Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Mobilní aplikace pro tvorbu objednávek

Bc. Martin Štěpánek

Diplomová práce
2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	[redacted]
Osobní číslo:	[redacted]
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Úroveň podpory typografie v současných kancelářských editorech**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V teoretické části budou popsána základní pravidla tvorby zejména odborného textu; včetně struktury textu, hladké, pořadové, technické a smíšené sazby.
Bude zhodnocena a porovnána úroveň podpory typografické sazby v současných kancelářských textových i tabulkových editorech (zejm. balíků od Microsoftu, LibreOffice a dokumenty Google).
Pro nepodporované vlastnosti budou uvedeny postupy, jak lze výsledku dosáhnout (nebo se mu přiblížit) alternativními způsoby (pokud je to možné).
Dále budou vytvořeny šablony pro psaní odborného textu (zejm. závěrečné práce), které budou splňovat typografická pravidla a budou odpovídat požadavkům UPa, a to pro editory MS Word, LibreOffice Writer a pro sázecí systém TeX ve variantách plain, LaTeX a ConTeXt. Šablony budou obsahovat nachystané stránkové, odstavcové, znakové (vyznačovací) i výčtové styly (včetně seznamů literatury, obrázků, obsahu apod.). Požadavky jednotlivých fakult budou konzultovány s příslušnými odpovědnými osobami. Šablony budou dodány ve variantách pro všechny fakulty a budou předány příslušným orgánům univerzity či fakult pro zveřejnění na webu či intranetu Univerzity. Šablona pro TeX bude vypracována jediná, potřebná nastavení (typ práce a fakulta) se bude volit příkazem či stylovým parametrem.

Rozsah grafických prací:	
Rozsah pracovní zprávy:	**cca 40–50 stran**
Forma zpracování diplomové práce:	**tištěná**
Seznam odborné literatury:	**viz příloha**

Vedoucí diplomové práce:	**Mgr. Tomáš Hudec** Katedra informačních technologií
Datum zadání diplomové práce:	**31. října 2015**
Termín odevzdání diplomové práce:	**13. května 2016**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2015

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 18. 5. 2018

Martin Štěpánek

PODĚKOVÁNÍ

Na tomto místě bych chtěl poděkovat všem, kteří mi při tvorbě této práce pomohli, poradili nebo mě inspirovali. Snad jsem na nikoho nezapomněl.

ANOTACE

Hlavní náplní této práce je vývoj mobilní aplikace pro podporu tvorby objednávek prodejci obchodní společnosti. V teoretické části práce jsou přiblíženy a srovnány vybrané metody vývoje mobilních aplikací. Důraz je kladen především na klady a zápory jednotlivých metod, dále pak na jejich společné či rozdílené vlastnosti. Jeden z popsaných postupů je následně na základě daných kritérií zvolen k implementaci dříve zmíněné prodejní aplikace.

V druhé kapitole jsou stručně představeny další technologie použité při vývoji této aplikace. Jsou rozděleny na technologie potřebné pro implementaci mobilní části aplikace a technologie části serverové.

Třetí kapitola popisuje samotný vývoj aplikace, od prvotních požadavků na aplikaci, přes návrh, až po samotnou implementaci. Výsledná aplikace je poté více přiblížena v kapitole čtvrté formou popisu jejích funkcionalit z pohledu potenciálního uživatele.

KLÍČOVÁ SLOVA

Mobilní aplikace, hybridní aplikace, webové technologie, Ionic, Angular, TypeScript

TITLE

Mobile App Facilitating Order Creation

ANNOTATION

The primary aim of this master's thesis is to develop a mobile app facilitating order creation. It can be used by salesmen of a sample trading company. In the theoretical part, different methods available when building mobile apps are described and compared, while stressing out their strong and weak points. Based on the criteria of the developed application, one of those methods is chosen to be used while implementing the app.

In the second chapter, other technologies used during the development are briefly described. They are divided into technologies used for the mobile app and those used on the server side.

In the third part, the development itself is illustrated. Initial requirements, concept of the application and the implementation are presented there. The application guidelines as if seen by the potential user are included in the last chapter.

KEYWORDS

Mobile app, hybrid app, web technologies, Ionic, Angular, TypeScript

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

CLI	Command Line Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
Sass	Syntactically Awesome Style Sheets
SQL	Structured Query Language
TS	TypeScript

ÚVOD

Hlavním tématem této diplomové práce jsou mobilní aplikace. Nejprve jsou vysvětleny základní pojmy v kontextu této práce. Následně jsou uvedeny postupy dostupné při implementaci mobilních aplikací. U každého z nich jsou zdůrazněny jeho výhody a nevýhody, případně podobnosti s ostatními metodami. Dále jsou tyto metody vývoje porovnány a pro každou z nich jsou navrženy doporučené scénáře užití. Porovnání shrnuje tabulka vložená formou přílohy.

Primárním praktickým cílem práce je vyvinout pomocí vhodně zvoleného postupu mobilní aplikaci s pracovním názvem *eKatalog*, podporující prodej produktů obchodní společnosti jejím zákazníkům. Tato aplikace by zároveň měla splňovat kritéria popsaná v zadání této práce. Hlavní funkcí aplikace je nabídka produktů obchodními prodejci, tvorba objednávek pomocí dostupných údajů o zákaznících a produktech společnosti a odeslání těchto objednávek do firemního systému k jejich zpracování.

Jedním z dalších kritérií při vývoji aplikace je přehlednost a udržitelnost zdrojového kódu usnadňující případné úpravy aplikace. Tato vlastnost by měla umožnit aplikaci snadno modifikovat podle požadavků konkrétní obchodní společnosti, která by se ji rozhodla při tvorbě svých objednávek využívat.

Vedlejším cílem je implementace serverové části práce simulující informační systém obchodní společnosti. Tato část slouží především k demonstraci jednotlivých funkcionalit aplikace. Serverová část využije pro ukládání dat MySQL databázi, ke které bude aplikace přistupovat pomocí webového serveru implementovaného za použití technologie Node.js.

Vývoj celé aplikace je popsán v posledních třech kapitolách. Nejprve jsou stručně představeny technologie použité při jejím vývoji, následně je přiblížen návrh aplikace a postup její implementace. V poslední části jsou představeny funkcionality výsledné aplikace z pohledu potenciálního uživatele nabízejícího produkty fiktivní obchodní společnosti.

1 MOBILNÍ APLIKACE

Mobilní telefony jsou v dnešní době pro většinu z nás nedílnou součástí života. Začaly se objevovat již v poslední čtvrtině minulého století, nicméně pouze v podobě pro dnešní generaci již zapomenuté. Jak se vyvíjely, rozšířilo se pole jejich působnosti. Přístroje původně určené k přenášení hlasových zpráv od jednoho uživatele k druhému se staly univerzálním pomocníkem při našich každodenních činnostech. Psaní textových zpráv, kalkulačka, pořizování, zobrazování a přehrávání multimédií tvoří dohromady jen zlomek potenciálního přínosu zařízení, které má dnes téměř každý na stole, v kapse, nebo v šuplíku. Samostatným tématem jsou mobilní hry. Tyto a spoustu dalších funkcí přinesl nástup chytrých zařízení, a především mobilních aplikací. Tímto slovním spojením se rozumí software navržený a vyvinutý tak, aby byl spustitelný na mobilním zařízení, jako je chytrý telefon nebo tablet [1].

Je to téměř 35 let od okamžiku, kdy spoluzakladatel společnosti Apple Steve Jobs na konferenci v Aspenu[1] prezentoval svou vizi týkající se nákupu a šíření softwaru pomocí telefonních sítí [2]. Právě tato vize v mírně upravené podobě nejspíše vedla o 25 let později ke vzniku prvního internetového obchodu s mobilními aplikacemi – projektu App Store společnosti Apple. Další podobné projekty na sebe nenechaly dlouho čekat.

Před tím, než byly tyto obchody spuštěny, byl uživatel mobilního telefonu odkázán na výrobce jeho zařízení a mobilního operátora, kteří rozhodli o tom, jaké aplikace bude zařízení obsahovat. Jejich nástup však vytvořil místa, kde mohou společnosti i jednotlivci své aplikace nabízet, a odkud mohou uživatelé stahovat právě ty aplikace, které sami preferují. O popularitě služby App Store a jí podobných svědčí nejen statistiky jejich návštěvností, prodejů a počtu stažených aplikací [3], ale i reklama společnosti Apple vydaná jen rok po spuštění jejich obchodu, v níž tvrdí, že v obchodě lze najít mobilní aplikaci *„asi tak na všechno“*[2] [4].

1.1 Mobilní platformy

Mobilní zařízení lze rozdělit do skupin podle jejich platformy. Tento pojem označuje soubor hardwarových a softwarových technologií, které slouží jako prostředí pro vývoj a běh dalších aplikací. Pro zjednodušení lze za platformu označit operační systém, který je na zařízení spuštěn. V kontextu mobilních zařízení jsou nejrozšířenějšími platformami iOS od společnosti Apple a Android společnosti Google. [5]

[1] Město v USA, stát Colorado.

[2] V anglickém originálu doslova *„There is an app for just about anything“*. Takové tvrzení je však i v dnešní době značnou nadsázkou. Stále je tedy důvod nové aplikace vyvíjet.

1.1.1 Android

V současnosti nejrozšířenějším mobilním operačním systémem na světě je Android spravovaný společností Google. Z důvodu jeho širokého využití se při vývoji většiny mobilních aplikací klade důraz právě na podporu této platformy. Aplikace lze na jednotlivá zařízení šířit pomocí centrálního virtuálního tržitě zvaného Google Play nebo také Play Store. [6]

Android dovoluje nejen uživatelům, ale i vývojářům využít velkou škálu nastavení systému. Vedlejším efektem jeho rozšířenosti mezi výrobci mobilních zařízení je velká fragmentace jeho verzí. Výrobci totiž do svých zařízení často instalují jimi upravenou implementaci tohoto operačního systému, což může vést k nekompatibilitě mezi jednotlivými verzemi (stejná aplikace se na zařízeních různých výrobců může chovat odlišně). [7]

1.1.2 iOS

Dalším významným mobilním operačním systémem je iOS společnosti Apple. Původně byl vyvinut pro chytré telefony této společnosti označované iPhone, později se však začal používat i na jejich tabletech iPad. Jde o druhý nejrozšířenější mobilní operační systém, k čemuž přispívá jednak jeho intuitivní ovládání, ale částečně také kult společnosti Apple a jejich zařízení. [8]

Protože je iOS dostupný pouze na zařízeních vydávaných tímto podnikem, existuje pouze omezené množství kombinací konkrétních zařízení s jednotlivými verzemi systému. Všechny jsou navíc spravovány jedinou společností. Apple zároveň klade vyšší nároky na vývojáře a jimi vyvíjené aplikace.

1.1.3 Dělení aplikací podle nasazení na platformy

Z hlediska možností jejich nasazení lze mobilní aplikace rozdělit do dvou kategorií. Na jedné straně jsou aplikace vyvíjené specificky na konkrétní (jednotnou) platformu[3], na straně druhé pak aplikace multiplatformní, které je možné nasadit na platforem více[4].

1.2 Implementace mobilních aplikací

Z pohledu implementace lze mobilní aplikace rozdělit do tří skupin. Nativní aplikace jsou vyvíjeny vždy na konkrétní platformu (patří tedy zároveň do kategorie *single-platform*) a při jejich implementaci jsou využívány nástroje a jazyky s danou platformou spojené. Do skupiny hybridních spadají aplikace implementované pomocí webových technologií, které jsou na mobilních zařízeních spouštěny pomocí aplikačního kontejneru WebView. Třetí

[3] Z angličtiny *single-platform.*
[4] Někdy též označováno jako *cross-platfrom* vývoj.

skupinou jsou aplikace kompilované, které jsou rovněž vyvíjeny pomocí webových technologií nebo jiných jazyků, jsou však následně (alespoň částečně) převedeny do nativní podoby a spouštěny obdobně jako aplikace první skupiny. [9]

1.2.1 Nativní aplikace

Tradiční možností je implementovat mobilní aplikaci formou nativní pro danou platformu. Tento způsob implementace má své kořeny již v počátcích vývoje mobilních aplikací a je stále hojně využíván. Vývoj probíhá v jazyce určeném pro danou platformu, takže není třeba aplikaci kompilovat. [9][10]

Výhody nativního přístupu tkví především v orientaci na jedinou konkrétní platformu a přizpůsobení se požadavkům této platformy. V důsledku toho mají nativní aplikace přístup ke všem funkcionalitám mobilního zařízení. To nejen usnadňuje komunikaci aplikace se zařízením, ale zároveň umožňuje lepší využití výkonu zařízení. Další výhodou pro vývojáře je nezávislost na knihovnách třetích stran – aplikaci je možné implementovat výhradně pomocí nástrojů poskytovaných v souvislosti s danou platformou, pro kterou je aplikace vyvíjena. [10]

Nativní přístup s sebou ale přináší i jistá úskalí. Největší z nich přímo vychází z jeho největší výhody – orientace na jedinou konkrétní platformu. Důsledkem toho totiž je, že aplikaci není možné nasadit na více platforem, a při snaze o pokrytí většího množství potenciálních uživatelů je třeba vyvinout pro každou platformu jinou aplikaci. Tato nevýhoda zasahuje nejen samotného vývojáře, který se buď omezí na vývoj pro konkrétní platformu, nebo se musí důkladně seznámit s každou platformou a s ní souvisejícím procesem vývoje, ale i společnost aplikaci vyvíjející, neboť ta musí sestavit a zaplatit tým vývojářů pro každou z cílových platforem, což může být finančně nákladné. [10][11]

Vývoj nativních aplikací pro Android

Nativní aplikace pro platformu Android jsou vyvíjeny v jazyce Java s využitím sady Android SDK[5]. Pro vývoj je k dispozici prostředí Android Studio. Alternativou k Javě může být od minulého roku také jazyk Kotlin [12]. Výhodou vývoje pro tuto platformu je skutečnost, že aplikace pro Android lze vyvíjet ze všech nejvíce rozšířených desktopových operačních systému – tedy Windows, MAC OS X i Linux. Jak již bylo zmíněno, jedná se o v současnosti nejrozšířenější platformou na světě, s čímž ale souvisí i značná nevýhoda tvorby aplikací pro Android – fragmentace vývoje. Platforma je podporována mnoha výrobci mobilních zařízení, z nichž každý tento volně dostupný operační systém upraví podle svých potřeb a implementuje

[5] *Software Development Kit*.

na svůj vlastní hardware. Je tedy velmi obtížné, až nemožné, otestovat správnou funkčnost aplikace na všech typech zařízení a všech verzích operačního systému. [6]

Vývoj nativních aplikací pro iOS

Při implementaci aplikace pro iOS existují také dvě možnosti volby programovacího jazyka. První z nich je použití staršího jazyka Objective C, který vznikl jako objektově orientované rozšíření známého jazyka C. Druhou cestou je využití novějšího jazyka jménem Swift vyvíjeného společností Apple. Ten by měl usnadnit psaní udržitelnějšího kódu a dovolit vývojáři méně chyb, při jeho vývoji však dochází k výrazným změnám v syntaxi [13]. Značnou nevýhodou tvorby aplikace pro iOS je nutnost vývoje na desktopovém operačním systému MAC OS X. Pro rozšíření aplikace prostřednictvím App Store je také nutné využít účet Apple Developer, za který si tato společnost účtuje finanční poplatek.

