

- Info – po stisknutí tlačítka se zobrazí stránka s detailem zakázky.
- Zpracování – po stisknutí tlačítka se zobrazí stránka zpracování zakázky.
- Smazat – po stisknutí tlačítka se daná zakázka smaže ze seznamu.

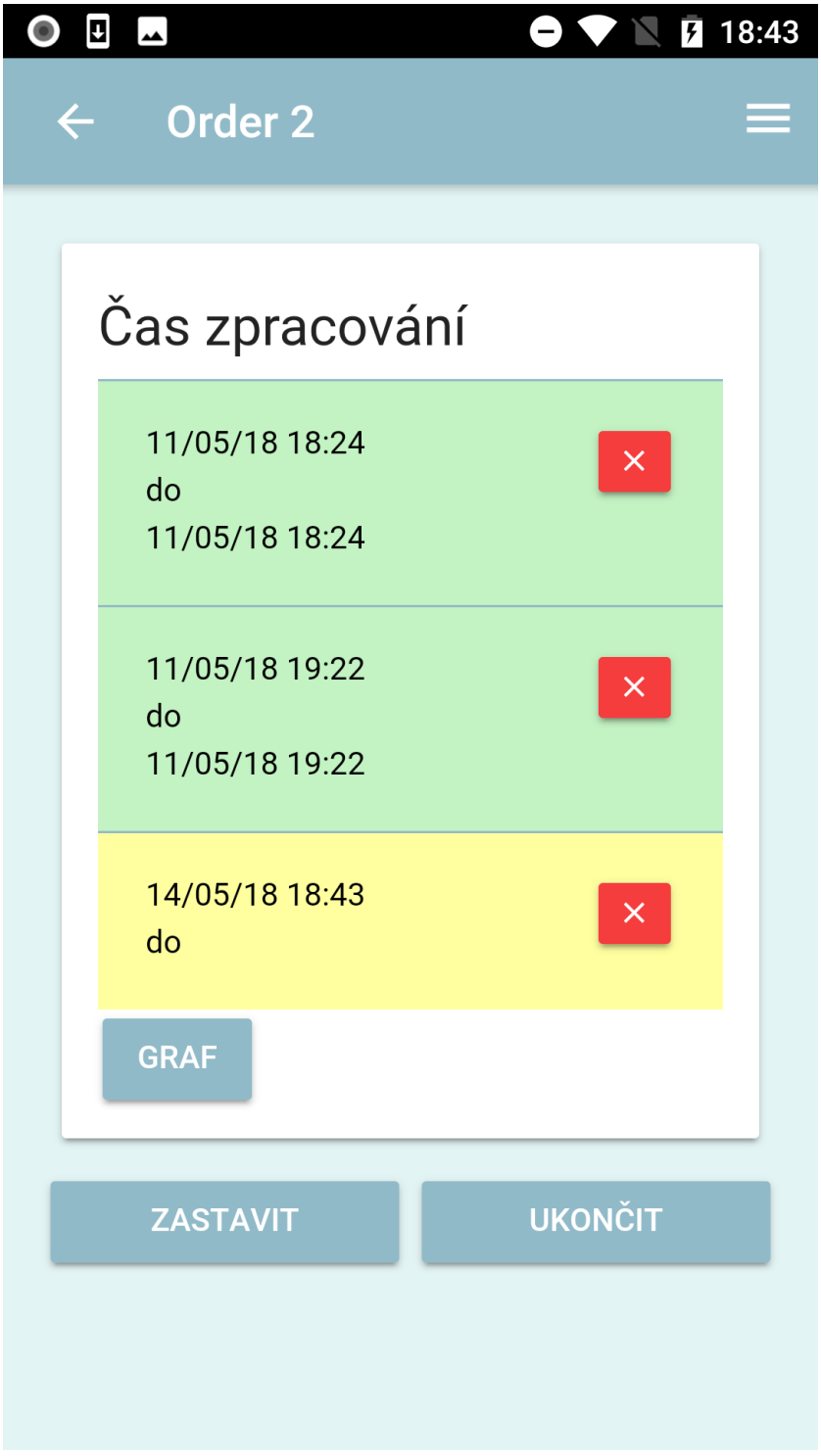
8.6.3 Stránka detailu zakázky



Obrázek 14 – Stránka detailu zakázky

Tato stránka obsahuje detail zakázky, ve kterém je obsažen název zakázky, zákazník a popis zakázky.