1.2.2 Hybridní aplikace

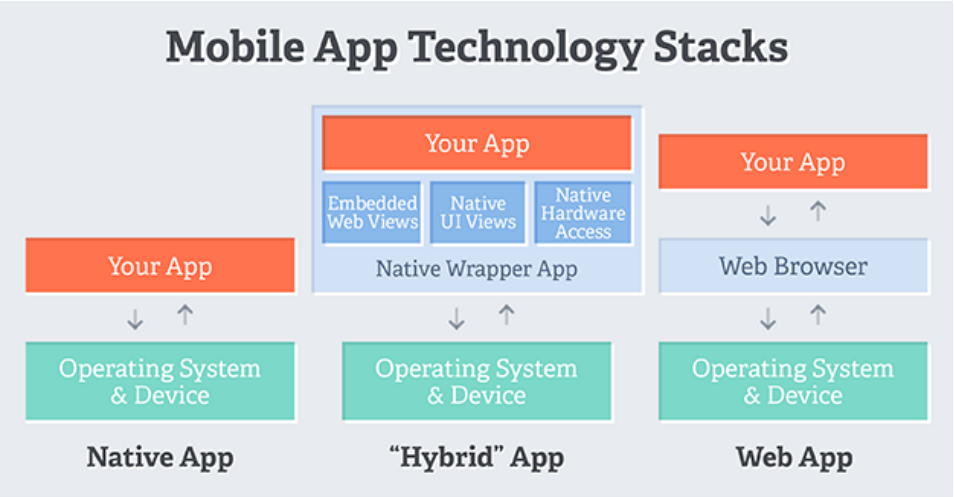
Jiným způsobem, jak zpřístupnit obsah pro mobilní zařízení, je vývoj responsivní webové aplikace. Zjednodušeně řečeno jde o běžnou webovou stránku uzpůsobenou pro zobrazení na různých velikostech displeje, tedy včetně těch menších. I tento přístup má své výhody a nevýhody. Výhodou je například přenositelnost webové aplikace, kterou lze spustit na jakémkoliv zařízení s webovým prohlížečem bez ohledu na platformu. Nevýhodou je pak rozhodně dostupnost, neboť taková aplikace je dostupná pouze na zařízeních připojených k internetové síti. [14]

Z webových aplikací vychází aplikace hybridní. Jde již o plnohodnotné mobilní aplikace, které jsou ovšem rovněž implementovány pomocí webových technologií[6]. Vnitřně se jedná stále o webové aplikace, ty jsou však obaleny kontejnerem, který je spuštěn uvnitř nativní aplikace dané platformy. Velmi zjednodušeně lze tento kontejner zvaný WebView označit za webový prohlížeč bez lišty a okrajů, který je pro uživatele mobilního zařízení zcela transparentní. Stačí tedy, aby každá platforma měla implementovaný svůj kontejner a hybridní aplikace si zachovají největší výhodu aplikací webových, tedy přenositelnost mezi platformami. [10][14]

Ačkoliv jsou hybridní aplikace spouštěny uvnitř nativního kontejneru, nejedná se o aplikace nativní. Přesnější je na ně nahlížet jako na webové aplikace spouštěné uvnitř nativního „prohlížeče“ (WebView). Hybridní aplikace řeší problém s dostupností aplikací webových – spustitelný kód je instalován přímo na mobilní zařízení, takže je možné jej spouštět i bez přístupu k internetovému připojení. Protože však nejsou zcela nativní, přístup k nativním

[6] Těmito technologiemi může být například HTML, CSS a JavaScript, jak je popsáno v kapitole 2.1.1.

funkcionalitám zařízení, které nejsou běžnému prohlížeči dostupné, závisí na použité technologii. Další nevýhodou je, že z pohledu výkonu budou hybridní aplikace vždy zaostávat za nativními, protože je za běhu interpretován kód v jazyce JavaScript, nikoliv pouze vykonáván nativní kód. Obrázek 1 ukazuje rozdíl mezi implementací nativní, hybridní a webové aplikace. [14][15]



Obrázek 1 – struktura nativní, hybridní a webové aplikace, zdroj [15]

Nástroje Cordova a PhoneGap

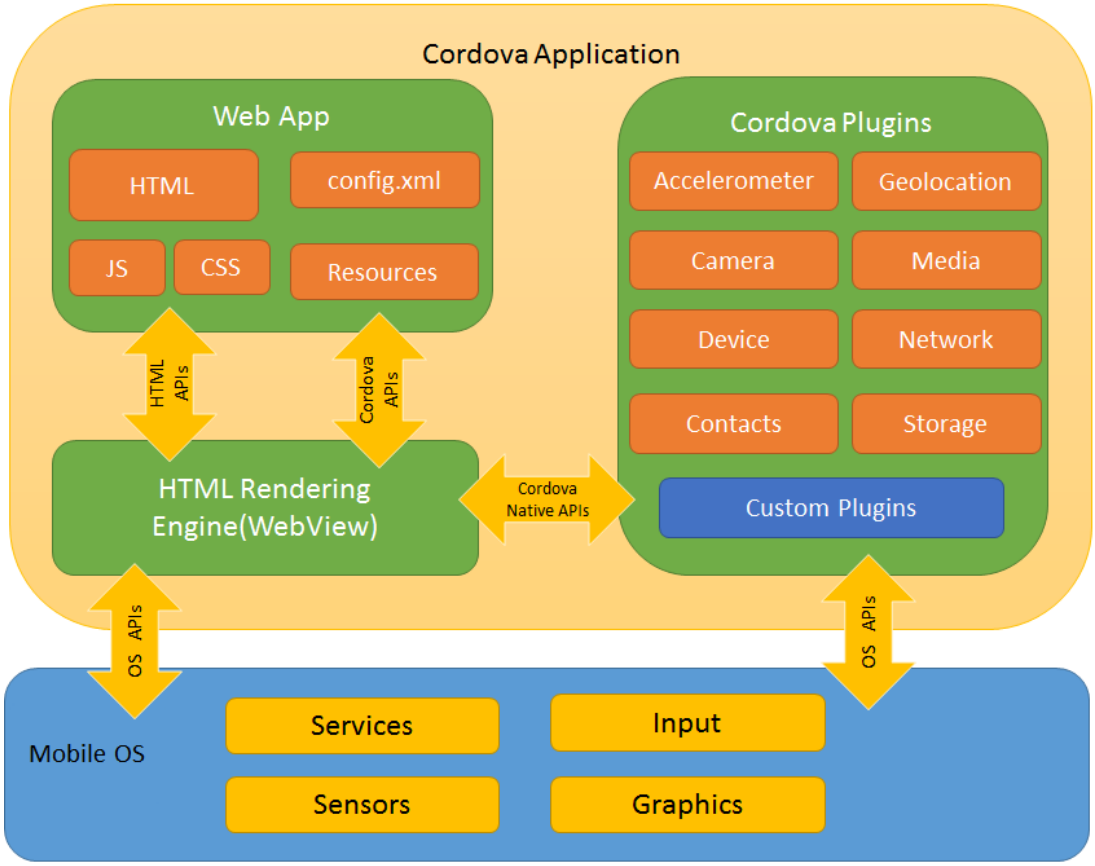
Pro vývoj hybridních aplikací existuje více nástrojů a technologií. Nejvíce skloňovanými jmény v tomto směru jsou Cordova a PhoneGap. Rozdíly mezi nimi jsou však patrné až při bližším zkoumání.

Na začátku byla společnost Nitobi a její technologie pro vývoj multiplatformních mobilních aplikací – PhoneGap. Jak se tento projekt vyvíjel, upoutal pozornost větší společnosti – Adobe. Ta v roce 2011 technologii (spolu s firmou Nitobi) koupila. O rok později pak věnovala zdrojové kódy projektu do nadace Apache Software Foundation[7] pro zajištění jejich otevřené distribuce. Zároveň však pokračovala ve vývoji své vlastní distribuce této technologie. Nadace Apache musela projekt nějakým způsobem odlišit od toho společnosti Adobe, rozhodla se proto „svůj“ PhoneGap přejmenovat na Apache Cordova. [17]

V dnešní době tedy existují dva velmi podobné nástroje – Apache Cordova, který je a bude volně dostupný pod licencí Apache License a Adobe PhoneGap, který je distribucí dříve jmenovaného, nicméně nabízející rozšířené služby, které mohou a nemusí být společností Adobe zpoplatněny. [17]

[7] ASF je nadace spravující obdržené projekty a dohlížející na jejich volnou dostupnost [16].

Co mají oba nástroje společné je struktura obalení webových aplikací a jejich spuštění na mobilním zařízení uvnitř nativní aplikace, která je popsána na obrázku 2. Obě technologie rovněž nabízejí přídavné pluginy[8] pro práci s nativními funkcemi mobilního zařízení, jako jsou kamera, geolokace nebo uložiště souborů. [18]



Obrázek 2 – struktura aplikace vyvíjené pomocí nástroje Cordova, zdroj [18]

Framework Ionic

Jednou z nevýhod výše zmíněných nástrojů je, že vývojáři nijak neusnadňují grafický návrh webových aplikací tak, aby dodržovaly konvence vžité uživatelům jednotlivých platforem. Existují totiž jisté znaky společné pro větší množství aplikací každé platformy a z důvodu lepšího uživatelského zážitku je žádoucí tyto zvyklosti dodržet i při vývoji aplikací hybridních. Tento problém řeší nástroj Ionic Framework.

Ionic je frontendový framework pro vývoj mobilních aplikací založený na JavaScriptovém frameworku Angular[9] (viz kapitola Angular) přinášející velké množství předem připravených komponent, které může vývojář při implementaci aplikace využít. Příkladem komponenty může

[8] Připravené části kódu, které vytvoří „můstky“ pro komunikaci mezi webovou aplikací a požadovanými nativními funkcionalitami zařízení.
[9] Nejnovější verze Ionicu přináší možnost jeho využití s libovolným JS frameworkem.

být jednoduché tlačítko, grafické karty, lišta záložek, ale i věci obecnější, jako například mřížkový systém pro rozložení komponent na stránce. [19][20]

Nejde pouze o komponenty čistě vizuální – součástí jsou také nástroje pro práci s různými typy upozornění na mobilním zařízení. Důležitá je skutečnost, že Ionic se stará nejen o to, aby komponenty vypadaly tak, jak je na nějaké platformě zvykem, ale také aby působily přirozeně (nativně) na **každé** z podporovaných platforem. Vývojáři tak stačí použít při vývoji připravenou komponentu a tento framework se postará o to, aby na každé platformě vypadala tak, jak to uživatelé dané platformy očekávají. Zároveň však vývojáři nesvazuje ruce – komponenty lze vzhledově upravovat a přizpůsobovat pomocí proměnných definovaných v Ionicu. A kdyby ani pomocí nich vývojář nedosáhl svého cíle, výsledkem je nakonec vždy webová aplikace, kterou lze modifikovat stejně, jakou jakoukoliv jinou. [19][20]

Ionic je z pohledu vývojáře postaven nad Cordovou. Další výhodou tedy je, že obaluje její pluginy pro přístup k nativním funkcím zařízení. Vývojář pracuje pouze s Ionicem, a skrz něj ovládá zároveň Cordovu (sekce frameworku zvaná Ionic Native). [19]

Ionic nabízí také příkazy pro manipulaci s projektem z příkazové řádky, například generování vzorových šablon, jednotlivých stránek, nebo spuštění aplikace v okně webového prohlížeče (sekce frameworku Ionic CLI). [19]

1.2.3 Kompilované aplikace

Posledním typem mobilních aplikací z hlediska jejich implementace jsou aplikace kompilované. Tyto aplikace, ač vyvíjeny v jiném jazyce, než je jazyk určený pro vývoj nativních aplikací dané platformy, jsou následně na nativní aplikace částečně převedeny. Cílem tohoto přístupu tedy je, podobně jako u hybridních aplikací, pomocí jedné sady technologií vyvinout aplikaci přenositelnou na více platforem. [9]

Technologie jako React Native nebo NativeScript sází, stejně jako aplikace hybridní, na vývoj v jazyce JavaScript. Další projekt spadající do této kategorie – Xamarin společnosti Microsoft – využívá místo webových technologií jazyk C#. Při vývoji v této technologii je možné buď propojit logiku aplikace napsanou v jazyce C# s nativními vizualizacemi pro dané platformy, nebo jít cestou kompletního vývoje aplikace v Xamarinu. [9][21]

Výhodou kompilovaných aplikací oproti aplikacím hybridním je vyšší výkon. Tato přednost je ještě umocněna u Xamarinu, kde je zdrojový kód kompilován, nikoliv interpretován. Vývoj kompilovaných aplikací, stejně jako hybridních, je jednodušší z důvodu jednotné sady technologií využívaných při vývoji napříč platformami. [9][21][22]

Mezi největší nevýhody patří závislost na zvolené technologii. Tento typ vývoje tedy nemusí být vhodný pro aplikace se specifickými nároky na nativní funkce mobilních zařízení. Kompilované aplikace také výkonnostně zaostávají za aplikacemi nativními. [9]

1.2.4 Porovnání typů implementací

Protože každý typ má své výhody a nevýhody, je vhodné vždy zvážit, který z postupů zvolit pro implementaci konkrétní aplikace. Následující zhodnocení těchto metod čerpá především ze zdrojů [9] a [22].

Nativní přístup bude vždy produkovat nejefektivnější a nejvýkonnější aplikace. Při vývoji je možné se opřít o rozsáhlou komunitu vývojářů a knihoven, zároveň však není třeba být na knihovnách třetích stran závislý. Nehrozí ani ukončení podpory nativního vývoje. Vývojář nativních aplikací bude mít také jako první přístup k novým funkcím mobilních zařízení. Plusem jistě je i dostupnost designových knihoven pro dosažení jednotného přirozeného vzhledu aplikací vyvíjených na danou platformu.

Největším problémem při vývoji nativních aplikací je nepřenositelnost. Pokud by aplikace měla cílit na co největší počet uživatelů roztříštěných na více mobilních platformách, je potřeba vyvinout aplikaci pro každou platformu zvlášť. Tato skutečnost může odradit jednak samotné vývojáře, kteří by se pro vývoj na různé platformy museli seznámit s větším množstvím odlišných technologií, ale i společnosti stojící o vývoj vlastní aplikace, neboť tento postup implementace je nejvíce nákladný právě z důvodu vývoje více aplikací najednou, což mnohdy zahrnuje sestavení více vývojářských týmů.

Hybridní aplikace staví na myšlence přenositelnosti aplikace a její nezávislosti na konkrétní platformě, zároveň však netrpí neduhem aplikací čistě webových, tedy závislostí dostupnosti aplikace na internetovém připojení. Z návaznosti na webové aplikace však pramení velké množství výhod. Při vývoji je možné využít rozsáhlých zdrojů a knihoven využívajících dané webové technologie. Rostoucí popularita jazyka JavaScript zajišťuje pro tyto technologie stále větší podporu a širší komunitu. V dnešní době je také trendem pro společnosti vytvářet mobilní aplikace pro svůj business podobně, jako tomu bylo dříve u aplikací webových. Spousta webových vývojářů tedy jistě uvítá snazší přechod na vývoj aplikací mobilních, protože mohou využít již nabyté znalosti a zkušenosti. S tím souvisí i výhoda znovupoužitelnosti částí zdrojového kódu z webových aplikací v aplikacích hybridních.

V čem hybridní aplikace zaostávají je využití výkonu zařízení. I když se rozdíl oproti nativním aplikacím stále zmenšuje, z důvodu interpretace JS a komunikace přes nativní mezivrstvu

budou hybridní aplikace vždy o něco pomalejší. Tento problém je nejvíce znatelný u graficky náročných aplikací, jako jsou například 3D hry. Další nevýhodou hybridního vývoje je odkázanost na technologie třetích stran. Jde jednak o podporu nativních funkcí zařízení daných platforem (tedy o dostupnost pluginů pro komunikaci s nativními prvky mobilu či tabletu), jednak také o závislost na podpoře a vývoji samotné použité technologie.

Kompilované aplikace jdou střední cestou mezi aplikacemi nativními a hybridními. Podobně jako hybridní aplikace kladou důraz na přenositelnost, jednotný vývoj a znovupoužitelnost kódu, nicméně konkrétní podpora se můžu u jednotlivých technologií lišit. Výkonnostně se díky částečné kompilaci na aplikace nativní řadí mezi předchozí dvě skupiny. Konkrétně technologie jako React Native nebo NativeScript sdílí s hybridními aplikacemi výhody využití webových technologií, tedy rozsáhlé zdroje a knihovny a snazší přechod pro webového vývojáře. Xamarin naopak vyniká vyšším výkonem díky celkové kompilaci kódu.

S hybridními aplikace sdílí i jednu z nevýhod – závislost na technologiích třetích stran. Jde především opět o pluginy pro přístup k nativním prvkům mobilních zařízení a o podporu a vývoj dané technologie.

Tabulka připojená formou přílohy A přehledně shrnuje výše popsané charakteristiky, výhody a nevýhody jednotlivých přístupů k implementaci mobilních aplikací, včetně konkrétních technologií.

Volba správného přístupu k implementaci mobilní aplikace není jednoduchou otázkou a při hledání správné odpovědi je třeba brát v potaz vícero skutečností souvisejících s implementací konkrétní aplikace.

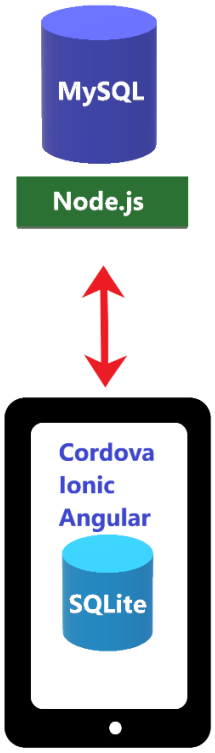
První důležitou otázkou je požadavek na výkon aplikace. Pokud by se jednalo o graficky náročnou aplikaci (grafické editory, náročnější mobilní hry), je ve většině případů nejlepší cestou vývoj nativní aplikace. Je zde ale nutné počítat se zvýšenými náklady na vývoj i údržbu, neboť bude nutné de facto vyvinout pro každou platformu zvláštní aplikaci.

Další, neméně důležitou otázkou, je cílová skupina potenciálních uživatelů aplikace, respektive jejich fragmentace napříč dostupnými platformami. S tím souvisí rozhodnutí, zda vyvíjet aplikace cílící na platformu jednotnou, nebo aplikaci multiplatformní. Pro jednodušší multiplatformní aplikace se jeví jako optimální cesta hybridní implementace, případně vývoj aplikace kompilované. Je pak nicméně nutné zvolit technologii podporující požadované nativní funkce mobilních zařízení.

Svou roli při výběru mohou sehrát i předchozí zkušenosti vývojáře aplikace. Programátor v jazyce Java může snadno přejít na vývoj nativních mobilních aplikací pro platformu Android. Webový vývojáři naopak jistě zužitkují své předchozí znalosti při přechodu na implementaci hybridních nebo kompilovaných mobilních aplikací. Konkrétně vývojář využívající framework Angular nebude mít problém s přechodem na Ionic, programátor se zkušenostmi s oblíbeným webovým frameworkem React zase snáze přejde na vývoj v React Native. Developeři programující v C# mohou navázat na své předchozí znalosti při vývoji v Xamarinu.

2 VÝVOJ MOBILNÍ APLIKACE PRO PODPORU PRODEJE

V rámci praktické části této práce byla vyvinuta mobilní aplikace usnadňující prodejcům obchodní společnosti nabízení produktů této společnosti, tvorbu objednávek a jejich následné odeslání ke zpracování. Pracovní název aplikace je *eKatalog*. Postup návrhu a implementace této aplikace bude popsán v kapitole *3 Implementace aplikace*, samotná aplikace pak v kapitole *4 Popis funkcionalit aplikace*. Před začátkem jejího vývoje bylo však třeba rozhodnout o postupu její implementace.



Obrázek 3 – přehled částí projektu a vybraných technologií

Cílem aplikace není provádět graficky náročné výpočty, není tedy třeba nutně postupovat cestou nejvyššího výkonu. Protože aplikace necílí na konkrétního zákazníka, bylo by vhodné vyvinout multiplatformní aplikaci, která by potenciálním zákazníkům umožnila větší flexibilitu při výběru zařízení, na kterých budou aplikaci využívat. Aplikace by také měla splňovat dříve zmíněnou podmínku zachování jednotného vzhledu napříč různými aplikacemi jedné platformy. Na základě těchto požadavků, a s přihlédnutím k předchozím zkušenostem vývojáře, byl zvolen postup implementace cestou hybridní aplikace s využitím technologií Ionic a Cordova. Protože tyto technologie byly dostatečně uvedeny v předchozí části dokumentu, následuje představení dalších vybraných technologií a nástrojů použitých při vývoji této aplikace. Popis je rozdělen na technologie *frontendu* (část mobilní aplikace) a *backendu* (část

vzdáleného serveru, se kterým aplikace komunikuje). Toto rozdělení ilustruje také obrázek 3 zobrazující jak samotné dělení projektu, tak i hlavní technologie jednotlivých částí.

2.1 Použité technologie – frontend

Jak bylo naznačeno výše, frontendovovou částí práce je myšlena samotná mobilní aplikace. Vývoj hybridní aplikace v Ionicu obnáší práci s tradičními webovými technologiemi, z důvodu uchování dat pro přístup bez připojení k internetu bylo také nutné využít možností databázového uložiště.

2.1.1 Webové technologie

HTML 5

Zkratka HTML[10] označuje hypertextový značkovací jazyk velmi hojně využívaný k tvorbě webového obsahu. Pomocí něj lze popsat strukturu webového dokumentu, včetně významu jednotlivých elementů, na které je dokument rozdělen. S příchodem jeho poslední verze HTML 5 se z něj stalo plnohodnotné rozhraní pro vývoj komplexních webových aplikací. Došlo k vylepšení elementů starších verzí (například vstupní pole *input*) i přidání elementů zcela nových (jako jsou elementy *audio* nebo *video* pro práci s multimédii). HTML 5 rovněž usnadňuje validaci zadávaných údajů na straně klienta bez nutnosti použít další technologie, jako například jazyk JavaScript. [23]

Framework Ionic doplňuje tradiční HTML tagy o skupinu vlastních předpřipravených elementů. Tyto lze využít obdobně jako ty tradiční a obě skupiny kombinovat. Elementy Ionicu typicky začínají předponou *ion* a popisují nějaký strukturní prvek mobilní aplikace. Příkladem může být element *ion-list* označující skupinu prvků *ion-item*, které mají být zobrazeny dohromady jako jeden seznam. Ionic se při vykreslování tohoto elementu postará o zachování designových zvyklostí jednotlivých platforem. [19]

CSS a Sass

CSS[11], neboli Kaskádové styly, slouží v kontextu webových technologií k definici a úpravě vzhledové stránky aplikace. Posledním standardem je CSS 3. Syntaxe zápisu kaskádových stylů se dělí na tři části:

- *selektor* sloužící k výběru elementů webové stránky, na které bude daný styl aplikován,
- *vlastnost* udávající parametr daného elementu, který bude stylem ovlivněn a
- *hodnota* popisující, jak bude daná vlastnost nastavena.

[10] HyperText Markup Language.
[11] Cascading Style Sheets.

Tyto tři části jsou zapisovány do následující formule:

```
selektor {
    vlastnost: hodnota;
    …
}
```

Selektorem může být například obecný typ HTML elementu (tag), třída přiřazená vybraným elementům, nebo identifikátor selektující konkrétní element. Selektory lze kombinovat, řetězit a spojovat. Pokud jeden element stránky vyhovuje více selektorům, které definují stejné vlastnosti různých hodnot, rozhodne o jejich použití specifičnost daných stylů. [24]

Vlastností elementů upravitelných kaskádovými styly existuje velké množství. Nastavit lze poziční elementy jako odsazení, umístění, rozměry, dále například okraje, zaoblení, různé barvy, stíny a spoustu dalších vlastností. Pomocí změny stylů lze přechody také animovat. [24]

Jednoduchá syntaxe kaskádových stylů postrádá některé funkcionality, které by bylo možné pro tvorbu stylů využít. Chybí například práce s proměnnými nebo možnost zanoření bloků se styly. Tyto vlastnosti do určité míry doplňují takzvané CSS preprocesory, které zajistí následný překlad kódu do klasických kaskádových stylů. Mezi hojně využívané preprocesory patří například Less[12] nebo Sass[13]. Framework Ionic přímo pracuje s druhým jmenovaným, soubory s kaskádovými styly zde proto mají příponu *scss*. [25]

JavaScript, TypeScript

Aby webové aplikace poskytovaly lepší uživatelský zážitek, měly by být dynamické. K tvorbě takových webových aplikací se s velkou oblibou používá programovací jazyk JavaScript, o jehož oblibě svědčí mimo jiné žebříček populárních jazyků využívaných v rámci verzovacího nástroje GitHub, kde s velkým náskokem obsadil první místo [26]. K jeho popularitě přispívá také velká dostupnost – díky jeho integraci v každém moderním prohlížeči je JavaScript dostupný na téměř všech desktopových i mobilních zařízeních.

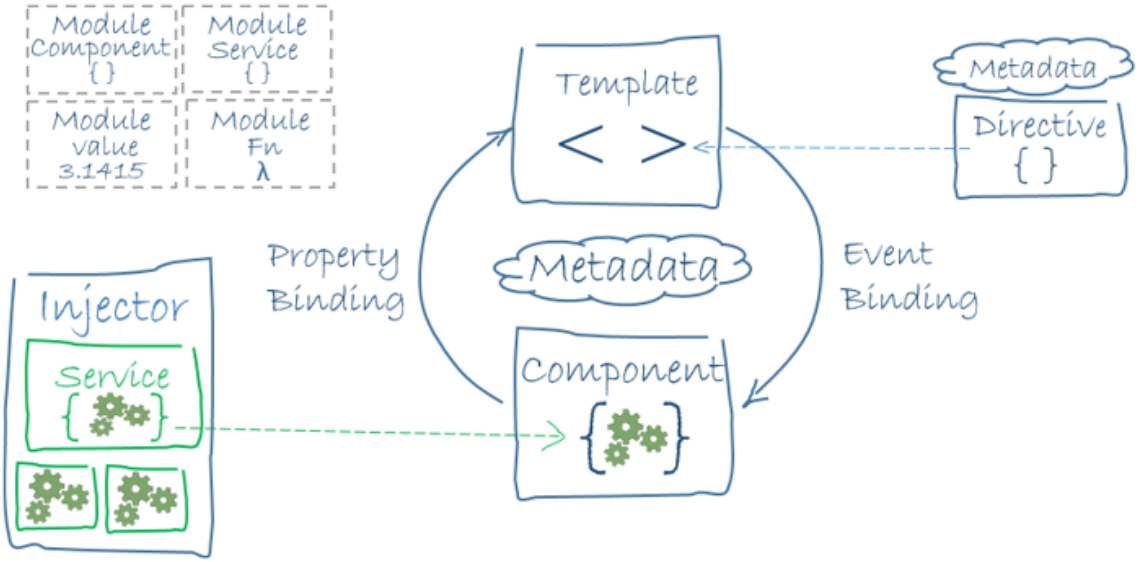
Tento jazyk je při tvorbě webových aplikací využíván k implementaci logických požadavků na aplikaci, dynamických změn či úpravě struktury nebo vzhledu stránky. Pracuje při tom se strukturou elementů nazývanou DOM[14]. [27]

[12] Leaner Style Sheets.
[13] Syntactically Awesome Style Sheets.
[14] Document Object Model.

Nadstavbou nad JS je jazyk TypeScript. Jde o programovací jazyk od firmy Microsoft, který je supersetem[15] JS. Přináší navíc především statickou typovou kontrolu kódu. Ta pomáhá mimo jiné odhalit některé programátorské chyby v kódu ještě před jeho spuštěním. Oproti starší verzi JS, do které je typescriptový kód před jeho vykonání překládán, přináší také možnost využití tříd, rozhraní nebo zkrácené zápisy některých méně přehledných struktur kódu. [28]

Angular

Pro JavaScript existuje celá řada užitečných knihoven a frameworků rozšiřujících jej o užitečné funkcionality. Jedním z nich je framework Angular vyvíjený společností Google. Ten implementuje strukturu knihoven usnadňujících tvorbu webových aplikací, které lze jednoduše importovat do vyvíjeného projektu. Angular lze používat také v kombinaci s TypeScriptem[16]. Pracuje se strukturou modulů[17], které tvoří základní stavební kameny webové aplikace využívající tento framework. [29]



Obrázek 4 – principy JS frameworku Angular, zdroj [29]

Modul obsahuje kód související s určitou komponentou aplikace. Komponenta v sobě seskupuje logicky související části kódu, které jsou díky tomu uchovány na jednom místě v kódu a jsou znovupoužitelné. Definuje určitý pohled aplikace (*view*). Dalšími stavebními prvky jsou služby (*services*) zapouzdřující a zpřístupňující danou funkcionalitu aplikace, ne nutně spojenou s konkrétním pohledem. Služby mohou být do komponent vloženy formou

[15] Obsahuje veškerou syntaxi JavaScriptu plus něco navíc. Každý validní kód v JavaScriptu je tedy zároveň validním TypeScriptem.
[16] V novějších verzích frameworku je toto preferovaná varianta.
[17] Označovaných jako *NgModules*.

závislostí (princip *dependency injection*). Jak služby, tak komponenty jsou z pohledu TypeScriptu reprezentovány třídami. [29]

Dalším nosným principem Angularu je jeho práce s HTML šablonami (*templates*), které lze dynamicky měnit jejich svázáním s proměnnými daty v TS (princip *data binding*). Tyto a další principy jsou znázorněny na obrázku 4. [29]

Jak bylo již zmíněno dříve, Ionic je postaven právě na Angularu a je tedy možné při vývoji mobilní aplikace touto cestou tento webový Framework využívat. Stejně jako v Angularu, i v Ionicu existují moduly, komponenty, služby, princip *dependency injection* nebo *data binding*. K vývoji hybridních mobilních aplikací pomocí Ionicu je možné využít jazyk TypeScript, soubory uchovávající logické stránky jednotlivých komponent proto mají příponu *ts*.