8.6.4 Stránka zpracování zakázky

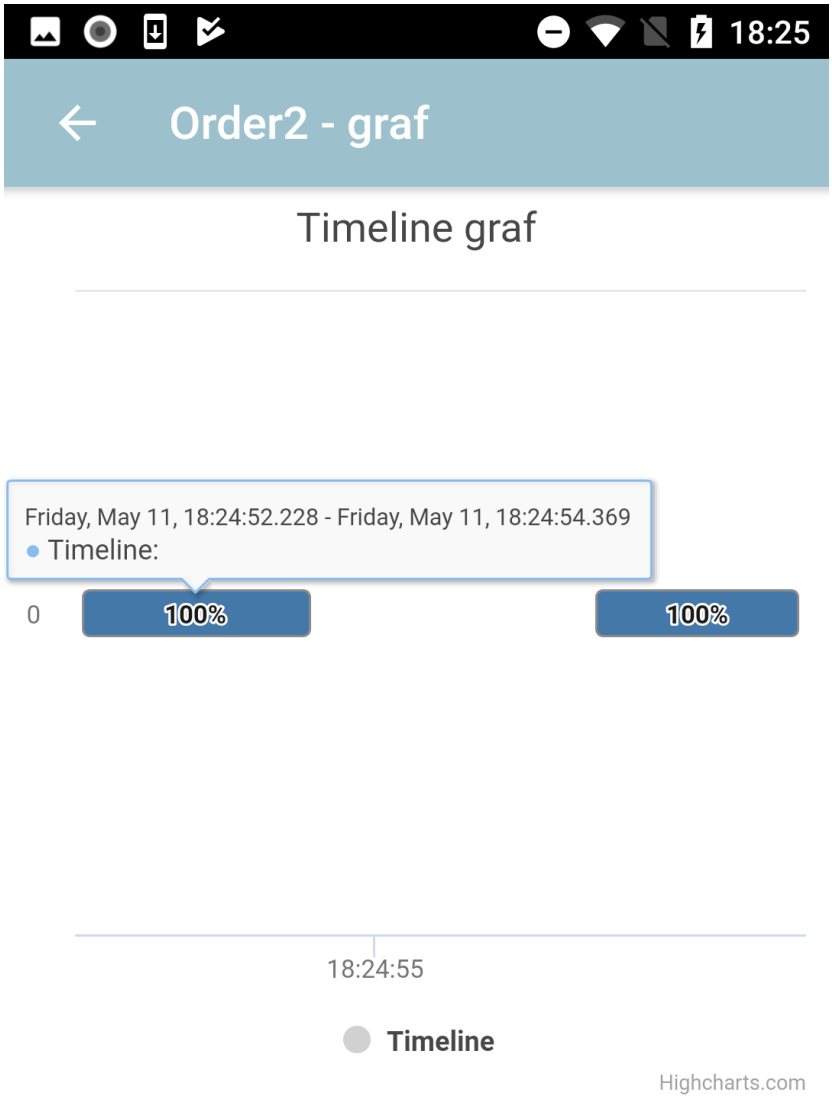


Obrázek 15 – Stránka zpracování zakázky

Tato stránka obsahuje časové informace o zpracovávané zakázce. Ukončené úseky zpracování jsou označeny zeleně, neukončené úseky žlutě. Každý úsek lze ze zpracování odstranit klinutím na červené tlačítko. Pod úseky se nachází tlačítka Graf, Start/Zastavit a Ukončit.

- tlačítko Graf – stisknutím tohoto tlačítka se zobrazí stránka s grafem.
- tlačíkto Start/Zastavit – stisknutím tohoto tlačítka se spustí/zastaví zpracování zakázky.
- tlačítko Ukončit – stisknutím tohoto tlačítka se ukončí zpracování zakázky.