2.1.2 Databázové technologie

Jak bylo popsáno v předchozích částech, hybridní mobilní aplikace sdílejí některé výhody a nevýhody aplikací webových. Další ze společných vlastností těchto dvou skupin je možnost přístupu k lokálnímu uložišti dostupného pomocí HTML 5. Toto uložiště však má své limity. Největším z nich je omezení maximální velikosti uložiště na 10 MB dat. Protože objem dat ukládaných v rámci projektované aplikace by mohl tento limit překročit, bylo třeba nalézt jiné řešení problému uchování dat aplikace bez přístupu k internetu. [30]

SQLite

SQLite je vestavěná databáze typu SQL pracující přímo se soubory na lokálním disku. Jde o jednu z nejrozšířenějších databází, která je dostupná mimo jiné ve všech zařízeních s operačními systémy Windows 10, MAC OS X, Android nebo iOS. Právě poslední dva jmenované jsou cílovými platformami projektované aplikace, což z SQLite činí ideálního kandidáta na její využití pro účely ukládání dat aplikace. Jen pro srovnání – maximální velikost databáze SQLite je 140 TB. Ionic (respektive Cordova) nabízí plugin pro přístup k SQLite databázi dostupné na daném zařízení a třídy pro snadnou manipulaci s ní. [31]

2.2 Použité technologie – backend

Protože projektovaná mobilní aplikace musí odněkud zdrojová data stahovat, a naopak mít kam ukládat vytvořené objednávky, bylo třeba implementovat zároveň vzdálený databázový server právě pro tyto účely. Backend aplikace tedy tvoří samotná vzdálená databáze, a dále internetový

server zprostředkující komunikaci mezi mobilní aplikací a touto databází. Tato část práce využívá níže popsané technologie.

2.2.1 Serverové technologie

Node.js

Node.js je ve zkratce platforma umožňující použití jazyka JavaScript na straně internetového serveru. Kromě jednotného jazyka vývoje webových aplikací (v obou částech práce je používán právě JS) přináší i řadu dalších výhod, jako je možnost sdílení knihoven, neblokující architektura a v neposlední řadě také rychlost. Předností je i malá velikost a nenáročnost použití této technologie. [32]

2.2.2 Databázové technologie

MySQL

MySQL je oblíbená relační databáze často využívaná v kombinaci s webovými aplikacemi. Mezi její výhody patří rychlost, spolehlivost a nenáročnost použití. Je využívána také velkými společnostmi jako jsou Facebook, Twitter nebo Youtube. Databáze je v současná době spravována společností Oracle. Podobně jako je tomu u dalších použitých technologií, je i MySQL dostupná v režimu open source (licence GPL). [33]

2.3 Technologie pro komunikaci v rámci aplikace

Z důvodu plánovaného přenosu většího množství dat mezi mobilní a serverovou částí projektu bylo třeba tato data nějakým způsobem při přenosu a jejich uchování organizovat.

JSON

Při komunikaci mezi webovou aplikací a serverem mohou být data přenášena pouze v textové podobě. Ta ale neumožňuje data jednoduše formátovat a organizovat. Jedním z možných řešení tohoto problému je využití formátu JSON[18]. Jedná se o nástroj schopný převést objekt jazyka JavaScript na jeho textovou reprezentaci podobné syntaxe. Objekt v textové podobě může být bez problému přenesen internetovou sítí na druhou stranu komunikačního kanálu (konkrétně tedy z mobilní aplikace na stranu serveru nebo obráceně), kde je pomocí nástroje JSON znovu sestaven do podoby původního JS objektu. [34]

Protože je tento formát s JS objekty úzce spjat, syntaxe jeho zápisu tvoří podmnožinu syntaxe těchto objektů. Data jsou reprezentována formou párových dvojic klíč – hodnota, které jsou

[18] JavaScript Object Notation.

oddělený čárkou. Klíčem může být pouze textový řetězec uzavřený ve dvojitých uvozovkách, hodnotu může představovat textový řetězec, číslo, pravdivostní hodnota, vnořený JSON objekt, speciální typ *null* nebo pole hodnot předchozích typů [34]. Formát využívá také dvou typů závorek. Složené závorky ({}) slouží k uchování objektů, hranaté závorky ([]) pak označují pole. Objekty i pole lze zanořovat do sebe. Příklad JSON objektu by mohl vypadat následovně:

```
{
    “jméno”: “Karel”,
    “příjmení”: “Novák”,
    “velikostBot”: 45,
    “děti”: [
        {
            “jméno”: “Anežka”,
            “věk”: 11
        },
        {
            “jméno”: “Pepík”,
            “věk”: 9
        }
    ],
    “ženatý”: true
}
```

Převod mezi JS objektem a jeho JSON reprezentací je velmi jednoduchý. Pro převedení objektu na JSON text stačí zavolat statickou metodu *JSON.stringify*, která převede objekt obdržený formou parametru na text. Při opětovném sestavování objektu lze využít další statickou metodu – *JSON.parse*, která z textu vytvoří JS objekt. Je však důležité poznamenat, že si při této transformaci objekt neuchová své dostupné funkce, neboť s tímto typem proměnných formát JSON pracovat neumí. [34]

Alternativou k využití nástroje JSON je formát XML, který je ale v porovnání s JSONem složitější z pohledu zápisu, čtení i převodu, a jeho výhody nejsou v kontextu projektované aplikace velkým přínosem.

2.4 Prostředky vývoje

Při vývoji aplikace bylo použito několik nástrojů usnadňujících nebo zpřehledňujících tento vývoj, z nichž vybrané jsou popsány v této části práce.

GitHub

Při vývoji (nejen) webových a mobilních aplikací může dojít k poškození nebo dokonce ztrátě zdrojového kódu aplikace nebo potřebných dat. Z tohoto důvodu je žádoucí při vývoji zdrojové

kódy jednak zálohovat, ale také uchovávat jednotlivé verze vznikající v průběhu vývoje. Tím lze jak předejít ztrátě dat, tak i získat možnost navrácení se k funkční verzi aplikace v případě poškození verze právě vyvíjené. Další výhodou použití verzovacího systému je možnost sdílet zdrojové kódy vyvíjené aplikace napříč jejími vývojáři. Jednou ze služeb umožňující jednoduchou správu a uchování verzí zdrojového kódu je GitHub.

Tato služba je postavena na rozšířeném verzovacím systému Git, který pracuje s verzemi jako se snímky zdrojových kódu v určitých časech. Při vytvoření dalšího snímku (verze) ukládá pouze ty zdrojové soubory, které byly nějakým způsobem změněny. Je také možné vývoj projektu v některé jeho fázi rozdělit, pracovat na více verzích současně, a ty poté znovu spojit do jedné společné. Pro práci se samotným nástrojem Git lze využít prostředí příkazového řádku. [35][36]

GitHub umožňuje správu Gitu pomocí webové aplikace dostupné na adrese *github.com*. V tomto prostředí mohou registrovaní uživatelé (vývojáři) vytvářet, sdílet a upravovat veřejné i soukromé repositáře[19], vytvářet vlastní kopie cizích repositářů a přispívat ostatním vývojářům do jejich projektů.

Mimo tyto základní vlastnosti související se správou a verzováním projektu poskytuje GitHub další služby, jako například některé statistiky konkrétních projektů a uživatelů. Díky napojení repositářů na uživatelský profil lze tento profil také využít jako jistou formu resumé vývojáře.

VS Code

V každém projektu, jehož cílem je vyvinout a upravovat zdrojové kódy, je potřeba využít nějaký editor tohoto kódu. Může jít o program tak jednoduchý, jako obyčejný textový editor, nebo třeba o rozsáhlé vývojové prostředí nazývané též IDE[20]. Volba editoru kódu závisí na jazyku vývoje, konkrétní aplikaci, ale i preferenci samotného vývojáře. Zvolením vhodného vývojového prostředí lze značně usnadnit a urychlit celý proces vývoje.

S ohledem mimo jiné na typ projektované aplikace, vývoj v jazyce TypeScript a předchozí zkušenosti vývojáře byl pro vývoj této aplikace zvolen editor kódu VS Code (celým názvem Visual Studio Code), který je populární zejména mezi programátory webových aplikací. Tento editor přichází s vestavěnou podporou pro vývoj v JavaScriptu, TypeScriptu i Node.js, což jsou jazyky použité v rámci této práce. [37]

[19] Repositáře jsou skupiny logicky souvisejících zdrojových kódů, jako jsou například jednotlivé projekty.
[20] Integrated Development Environment.

VS Code pochází, stejně jako jazyk TypeScript, z dílen společnosti Microsoft. Tato společnost stojí také za dalším IDE – nástrojem Visual Studio, který je oblíbený především mezi vývojáři pracujícími s platformou .NET. S tímto komplexním prostředím má však VS Code společné pouze některé vnitřní technologie. Code je dle slov jeho vývojářů *„odlehčený, avšak mocný editor zdrojových kódů, který je dostupný na desktopových operačních systémech Windows, macOS a Linux“*[21] [37]. Jednou z jeho vlastností je vyváženost mezi jednoduchostí použití a poskytnutím dostatečného množství prostředků usnadňujících vývoj aplikací. Obsahuje spoustu užitečných funkcionalit, z nichž některé byly využity při tvorbě projektované mobilní aplikace.

V editoru jsou integrovány nástroje jako konzole pro výpis hlášek kódu, vlastní příkazový řádek nebo debugger, s jehož pomocí lze aplikaci procházet krok po kroku. Zahrnut je také modul pro práci s Gitem, který umožňuje připojení lokální složky projektu ke vzdálenému repositáři, správu verzování přímo v editoru kódu a usnadňuje kontrolu změn provedených v jednotlivých souborech. Nechybí ani paleta s přednastavenými spustitelnými příkazy, které lze rozšířit o příkazy vlastní. [37]

Užitečná je také implementace nápovědy při psaní kódu zvané *IntelliSense*. Tento termín v sobě skrývá kombinaci automatického doplňování kódu a zobrazení informací o využívaných objektech a metodách, jako jsou jejich vlastnosti nebo parametry. Neméně prospěšná je také možnost „nakouknout“ na místo v kódu, kde jsou využívané funkce definovány (*peek definition*), díky které lze nalézt požadované informace bez nutnosti přepínat mezi zdrojovými soubory. Při úpravě kódu lze dále využít možnosti snadného přesunu a duplikace řádků nebo zápisu na více míst v kódu zároveň. Samozřejmostí je i možnost vyhledání a nahrazení textu napříč zdrojovými soubory. [37]

Code si zakládá na podpoře vývoje webových aplikací, jeho instalace i případné úpravy nastavení jsou rychlé a intuitivní. Podporována je široká škála nastavení upravujících editor na míru potřebám vývojáře. Jednoduše lze vytvořit vlastní zkratky pro často používané části kódu nebo upravitelné šablony (*snipety*), které Code po zapsání zkratky generuje. Snadná je rovněž změna vzhledu celého prostředí nebo barevných kombinací zvýrazňujících sémanticky odlišné části kódu. [37]

[21] Z anglického originálu „*Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux“.*

Pokud by vývojář přeci jen postrádal v základní verzi editoru nějakou funkcionalitu, k dispozici je také rozsáhlý market obsahující velké množství rozšíření, které lze do prostředí jednoduše integrovat. Touto cestou lze přidat podporu pro další programovací jazyky, personalizovat vzhled editoru, usnadnit práci s Gitem a dalšími nástroji nebo třeba zavést přísnější kontrolu zdrojového kódu. Stejně jednoduše je možné rozšíření opět odebrat nebo pozastavit. S výběrem správného rozšíření pro požadovanou funkcionalitu pomůže nejen jeho popis a návod obvykle dostupný v marketu, ale rovněž informace o počtu jeho stažení a hodnocení uživateli. [37]

Úprava kódu

Vedlejším cílem práce bylo udržet zdrojový kód v rámci možností přehledný, a to bez ohledu na rozsah práce. S ohledem na úroveň práce bylo také žádoucí produkovat kód efektivní, řídící se doporučenými praktikami programování v daných jazycích.

Přehlednost a čitelnost kódu zvyšuje jednotná úprava kódu napříč zdrojovými soubory. Ačkoliv editor VS Code sám o sobě obsahuje nástroje podporující jednotnou úpravu, bylo vhodné je doplnit nástrojem Prettier. Ten umí formátovat jak zdrojové kódy v TypeScriptu a JavaScriptu, tak i soubory preprocesoru kaskádových stylů Sass nebo objekty typu JSON. Je také možné upravit jednotlivá pravidla formátování podle preferencí vývojáře. Výsledkem je nicméně vždy kód upravený stejně napříč strukturou zdrojových souborů. [38]

Pro zvýšení úrovně kontroly správnosti a kvality produkovaného kódu byly využity *lintery*. Principem linteru je analýza kódu a odhalení potenciálních chyb ještě před jeho spuštěním. Linterů existuje celá řada a obvykle se specializují na konkrétní jazyky. Pro JavaScript existuje například ESLint nebo JSLint, pro TypeScript pak TSLint. Nástroje obsahují celou řadu parametrů, kterými lze upravit vlastnosti prováděné kontroly. Ty lze rozdělit na pravidla formátovací, která plní podobný účel jako nástroj Prettier, a pravidla pro zkvalitnění produkovaného kódu, která mají vývojáře upozornit na neoptimální využití některých nástrojů jazyka. Kromě vlastního nastavení lze využít i předpřipravené sady doporučených pravidel. [38][39]

Jak Prettier, tak TSLint existují mimo jiné ve formě rozšíření pro editor VS Code. To značně usnadnilo jejich zavedení a nastavení v rámci projektu.

MySQL Workbench

Pro správu vzdálené databáze MySQL byl využit nástroj MySQL Workbench. Toto vizuální prostředí nabízí velkou škálu funkcionalit, například vizualizaci dat, grafický návrh databáze,

editor SQL nebo podporu migrace databáze. Nástroj je dostupný na všech nejrozšířenějších desktopových platformách, tedy Windows, Linux i MAC OS X. [40]

3 IMPLEMENTACE APLIKACE

Jak již bylo zmíněno, cílem projektované aplikace je usnadnit cestujícím obchodníkům prodej jejich produktů[22]. Základním využitím aplikace je tedy tvorba objednávek a jejich následné odeslání ke zpracování. Pro usnadnění této činnosti bylo třeba zohlednit postup prodejců při nabízení jejich produktů zákazníkům.

Prodejce potřebuje mít v době tvorby objednávky dostupné jednak údaje o nabízených produktech, ale také informace týkající se konkrétních zákazníků. Dále lze předpokládat výskyt lidských chyb při tvorbě objednávky, je tedy nutné umožnit úpravu již zadaných informací, a to jak na úrovni jednotlivých položek, tak i na úrovni celé objednávky. Protože chybovat mohou nejen lidé, je třeba očekávat také možné reklamace již prodaných produktů a implementovat prostředky pro podporu tohoto procesu. Zákazník (nebo samotný prodejce) by také mohl požadovat kontrolu objednávky po jejím odeslání, proto je nutné umožnit zobrazení již odeslaných objednávek uložených na straně vzdáleného serveru.

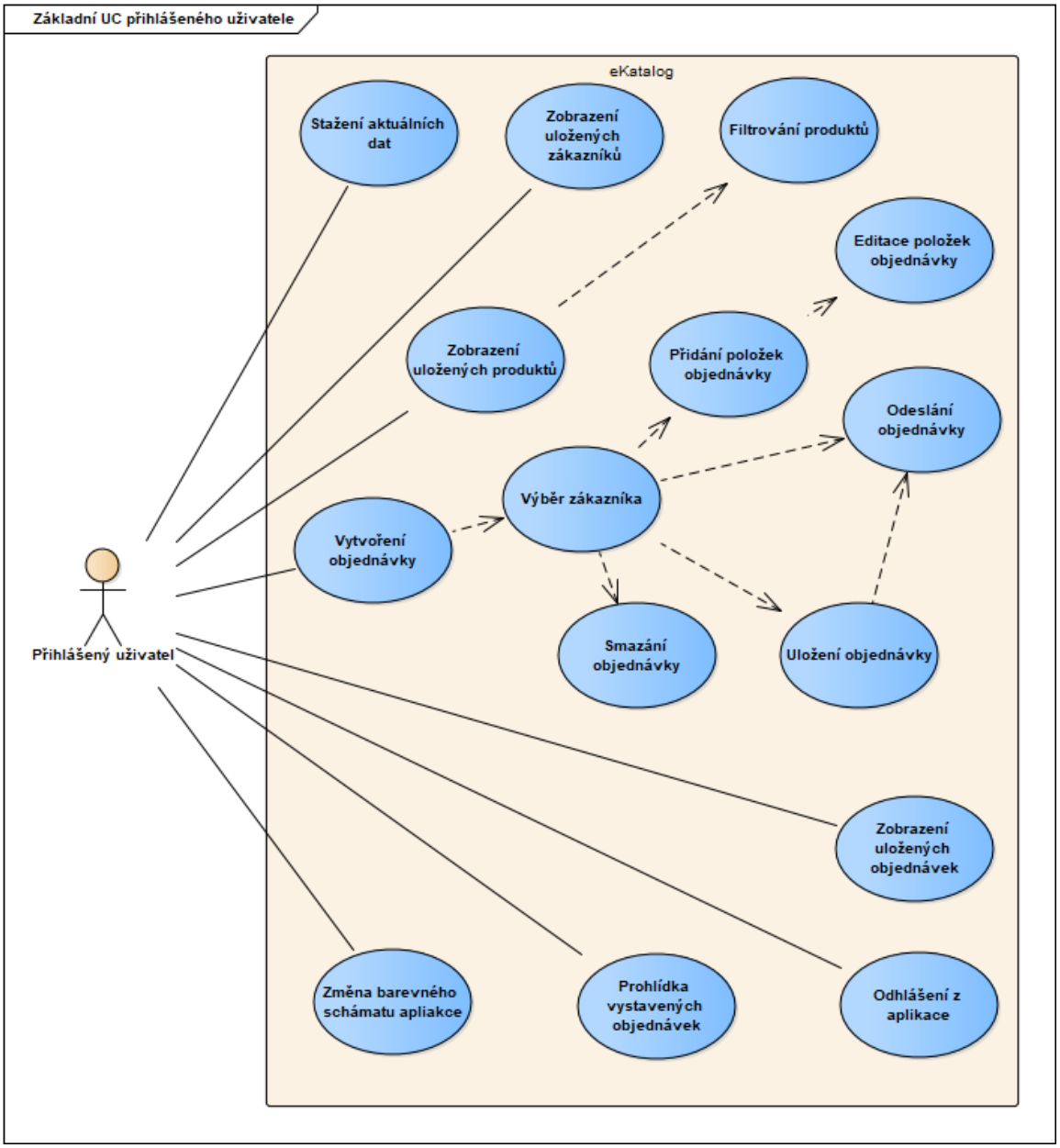
Z bezpečnostního hlediska je důležité zachovat důvěrnost stažených informací společnosti, a proto povolit přístup pouze oprávněným osobám. Nevhodné by bylo rovněž uchování citlivých informací uživatele v jejich čitelné podobě.

V následujících podkapitolách bude popsán postup vývoje projektované aplikace od definice prvotních požadavků, až po nasazení aplikace na mobilní zařízení a její testování. Celkový proces návrhu aplikace byl inspirován metodikou unifikovaný proces vývoje aplikací (UP) podle výkladu zdroje [41], avšak samotná implementace byla realizována spíše formou inkrementačního (přírůstkového) přístupu.

3.1 Požadavky, případy užití

Na začátku návrhu bylo třeba definovat konkrétní požadavky, které by měla projektovaná aplikace splňovat. Ty lze rozdělit na požadavky funkční, popisující chování projektované aplikace, a požadavky nefunkční, které specifikují obecné vlastnosti dané aplikace. Požadavky byly následně zapracovány do případů užití, jejichž ukázka vytvořená v programu Enterprise Architect je na obrázku 5.

[22] Svým způsobem tedy aplikace řeší „problém obchodního cestujícího“, i když se jedná o zcela odlišný problém než ten, jehož řešení je předmětem některých optimalizačních algoritmů.



Obrázek 5 – diagram vybraných případů užití

3.1.1 Funkční požadavky na aplikaci

- eKatalog bude zpřístupňovat veškeré funkcionality (kromě přihlášení) pouze po ověření identity uživatele.
- eKatalog bude umožňovat přihlášení do aplikace pomocí ověření zadaných údajů na straně serveru.
- eKatalog bude uchovávat údaje posledního přihlášeného uživatele a po opětovném spuštění aplikace jej znovu přihlásí. Citlivé údaje budou uchovány v šifrované podobě.
- eKatalog bude po odhlášení odstraňovat uložené uživatelské údaje.

- eKatalog bude získávat údaje o produktech, zákaznících a fakturách ze vzdáleného databázového uložiště.
- eKatalog bude získané údaje o produktech a zákaznících uchovávat v lokálním uložišti tak, aby tyto byly dostupné bez nutnosti připojení k internetové síti.
- eKatalog bude umožňovat přehledné zobrazení uložených údajů.
- eKatalog bude při připojení k internetu schopen zobrazit vytvořené objednávky uložené na straně vzdálené databáze.
- eKatalog bude dostupný ve dvou barevných provedeních (světlé a tmavé), mezi kterými bude možné za běhu aplikace přepínat.
- eKatalog bude umožňovat tvorbu objednávek formou výběru zákazníka a následným přidáváním produktů z nabídky katalogu.
- eKatalog bude umožňovat změnu zákazníka objednávky i editaci přidaných položek.
- eKatalog bude podporovat prodej i reklamaci produktů.
- eKatalog bude při prodeji produktů kontrolovat základní omezení, jako například prodej nezáporného počtu kusů.
- eKatalog bude při reklamaci poskytovat prostředky pro přiložení fotodokumentace k reklamované položce.
- eKatalog bude umožňovat vyhledávání v katalogu produkty i mezi zákazníky. Produkty bude dále možné filtrovat podle připravených skupin nebo jejich kategorie a zobrazovat v různých pořadích.
- eKatalog bude pro vytvořené objednávky poskytovat možnosti odeslání na server, uložení do lokálního uložiště nebo smazání objednávky.
- eKatalog bude objednávky, které nebylo možné odeslat na vzdálený server, ukládat do lokálního uložiště, odkud je bude možné případně znovu odeslat nebo odstranit.
- eKatalog bude při tvorbě objednávky využívat data související s preferencemi daného zákazníka, pro kterého je objednávka vytvářena.

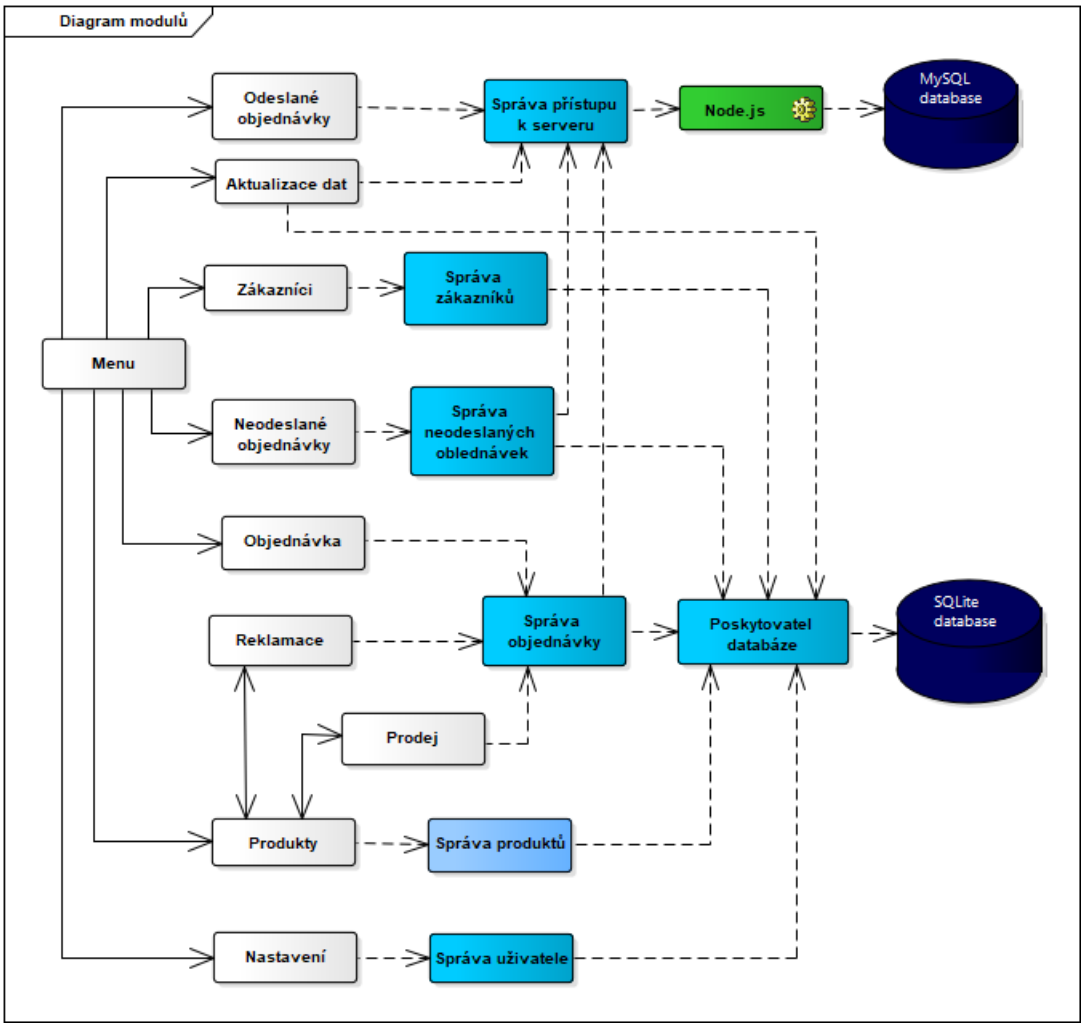
3.1.2 Nefunkční požadavky na aplikaci

- eKatalog bude realizován formou hybridní mobilní aplikace s využitím nástroje Ionic.
- eKatalog bude podporovat nejrozšířenější mobilní platformy, jmenovitě Android a iOS.

- eKatalog bude zároveň implementovat serverovou stranu projektu simulující do jisté míry reálný systém.

3.2 Návrh aplikace

Dalším krokem při vývoji aplikace byl návrh jejích modulů. Tyto moduly jsou na obrázku 6 znázorněny jednotlivými obdélníky. V kontextu eKatalogu mohou moduly reprezentovat určitý nástroj využívaný napříč aplikací, třídu poskytující svou vnitřní službu, datový model nebo komponentu (konkrétní stránku aplikace nebo její část). Následující obrázek je z důvodu zachování přehlednosti zjednodušen, takže se v něm některé skupiny modulů vůbec nevyskytují, i když jsou ve výsledné aplikaci implementovány.



Obrázek 6 – diagram vybraných modulů a jejich vazeb

Nástrojem jsou myšleny například polymorfní stránkovače uchovávající pole zobrazovaných objektů daného typu. Implementován je stránkovač jednoduchý (uchovávající jedinou stránku) a stránkovač komplexní, který kromě aktuální stránky aktualizuje rovněž stránku předchozí a následující. Druhý jmenovaný je využit pro zobrazení zákazníků a katalogu produktů, kde

jsou implementovány dynamické přechody mezi jednotlivými stránkami, a je tedy žádoucí z důvodu rychlejší odezvy systému načítat zobrazovaná data předem. Dalšími nástroji jsou pak filtry využívané pro selektivní výběr objektů nebo validátory ověřující správnost vstupů zadávaných uživatelem do formulářů.

Třídy poskytující službu (*providers, services*) obalují konkrétní vnitřní funkcionalitu aplikace. Tyto třídy pak mohou být pomocí principu *dependency injection* zpřístupněny pro každou z komponent, která službu využívá, nebo vloženy přímo do inicializačního modulu aplikace. V druhém případě je vytvořena pouze jediná instance této třídy, kterou pak mohou využívat všechny komponenty aplikace, které požádají o její referenci. Takto využívaní poskytovatelé služeb plní zároveň roli implementace návrhového vzoru *singleton*[23].

Poskytovatelů je v eKatalogu hned několik a na předchozím obrázku 6 jsou zvýrazněny modrou výplní. Kromě služeb zapouzdřujících stránkovače, filtry a další potřebné proměnné týkající se konkrétních katalogů (zákazníků, produktů, neodeslaných a odeslaných objednávek) je dostupná také služba spravující aktuálně rozpracovanou objednávku nebo služba zpřístupňující informace o přihlášeném uživateli.

Dobrým příkladem využití služby jako singletonu jsou pak služby spravující přístup k lokální databázi nebo ke vzdálenému serveru. Pokud kterákoliv komponenta aplikace chce pracovat s SQLite databází daného mobilního zařízení, může tak učinit pouze s využitím poskytovatele služeb lokální databáze, který spravuje jedinou instanci této databáze v rámci celé aplikace.

Dalším z návrhových vzorů využitých při implementaci aplikace je MVC (*Model – View – Controller*). Tento návrhový vzor odděluje třídy reprezentující datovou strukturu (model) od prezentace těchto dat (pohled)[24] a funkční logiky celou komponentu spravující (ovladač)[25].

V eKatalogu jsou modely uchovány ve zvláštní složce. Nachází se zde jednak třídy využívané při deserializaci[26] dat z databáze, a jednak třídy použité při práci s dynamickými daty objednávek.