8.6.5 Stránka grafu zpracování



Obrázek 16 – Stránka grafu zpracování

Na této stránce je zobrazen graf, který je vytvořen z časových údajů zpracování zakázky.

8.7 Sestavení a testování

Tato kapitola obsahuje postupy sestavení a možnosti testování vytvořené mobilní aplikace. Dle zadání je aplikace sestavována pro platformu Android. Aplikace vytvořené frameworkem Ionic lze však sestavovat na více platforem. Jelikož se jedná o webovou hybridní mobilní aplikaci, která se vyvíjí pomocí webových technologií, lze toto sestavení provést velice jednoduše pomocí příkazu Ionic CLI. Žádné další úpravy kódu vzhledem k ostatním platformám nejsou potřeba.

Sestavení aplikace lze provést příkazem *ionic cordova build android*. Tento příkaz vytvoří ve složce ..\platforms\android\build\outputs\apk soubor s příponou .apk, což je klasický instalační balíček platformy Android.

Testování aplikace probíhá ve všech fázích jejího vývoje. Nejrychlejší cestou jak aplikaci testovat je přes lokální server, který se spustí příkazem *ionic serve*. Další možností je platforma Browser. Oba tyto přístupy aplikaci spustí ve webovém prohlížeči, což způsobí, že některé funkce aplikace, které využívají konkrétní hardware mobilního zařízení, nebudou dostupné. Obecně některé funkce pluginů Ionic Native budou nefunkční, v horším případě aplikace zahlásí chybovou hlášku.

Aplikace byla také testována na mobilních zařízení s operačním systémem Android. V tomto případě je nutné, aby na zařízení byl nainstalovaný systém Android ve verzi 4.4 nebo vyšší. Tyto verze pokrývají téměř 95 % všech zařízení. Existuje však možnost, jak toto číslo dále rozšířit a přidat podporu zařízení se starší verzí systému instalací pluginu Crosswalk. Tento plugin zajistí podporu zařízení s operačním systémem Android verze 4.1 a vyšší, což v současné době pokrývá přes 99 % všech zařízení. Použití tohoto pluginu však způsobí nárůst velikosti aplikace, což může být někdy až v řádu desítek MB. V mobilní aplikaci, která byla vytvořena v rámci této práce, tento plugin použit není.

ZÁVĚR

Cílem této práce bylo vytvořit aplikaci pro mobilní monitoring zpracování zakázek. Hlavní funkcí této aplikace je poskytnout uživateli na pracovišti jednoduchý a rychlý prostředek k zaznamenávání časových informací o zakázkách, které momentálně zpracovává. Součástí práce je také návrh webového rozhraní, které umožňuje uživateli zobrazení výstupů této aplikace.

V teoretické části jsou popsány technologie, které byly při vývoji aplikace použity. Jedná se zejména o framework Ionic, který byl pro vývoj aplikace použit, a související technologie jako Apache Cordova, Angular a TypeScript. V rámci Ionic frameworku jsou dále zmíněny nástroje, které jsou pro vývoj nezbytné. To zahrnuje především běhové prostředí Node.js s manažerem balíčků npm. Je zde zmíněn také systém správy verzí Git.

V praktické části byla nejdříve vypracována sekce požadavků na aplikaci. Tato sekce obsahuje definici funkčních požadavků, nefunkčních požadavků a návrh diagramu případů užití. Dále je zde popsána webová stránka, která byla vytvořena pomocí frameworku Angular, a slouží pro zobrazení výstupů mobilní aplikace. Tento popis zahrnuje vysvětlení struktury projektu s uvedením hlavních souborů a rozebráním jejich obsahu. Je zde také popsáno uživatelské rozhraní této stránky.

Další sekce se zabývá vývojem mobilní aplikace. Ta byla vytvořena pomocí frameworku Ionic, jehož instalace a instalace dalších prerekvizit je zde popsána. Dále je zde uvedena struktura projektu s popisem implementace hlavních funkcí aplikace. Práce obsahuje také popis uživatelského rozhraní této aplikace. Zde jsou zobrazeny jednotlivé stránky s vysvětlením jejich významu a způsobu, jakým s nimi uživatel interaguje. Poslední částí této sekce je kapitola sestavení a testování, ve které jsou rozebrány způsoby testování aplikace a postup jejího sestavení.