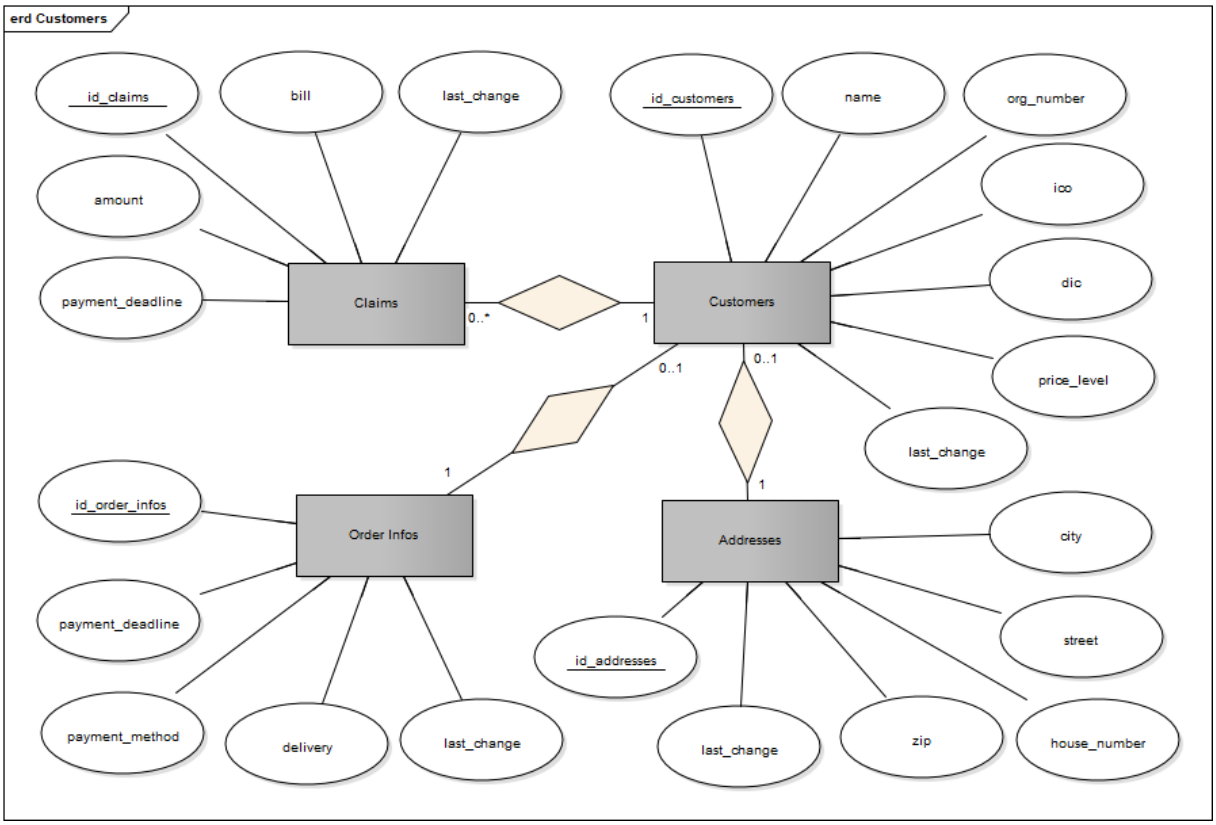
Další dvě části MVC, tedy pohledy a ovladače, jsou v aplikacích využívajících nástroj Ionic uchovávány společně ve struktuře jednotlivých komponent aplikace (viz. kapitola 3.3). Ovladače jsou reprezentovány třídami implementujícími funkční logiku dané komponenty.

[23] Singleton zajišťuje jednotný přístupový bod k funkcionalitě, kterou obaluje.
[24] View.
[25] Controller.
[26] Proces deserializace je převod uložených (serializovaných) dat na objekty programovacího jazyka, se kterými lze dále dynamicky pracovat.

Každému ovladači je pak přiřazena šablona popisující strukturu jeho pohledu (s využitím jazyka HTML obohaceného o vlastní elementy Ionicu). Vzhledová stránka pohledu je pro větší přehlednost implementována v samostatném souboru (s využitím CSS preprocesoru Sass). Komponenty jsou na obrázku 6 znázorněny bílou barvou.

Souběžně s návrhem mobilní aplikace byl upravován také koncept serverové části sloužící k simulaci centrálního uložiště obchodní společnosti, se kterým by tato aplikace komunikovala. Protože hlavním účelem serverové části bylo demonstrovat jednotlivé funkcionality mobilní aplikace bylo možné zvolit vysokou míru abstrakce skutečného informačního systému, který by byl v dané společnosti implementován.

Hlavním segmentem serverové části je MySQL databáze simulující zmíněné uložiště dat obchodní společnosti. Zde jsou uchovány jednak katalogy produktů a zákazníků nebo objednávky, ale také informace konkrétních uživatelů aplikace – zaměstnanců společnosti. K této databázi by v reálném případě přistupovali také další informační systémy, například účetní systém, správce uživatelských účtů nebo systém zpracování objednávek. Ukázka struktury entit vytvořených pro uchování katalogu zákazníků společnosti je zobrazena na obrázku 7.



Obrázek 7 – diagram entit pro uchování katalogu zákazníků v databázi

Pro přístup k této databázi mobilní aplikace využívá webovou službu implementovanou pomocí technologie Node.js. Tato služba slouží jako prostředník pro vzdálené volání procedur databáze, kterým předává zadané parametry. Databáze vstupy ověří, dotaz zpracuje a výsledek je opět prostřednictvím této služby odeslán zpět do mobilní aplikace.

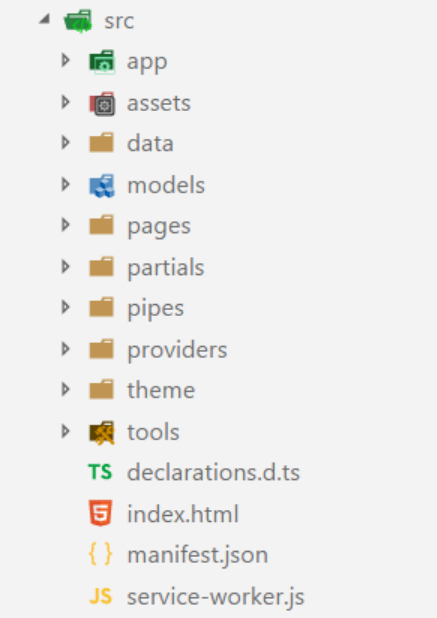
Před přeposláním parametrů a vyvoláním požadované procedury webová služba nejprve ověří v databázi autentizační údaje zadané v hlavičce dotazu. Pokud autentizace skončí neúspěchem je odesílatel o jejím selhání informován. Tímto je databáze chráněna proti potenciálním dotazům neoprávněných uživatelů.

3.3 Struktura zdrojového kódu a popis modulů

Na začátku samotné implementace navržené aplikace bylo třeba vytvořit projekt v nástroji Ionic. K tomu slouží příkaz Ionic CLI[27]

```
ionic start eKatalog sidemenu --v2,
```

kde první parametr označuje název vytvářeného projektu a druhý parametr šablonu, podle které bude projekt vygenerován. Zvolená šablona *sidemenu* nejvíce odpovídá navrhované struktuře aplikace. Po zadání předchozího příkazu Ionic vytvoří projekt nastiňující souborovou strukturu, kterou by se měl vývojář při implementaci aplikace řídit. Klíčová při doplňování vlastních zdrojových kódů je složka *src*, jejíž obsah ukazuje obrázek 8. Dále budou popsány vybrané podsložky a moduly aplikace v nich uložené.



Obrázek 8 – struktura zdrojové složky src

[27] Příkazy Ionic CLI a postupy v kapitolách 3.3 až 3.6 čerpají ze zdrojů [19], [42] a [43].

3.3.1 Složka app

Vstupním bodem aplikace je soubor *index.html* umístěný přímo ve složce *src*, který obsahuje kromě skriptů a kaskádových stylů především element *ion-app*, do kterého je celá aplikace při spuštění načtena[28].

Soubory upravující aplikaci jako celek (nikoliv pouze konkrétní komponentu) se nacházejí ve složce *app*. Zde je aplikační modul, do kterého jsou importovány všechny ostatní moduly a kde jsou definováni celoaplikační poskytovatelé služeb. Dále je zde také uložena výchozí komponenta aplikace, která, v případě použité šablony *sidemenu*, spravuje pole navigačních položek přístupných formou bočního menu, a šablona popisující strukturu jejího vzhledu. Složka *app* obsahuje také soubor pro definici vzhledu designových prvků, které jsou využívány napříč celou aplikací.

3.3.2 Složka models

Další důležitou částí je složka *models*, kde jsou uloženy dříve zmiňované modely ze struktury návrhového vzoru MVC. Ty jsou pomocí podsložek rozděleny do několika logicky souvisejících skupin. Skupiny *customer*, *product*, *invoice* a *user* seskupují třídy sloužící k dynamické reprezentaci dat načítaných ze vzdáleného serveru nebo místní databáze. Podsložka *order* obsahuje třídy využívané při tvorbě objektů souvisejících s právě vytvářenou objednávkou a jejich serializaci.

3.3.3 Složky pages a partials

Nejvýznamnějšími segmenty souborové struktury jsou složky *pages* a *partials*, kde jsou uloženy komponenty aplikace, tedy pohledy a ovladače stránek z již zmíněného MVC. Rozdíl mezi komponentami těchto dvou složek je ten, že zatímco první jmenovaná obsahuje komponenty využívané jako jednotlivé stránky webové aplikace (celostránkové), třídy ve složce *partials* implementují související nebo opakující se designové prvky stránek. Nejjednodušší cestou pro tvorbu komponent je využití příkazu Ionic CLI

```
ionic g page NázevKomponenty,
```

který na odpovídajících místech v adresářové struktuře vygeneruje potřebné podsložky a soubory. Tyto komponenty je pak třeba importovat v již zmíněném aplikačním modulu nacházejícím se ve složce *app*.

[28] Tento proces je označován jako *bootstrapping* – jednoduchá aplikace aktivuje a načte složitější.

Příkladem celostránkové komponenty ze složky *pages* může být například *update-page* spravující stránku aktualizace dat nebo *buy-page*, která vykresluje a řídí stránku určenou pro přidávání položek do právě vytvářené objednávky. Některé komponenty využívají pro část svého pohledu jinou komponentu ze složky *partials*. Konkrétně celostránková komponenta *catalog-page* využívá prvkové komponenty *catalog-pager-part* a *catalog-product-part* pro zobrazení opakujících se designových prvků, kterými jsou jednotlivé produkty na stránce. Za zmínku stojí i prvková komponenta *header-part* ze složky *partials*, která spravuje přizpůsobitelnou horní lištu stránky (hlavičku). Tato komponenta je využívána téměř ve všech celostránkových komponentách, ve kterých se hlavička vyskytuje. Její implementace je více přiblížena v rámci kapitoly 3.4.

3.3.4 Složka providers

Protože byly při návrhu aplikace odhaleny oblasti funkcí, ke kterým může přistupovat více komponent, bylo třeba tyto funkce zapouzdřit do služeb sdílených napříč komponentami. O správu služeb se ve struktuře projektu nástroje Ionic starají poskytovatelé služeb, kteří jsou následně prostřednictvím principu *dependency injection* využíváni jednotlivými komponentami. Jak bylo zmíněno dříve, pro implementaci návrhového vzoru *singleton* lze vložit poskytovatele přímo do aplikačního modulu a tím docílit vytvoření pouze jedné jeho instance.

Poskytovatelé jsou uloženi ve složce *providers*. Lze je jednoduše vytvořit pomocí příkazu Ionic CLI

```
ionic g provider NázevPoskytovatele,
```

který připraví potřebné soubory a třídy, včetně anotací. V aplikaci jsou dostupné například služby spravující stránky produktů katalogu, zákazníků, neodeslaných objednávek, či instanci SQLite databáze.

3.3.5 Složka theme

Implementace vzhledové stránky aplikace je rozdělena na několik částí. Jak již bylo zmíněno, kaskádové styly využívané napříč celou aplikací jsou zapsány ve složce *app*. Každá komponenta ze složky *pages* nebo *providers* pak využívá vlastní soubor se styly aplikovanými pouze na elementy této komponenty. Zvlášť jsou však umístěny styly pracující s barvami pozadí, písma nebo rámečků. Pro tyto styly byly ve složce *theme* vytvořeny dva soubory, které pro stejné elementy definují jejich světlou a tmavou variantu. Mezi těmito barevnými provedeními lze pak v aplikaci přepínat na stránce *Nastavení*. Složka *theme* také obsahuje

soubor definující proměnné preprocesoru Sass, kde mohou být rovněž přepisovány přednastavené proměnné nástroje Ionic upravující vzhled aplikace.

3.3.6 Další složky

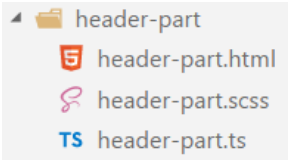
Ve složce *src* se nachází také podsložka *tools*, kde jsou uloženy dříve zmiňované nástroje jako jsou stránkovače, filtry nebo validátory. Za nástroje lze označit i roury pro formátování výstupu, ty jsou však pomocí příkazu Ionic CLI

```
ionic g pipe NázevRoury
```

generovány do vlastní složky (*pipes*). Poslední významnou podsložkou je složka *data*, kde jsou uloženy statické atributy aplikace jako nastavení databáze, texty upozornění využívané v rámci aplikace nebo některé výchozí hodnoty objektů.

3.4 Příklad implementace vybraného modulu

K demonstraci implementace konkrétního modulu byla vybrána komponenta *header-part* realizující hlavičku celostránkových komponent. Jedná se o komponentu ze složky *partials*.



Obrázek 9 – obsah složky komponenty header-part

Tato komponenta je implementována tak, aby bylo možné ji univerzálně nasadit v rámci téměř celé vyvíjené aplikace. S tím souvisí nutnost její parametrizace. Hodnoty atributů jsou zadávány při vytváření instancí této komponenty v nadřazených komponentách spravujících jednotlivé stránky. Na základě těchto hodnot je pak při načítání šablony této komponenty rozhodnuto o vykreslení některých prvků (vyhledávací pole), jsou přizpůsobeny zobrazované řetězce (titulek stránky) nebo formována napojení jednotlivých prvků šablony na dynamické proměnné odpovídajícího modelu pomocí principu *data-binding* (vyhledávaný řetězec).

Stejně jako každá komponenta v aplikaci je i *header-part* rozdělena do tří souborů, jak je patrné z obrázku 9. Soubor s příponou *ts* obsahuje ovladač komponenty realizovaný v jazyce TypeScript pomocí anotované třídy. Zde jsou uchovány parametry této komponenty společně s metodami zpřístupňujícími potřebné vlastnosti nebo chování. Ukázka z tohoto souboru je na následujícím obrázku 10.

```
// ...
import { Component, Input, OnInit } from "@angular/core";

@Component({
  selector: "header-part",
  templateUrl: "header-part.html"
})
export class HeaderPart implements OnInit {
  @Input() protected headerConfig: HeaderSettings;
  protected searchbarPlaceholder: string = "Hledej";
  protected userPopover: Popover;
  protected popoverPresented: boolean = false;
  private filterSubject = new Subject();

  constructor(
    public popoverCtrl: PopoverController,
    protected orderService: OrderService,
    protected userService: UserService) { }

  public ngOnInit() {
    if (this.headerConfig.titlePager !== undefined && this.headerConfig.titlePager !== null) {
      // to prevent DB from freezing on multiple updates when typing, debounce time is set to 0,5 s
      // (DB asked only after 0,5s of not typing)
      this.filterSubject.debounceTime(500).subscribe(() => {
        this.headerConfig.titlePager.goToFirstPage();
      });
    } else {
      this.filterSubject.debounceTime(500);
    }
  }

  protected showCustomer(): boolean {
    return this.headerConfig.showCustomerName && this.orderService.hasCurrentOrder();
  }

  protected toggleUserPopover(event: any): void {
    if (!this.popoverPresented) {
      this.userPopover = this.popoverCtrl.create(UserPage, {}, { cssClass: "user-popover" });
      this.userPopover.present({ ev: event });
      this.popoverPresented = true;
    } else {
      this.userPopover.dismiss();
      this.popoverPresented = false;
    }
  }

  protected lazyPageReload(): void {
    this.filterSubject.next();
  }
  // ...
```

Obrázek 10 – ukázka implementace ovladače komponenty header-part

Šablona napojená na tuto komponentu je uložena v souboru s příponou *html*. Kromě klasických HTML tagů se zde vyskytují také elementy frameworku Ionic (označené předponou ion) a šablonové prvky Angularu (například rozhodování o zobrazení na základě proměnné pomocí formule *ngIf). Celý soubor je zobrazen na obrázku 11.

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle *ngIf="headerConfig.showMenuButton"
      [disabled]="headerConfig.disableMenuButton" >
      <ion-icon name="menu"></ion-icon>
    </button>
    <div *ngIf="showCustomer()" class="bar-customer-name" >
      {{ getCustomerName() }}
    </div>
    <ion-title>
      {{ headerConfig.title }}
      <span *ngIf="headerConfig.showTitlePager">
        #{{ getCurrentPageNumber() }}/{{ getTotalAmountOfPages() }}
      </span>
    </ion-title>
    <button ion-button clear *ngIf="headerConfig.showUser" (click)="toggleUserPopover($event)"
      [disabled]="headerConfig.disableMenuButton" class="end">
      <ion-icon name="contact"></ion-icon>
    </button>
    <ion-searchbar *ngIf="headerConfig.showSearchbar"
      [(ngModel)]="headerConfig.searchbarModel.filterString" (ngModelChange)="lazyPageReload()"
      placeholder="{{ searchbarPlaceholder }}" class="end">
    </ion-searchbar>
  </ion-navbar>
</ion-header>
```

Obrázek 11 – šablona (pohled) komponenty header-part

Posledním částí jsou kaskádové styly ovlivňující chování pouze této komponenty. Ty jsou uloženy v souboru s příponou *scss*, jehož podoba je na obrázku 12. Kromě klasické syntaxe jednotlivých stylů jsou zde patrné také prvky preprocesoru Sass jako zanoření bloků nebo využití sdílených proměnných[29].

Všechny specifické styly jsou obaleny stylem definovanými selektory „*.ios, .md*“, který slouží především k dosažení vyšší míry specifičnosti těchto stylů. Důvodem je skutečnost, že při konfliktu stylů, kdy je definováno více hodnot stejné vlastnosti na jednom elementu stránky se využije hodnota z toho stylu, který je nejspecifičtější. Cílem je tedy implementovat styly více specifické, než jsou ty výchozí definované nástrojem Ionic.

Další obalový styl je definován selektorem „*header-part*“ (bez tečky). Ten zajišťuje to, že specifické styly implementované v rámci tohoto souboru budou aplikovány pouze v rámci této komponenty a neovlivní zbytek aplikace. Pokud by nějaký styl měl být aplikován rovněž na jiné komponenty, bude přesunut do dříve zmíněného souboru celoaplikačních stylů ve složce *app*.

[29] Ve skutečnosti se nejedná o proměnné v typickém smyslu slova, neboť jejich hodnota je před spuštěním programu pouze dosazena do míst, kde jsou dané proměnné využity. Jedná se o nástroj vhodný ke sdílení hodnot napříč zdrojovými kódy kaskádových stylů. Přesnější by však bylo označovat tyto hodnoty jako konstanty.

```scss
.ios, .md {
  header-part {
    .end {
      float: right;
    }

    ion-searchbar {
      width: 30%;
      height: $header-height;
    }

    .toolbar-content > div {
      position: relative;
      height: $header-height;
      line-height: $header-height;
      font-size: 1.1em;
    }

    .bar-customer-name {
      padding-left: 10px;
      width: 30%;
      overflow-y: auto;
      float: left;
      white-space: nowrap;
    }
  }
}
```

Obrázek 12 – implementace kaskádových stylů komponenty header-part

Pro použití této komponenty stačí jednoduše vložit element s názvem definovaným v anotaci třídy této komponenty do šablony komponenty nadřazené. Obrázek 13 ukazuje také formu předání parametrů, které jsou následně zpracovány v ovladači komponenty pomocí proměnné anotované klíčovým slovem *„Input“* (viz obrázek 10).

```html
<header-part [headerConfig]="headerConfig"></header-part>
<!-- ... -->
```

Obrázek 13 – ukázka použití komponenty header-part v komponentě nadřazené

3.5 Potřebné pluginy

Pro přístup k nativním funkcionalitám mobilního zařízení bylo třeba využít některé přídavné moduly (pluginy) nabízené nástrojem Cordova. Jak již bylo zmíněno, při vývoji hybridní mobilní aplikace lze tyto pluginy přidávat a spravovat přímo prostřednictvím Ionicu. Pro přidání doplňku do aplikace slouží příkaz Ionic CLI

```
ionic cordova add plugin název-pluginu,
```

který zařídí instalaci potřebných balíčků a jejich připojení k projektu.

Aplikace eKatalog využívá mimo jiné plugin pro práci s fotoaparátem mobilního zařízení, dále s jeho uložištěm, virtuální klávesnicí nebo geolokačními informacemi. Seznam všech použitých

pluginů, které byly v rámci práce použity, lze získat ze souboru *config.xml* umístěného v kořenovém adresáři projektu.

3.6 Spuštění, nasazení a testování aplikace při vývoji

Při vývoji bylo nutné testovat správnost implementace jednotlivých funkcionalit. Protože vývoj v nástroji Ionic probíhá na úrovni webové aplikace, která může být následně převedena na aplikaci hybridní, je možné projekt spustit a testovat přímo ve webovém prohlížeči počítače, kde je aplikace vyvíjena. Toho lze docílit spuštěním příkazu Ionic CLI

```
ionic serve,
```

který webovou aplikaci v prohlížeči spustí. K dispozici je užitečný přepínač *--lab*, pomocí nějž lze snadno ladit vzhledové odlišnosti mezi jednotlivými mobilními platformami. Tento příkaz také automaticky obnovuje zobrazovanou webovou aplikaci při uložení nových změn ve zdrojovém kódu.

Předchozí postup lze však spolehlivě uplatnit pouze v případě, kdy není potřeba využívat služeb některého z pluginů nástroje Cordova. Funkce mobilního zařízení, které tyto pluginy zpřístupňují, nejsou ve webovém prohlížeči dostupné. Zároveň je třeba otestovat konzistenci v chování některých designových komponent mezi webovým prohlížečem a kontejnerem WebView dané platformy. Z těchto důvodů je žádoucí prováděné změny testovat rovněž přímo na mobilním zařízení.

Před samotným nasazením aplikace na zařízení je nutné připojit k projektu požadované platformy, pro které má být aplikace připravena. K tomu slouží příkaz Ionic CLI

```
ionic cordova platform add NázevPlatformy.
```

Dalším krokem je připojení samotného zařízení k počítači pomocí konektoru USB. Důležité je také povolit v nastavení daného zařízení vývojářský režim připojení. Konkrétní umístění tohoto nastavení se liší v závislosti na operačním systému zařízení. Spustit aplikaci na zařízení lze pomocí příkazu Ionic CLI