SEZNAM POUŽITÉ LITERATURY

[2] What is a Hybrid Mobile App?. *Mendix* [online]. Boston, Massachusetts, USA: Mendix, 2018 [cit. 2018-04-26]. Dostupné z: https://www.mendix.com/what-is-a-hybrid-mobile-app/

[2] Hybrid app architecture. *Telerik* [online]. Sofie, Bulharsko: Telerik, 2018 [cit. 2018-04-26]. Dostupné z: https://developer.telerik.com/wp-content/uploads/2015/03/hybrid-app-architecture.png

[3] What is a Hybrid Mobile App?. *Telerik* [online]. Sofie, Bulharsko: Telerik, 2018 [cit. 2018-04-26]. Dostupné z: https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/

[4] PhoneGap, Cordova, and what's in a name?. *PhoneGap* [online]. San José, Kalifornie, USA: Adobe Systems, 2018 [cit. 2018-04-26]. Dostupné z: https://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/

[5] Architectural overview of Cordova platform - Apache Cordova. Apache Cordova [online]. Forest Hill, Maryland, USA: The Apache Software Foundation, 2018 [cit. 2018-04-24]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[6] Cordova Docs / Website. GitHub [online]. San Francisco, USA: GitHub, 2018 [cit. 2018-04-24]. Dostupné z: https://github.com/apache/cordova-docs

[7] Introduction to AngularJS. *W3Schools Online Web Tutorials* [online]. Sandnes, Norsko: Refsnes Data, 2018 [cit. 2018-04-26]. Dostupné z: https://www.w3schools.com/angular/default.asp

[8] Angular - Architecture overview. *Angular* [online]. Mountain View, Kalifornie, USA: Refsnes Data, 2018 [cit. 2018-04-26]. Dostupné z: https://angular.io/guide/architecture

[9] Angular 4 Module. *Tutorialspoint* [online]. Hyderabad, Indie: Tutorialspoint, 2018 [cit. 2018-04-26]. Dostupné z: https://www.tutorialspoint.com/angular4/angular4_module.htm

[10] Angular - Introduction to components. Angular [online]. Mountain View, Kalifornie, USA: Refsnes Data, 2018 [cit. 2018-04-26]. Dostupné z: https://angular.io/guide/architecture-components

[11] Angular 4 Services. *Tutorialspoint* [online]. Hyderabad, Indie: Tutorialspoint, 2018 [cit. 2018-04-26]. Dostupné z: https://www.tutorialspoint.com/angular4/angular4_services.htm

[12] Angular - Angular Dependency Injection. Angular [online]. Mountain View, Kalifornie, USA: Refsnes Data, 2018 [cit. 2018-04-26]. Dostupné z: https://angular.io/guide/dependency-injection

[13] TypeScript—JavaScript with superpowers. *FreeCodeCamp* [online]. San Francisco, Kalifornie, USA: freeCodeCamp, 2018 [cit. 2018-04-26]. Dostupné z: https://medium.freecodecamp.org/typescript-javascript-with-super-powers-a333b0fcabc9

[14] Is it worth migrating from JavaScript to TypeScript today?. *Quora* [online]. Mountain View, Kalifornie, USA: Quora, 2018 [cit. 2018-04-26]. Dostupné z: https://www.quora.com/Is-it-worth-migrating-from-JavaScript-to-TypeScript-today

[15] An Introduction to TypeScript: Static Typing for the Web. *SitePoint* [online]. Melbourne, Austrálie: SitePoint, 2018 [cit. 2018-04-27]. Dostupné z: https://www.sitepoint.com/introduction-to-typescript/

[16] Basic Types - TypeScript. *TypeScript - JavaScript that scales* [online]. Redmond, Washington, USA: Microsoft, 2018 [cit. 2018-04-27]. Dostupné z: https://www.typescriptlang.org/docs/handbook/basic-types.html

[17] TypeScript Overview. *Tutorialspoint* [online]. Hyderabad, Indie: Tutorialspoint, 2018 [cit. 2018-04-27]. Dostupné z: https://www.tutorialspoint.com/typescript/typescript_functions.htm

[18] Interfaces - TypeScript. *TypeScript - JavaScript that scales* [online]. Redmond, Washington, USA: Microsoft, 2018 [cit. 2018-04-27]. Dostupné z: https://www.typescriptlang.org/docs/handbook/interfaces.html

[19] TypeScript Classes. *Tutorialspoint* [online]. Hyderabad, Indie: Tutorialspoint, 2018 [cit. 2018-04-27]. Dostupné z: https://www.tutorialspoint.com/typescript/typescript_classes.htm

[20] Modules - TypeScript. *TypeScript - JavaScript that scales* [online]. Redmond, Washington, USA: Microsoft, 2018 [cit. 2018-04-27]. Dostupné z: https://www.typescriptlang.org/docs/handbook/modules.html

[21] Ionic From Scratch: What Is Ionic?. *Learn How To Code by Envato Tuts+* [online]. Melbourne, Austrálie: Envato, 2017 [cit. 2018-04-27]. Dostupné z: https://code.tutsplus.com/tutorials/ionic-from-scratch-what-is-ionic--cms-29323

[22] Is there a graphic diagram available that shows how Ionic 2 and all related technologies connect?. In: *Ionic* [online]. Madison, Wisconsin, USA: Drifty Co., 2017 [cit. 2018-04-27]. Dostupné z: https://forum.ionicframework.com/t/is-there-a-graphic-diagram-available-that-shows-how-ionic-2-and-all-related-technologies-connect/88375/15

[23] RAVULAVARU, Arvind. Learning Ionic - Second Edition. 2nd ed. Birmingham, Spojené království: Packt Publishing, 2017. ISBN 9781786466051.

[24] CHENG, Fu. Build mobile apps with Ionic 2 and Firebase. New York, NY: Springer Science Business Media, 2017. ISBN 978-148-4227-367.

[25] About | Node.js. *Node.js* [online]. San Francisco, Kalifornie, USA: Linux Foundation, 2018 [cit. 2018-04-27]. Dostupné z: https://nodejs.org/en/about/

[26] What is npm?. *Npm* [online]. Oakland, Kalifornie, USA: npm, 2018 [cit. 2018-04-27]. Dostupné z: https://docs.npmjs.com/getting-started/what-is-npm

[27] About - Git. *Git* [online]. New York, USA: Software Freedom Conservancy, 2018 [cit. 2018-04-27]. Dostupné z: https://git-scm.com/about

[28] What's the difference between committing and pushing in Git?. *Quora* [online]. Mountain View, Kalifornie, USA: Quora, 2018 [cit. 2018-04-27]. Dostupné z: https://www.quora.com/Whats-the-difference-between-committing-and-pushing-in-Git

[29] Exploring App Module And App Component In An Ionic 2+ App. *Gone Hybrid | Angular and Ionic Tutorials* [online]. Cork, Irsko: Gone Hybrid, 2017 [cit. 2018-04-27]. Dostupné z: https://gonehybrid.com/exploring-app-module-and-app-component-in-an-ionic-2-app/

[30] Ionic Component Documentation. *Ionic* [online]. Madison, Wisconsin, USA: Drifty Co., 2018 [cit. 2018-04-27]. Dostupné z: https://ionicframework.com/docs/components/#overview

[31] NavController - Ionic API Documentation - Ionic Framework. *Ionic* [online]. Madison, Wisconsin, USA: Drifty Co., 2018 [cit. 2018-04-27]. Dostupné z: https://ionicframework.com/docs/api/navigation/NavController/

[32] Storage - Ionic Framework. *Ionic* [online]. Madison, Wisconsin, USA: Drifty Co., 2018 [cit. 2018-04-27]. Dostupné z: https://ionicframework.com/docs/storage/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
CLI	Command Line Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LTS	Long Term Support
PC	Personal Computer
SDK	Software Development Kit
URL	Uniform Resource Locator

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA I: CD