```
ionic cordova run NázevPlatformy --device.
```

Připojit lze podle potřeby také přepínače *--prod* pro nasazení produkční verze aplikace nebo *--livereload* pro automatickou obnovu aplikace při uložení změn ve zdrojovém kódu.

Při vývoji pro platformu iOS je třeba tento příkaz spustit na operačním systému MAC OS X. Dále je nutné z důvodu bezpečnostních politik společnosti Apple nasazení rozdělit do dvou

částí. Nejprve je vytvořen aplikační balíček, například pomocí výše zmíněného příkazu[30]. Při tomto procesu bude mimo jiné vytvořen soubor s příponou *xcodeproj* v adresáři *platforms/ios*. Tento soubor lze následně otevřít pomocí vývojového prostředí pro aplikace platformy iOS zvaného Xcode. Zde je možné po vytvoření profilu vývojáře a „podepsání" aplikace vybrat připojené zařízení a provést samotné nasazení. [19]

Po jejich nasazení se projeví další výhoda hybridních aplikací. Protože jde vnitřně stále o aplikace webové, je možné je i nadále testovat pomocí webového prohlížeče. Aplikace pro platformu iOS lze testovat v prohlížeči Safari, aplikace pro Android pomocí prohlížeče Chrome. Stručný postup připojení zařízení platformy Android k počítači, nasazení aplikace a spuštění testovacího nástroje v Chromu vypadá následovně:

1. Připojte zařízení k počítači pomocí konektoru USB.
2. Zapněte v nastavení zařízení rozšířené vývojářské možnosti a povolte USB debugging (konkrétní nastavení se liší v závislosti na daném zařízení).
3. V příkazovém řádku počítače ve složce projektu spusťte příkaz pro spuštění aplikace na zařízení.
4. Na počítači otevřete webový prohlížeč Chrome.
5. V Chromu vyberte v nastavení pod záložkou „Další nástroje" položku „Nástroje pro vývojáře"
6. Pomocí menu tří vertikálních teček na nové liště nástroje pro vývojáře vyberte pod záložkou „Další nástroje"[31] položku „Připojená zařízení"[32].
7. V nově otevřené záložce vyberte z připojených zařízení to, na kterém je aplikace spuštěna.
8. Na stránce spuštěných webových aplikací tohoto zařízení, kam budete přesměrováni, klikněte na tlačítko „Prozkoumat"[33] vedle názvu spuštěné aplikace.
9. Dojde k otevření nového okna prohlížeče napojeného na aplikaci spuštěnou na zařízení.

Po provedení předchozích kroků je možné s aplikací v prohlížeči pracovat podobně jako s běžnou webovou aplikací. Lze sledovat výpisy a chybové hlášky v konzoli, prohlížet jednotlivé elementy stránky nebo upravovat kaskádové styly. Všechny tyto nástroje značně usnadňují ladění především designové stránky aplikace. Dále je možné také sledovat

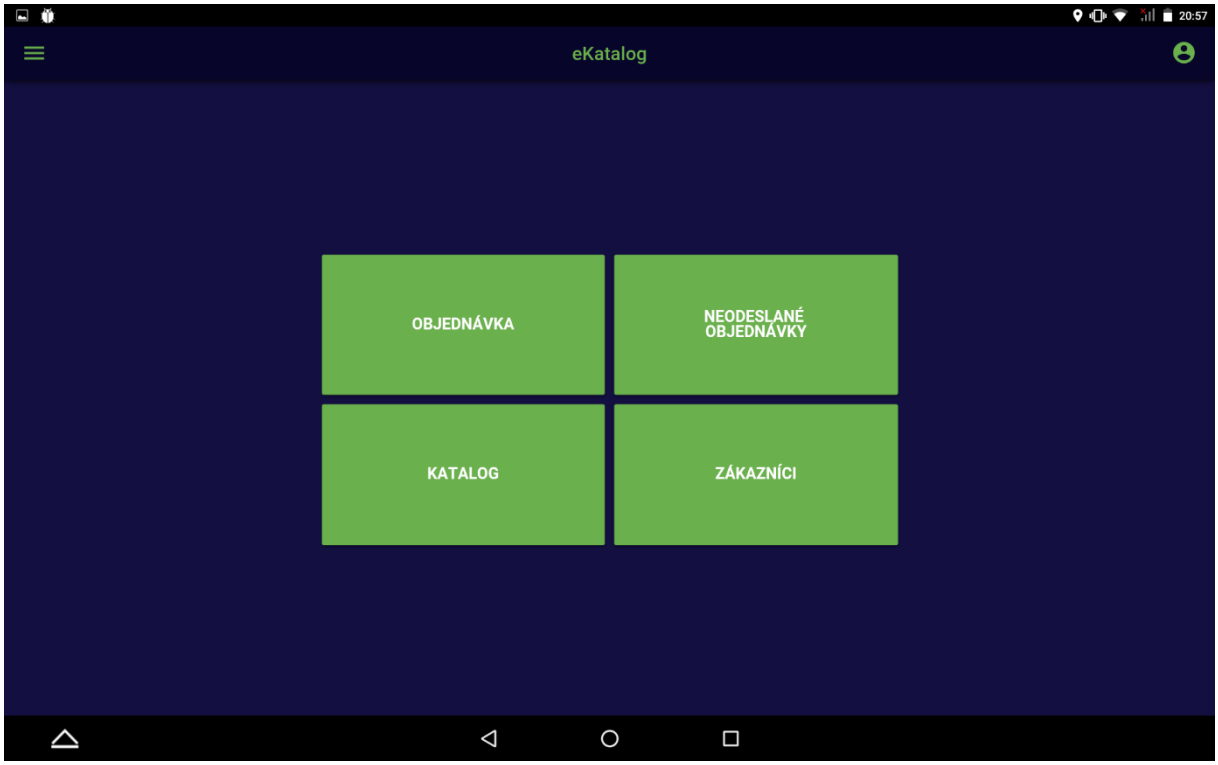
[30] Vhodnější by pro tento účel byl příkaz *ionic build ios*, případně doplněný o přepínač *--prod.*
[31] *More tools.*
[32] *Remote devices.*
[33] *Inspect.*

internetový provoz aplikace (odesílané dotazy a obdržené odpovědi), ladit výkon animovaných prvků nebo zobrazit zdrojové kódy aplikace a postupovat v nich krok po kroku při hledání jejich nedostatků. Zároveň je možné mít ve stejném okně prohlížeče zobrazen také aktuální obraz aplikace běžící na připojeném zařízení, který slouží k lepší orientaci mezi jednotlivými prvky aplikace. [43]

V průběhu vývoje eKatalogu byly pro účely ladění a testování využity tablet Lenovo TAB 3 Business 10,1" pro implementaci na platformě Android, a iPad Pro 9,7" pro implementaci platformy iOS. Obě verze aplikace byly laděny v příslušných prohlížečích.

4 POPIS FUNKCIONALIT APLIKACE

V této kapitole budou postupně představeny vybrané funkcionality implementované ve výsledné aplikaci. Nejprve bude představen základní průchod aplikací z pohledu prodejce obchodní společnosti, který by danou aplikaci potenciální využíval. Následně budou popsány doplňující funkcionality, které lze v aplikaci využít.



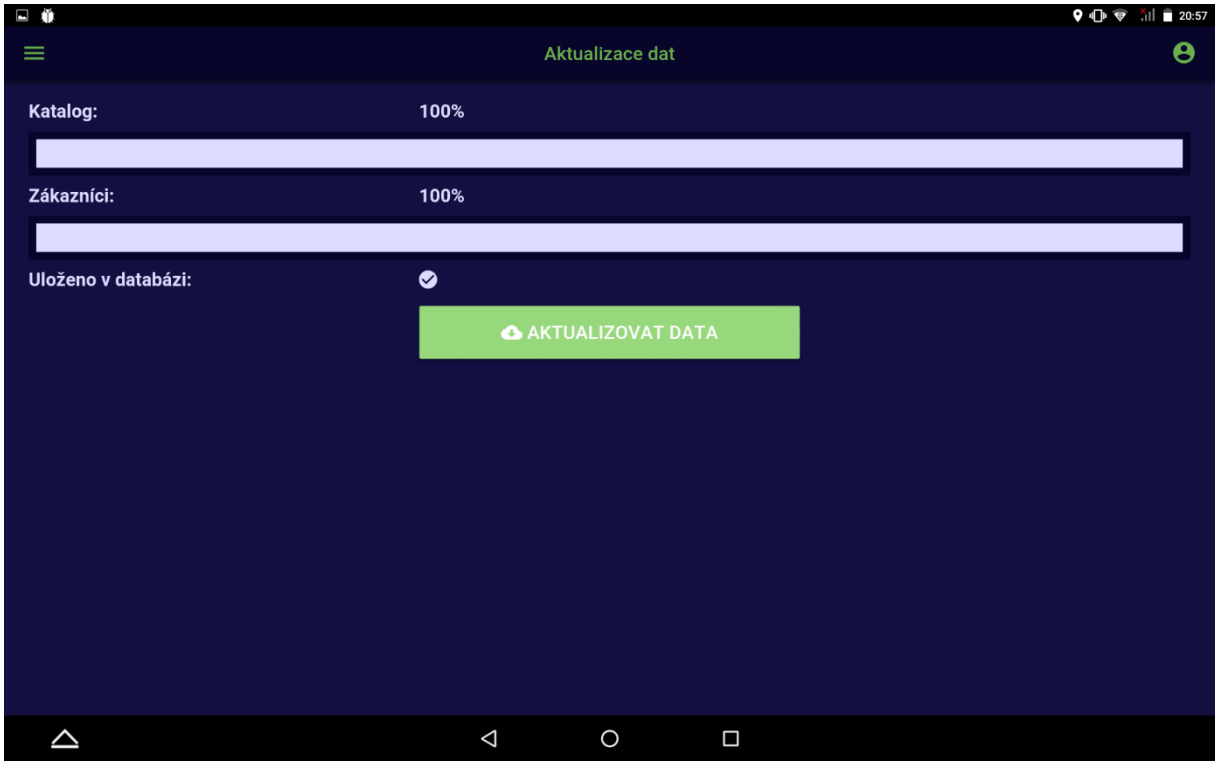
Obrázek 14 – úvodní obrazovka aplikace po přihlášení uživatele

4.1 Základní funkcionality

Prvním krokem uživatele po spuštění aplikace musí být ověření jeho identity. Bez zadání platných přihlašovacích údajů není možné přistupovat k údajům prodejní společnosti a jejích zákazníků. Aplikace nepracuje s registrací nových uživatelů, neboť tato služba spadá pod správu samotné obchodní společnosti. Předpokladem pro přihlášení jsou tedy přihlašovací údaje získané od správce informačních systémů dané společnosti.

Po vyplnění příslušných vstupů na přihlašovací obrazovce jsou tyto odeslány na stranu serveru, kde dojde k jejich ověření. Uživatelské heslo je po celou dobu jeho evidence uchováno v hashované podobě. V případě zadání platných údajů jsou ze serveru do aplikace odeslány informace o daném uživateli, které jsou dále uloženy do lokálního uložiště mobilního zařízení. Při dalším spuštění aplikace jsou tyto informace využity k automatickému přihlášení posledního uživatele, pokud se tento před jejím vypnutím neodhlásil.

Po úspěšném přihlášení je uživatel přesměrován na hlavní stránku aplikace zobrazené na obrázku 14, kde jsou zobrazeny některé nejdůležitější nabídky aplikace. Od této chvíle je také formou tlačítka v levém horním rohu obrazovky dostupná nabídka všech funkcionalit aplikace.



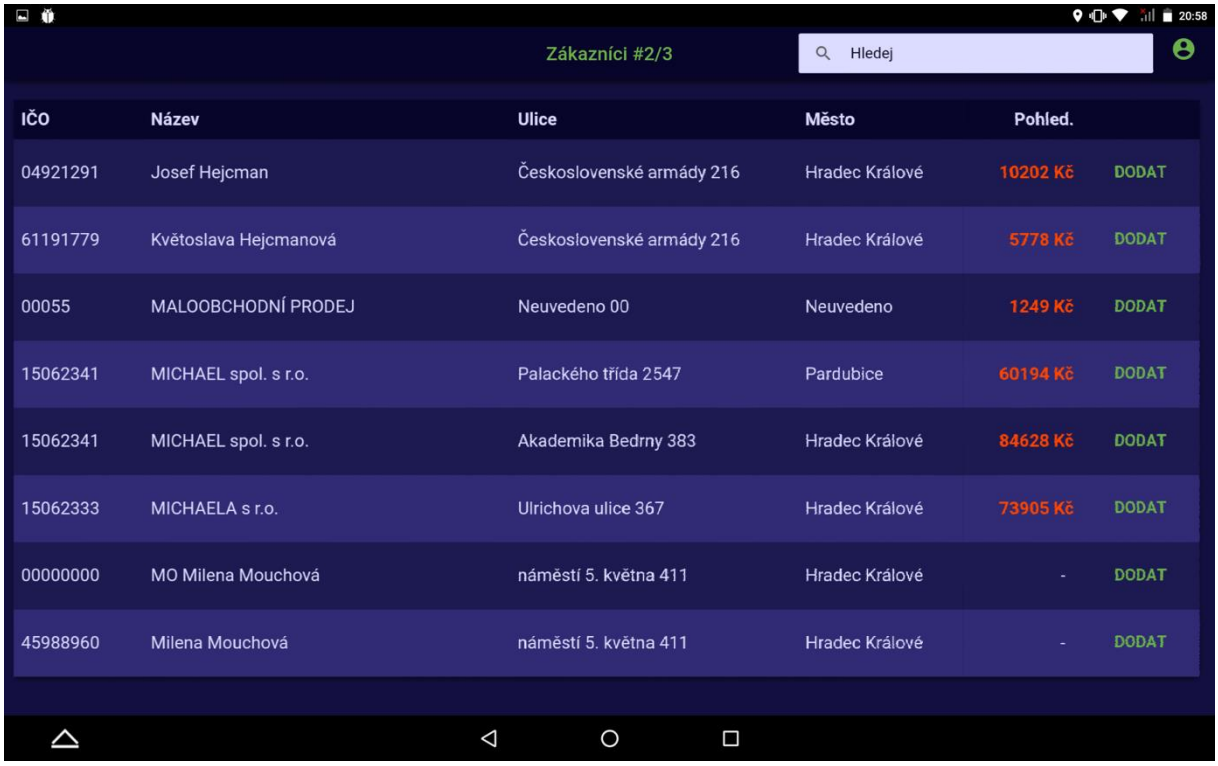
Obrázek 15 – modul Aktualizace dat po úspěšném stažení a lokálním uložení dat

Dalším krokem prodejce je získání dat potřebných pro tvorbu objednávek. Jde především o katalog produktů společnosti a seznam jejích zákazníků. Tato data jsou pomocí modulu *Aktualizace dat* (obrázek 15) stažena ze vzdálené MySQL databáze a uložena do lokálního uložiště SQLite. Doba potřebná k přenosu informací se může lišit v závislosti na objemu stahovaných dat a rychlosti internetového připojení. Z tohoto důvodu je žádoucí aktualizovat data v blízkosti kvalitního připojení, například ještě před služební cestou za zákazníky společnosti.

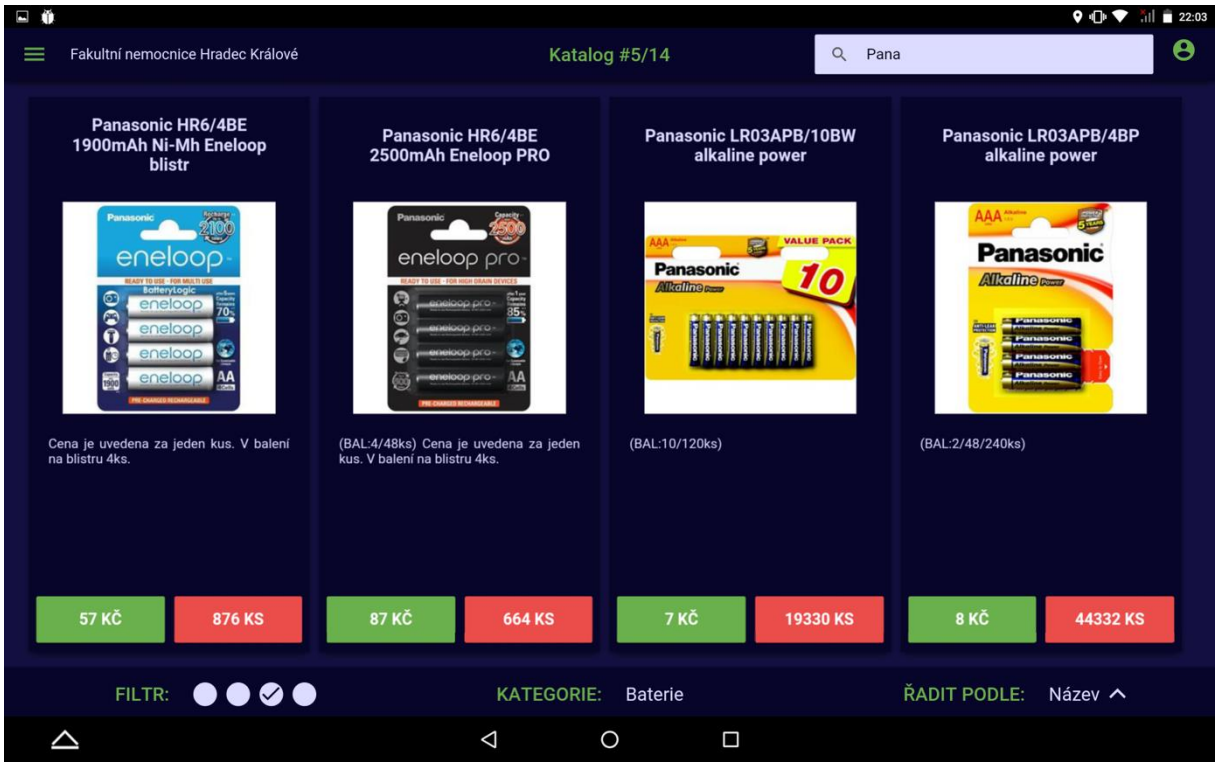
V závislosti na firemní politice je také nutné určit frekvenci stahování nutnou k udržení aktuality stažených dat. Pokud by aktualizace cen, přidávání zákazníků a doplnění produktů probíhalo například pouze v nočních hodinách, bylo by postačující data aktualizovat jednou denně, vždy před první služební cestou.

Jakmile jsou data stažena a uložena v lokální databázi, je možné je prohlížet pomocí modulů *Zákazníci* a *Katalog zboží* (obrázky 16 a 17). Oba moduly využívají dříve zmíněný komplexní

stránkovač umožňující animovat přechod mezi jednotlivými stránkami katalogů. Listování je realizováno pomocí gesta přetažení prstem zprava doleva pro posun na další stranu, obráceným směrem pak pro zobrazení strany předchozí.



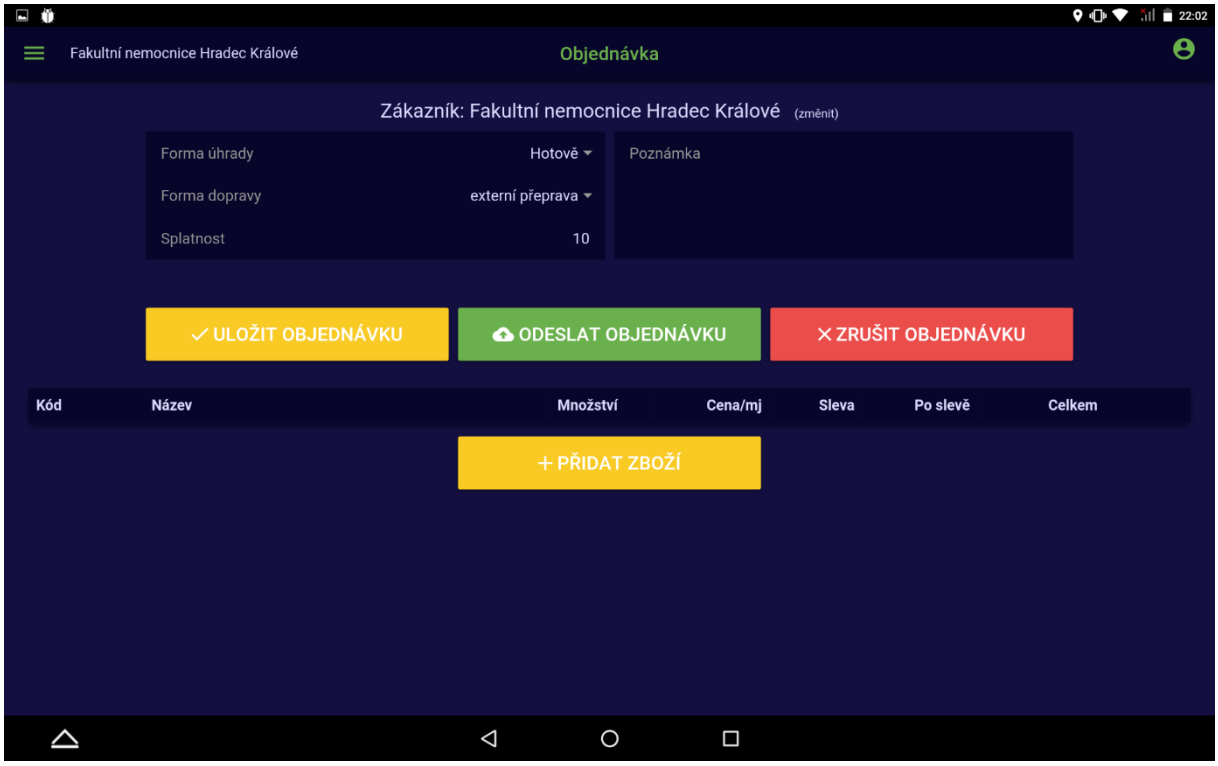
Obrázek 16 – katalog zákazníků při vytváření objednávky



Obrázek 17 – katalog produktů při vytváření objednávky

V obou seznamech je možné vyhledávat pomocí pole v horní liště aplikace. Zákazníky lze vyhledávat podle jejich jména, identifikačního čísla organizace nebo adresy, produkty pouze podle jména. U produktů jsou však také dostupné rozšířené možnosti filtrace. Je možné zobrazit jen určité kategorie produktů nebo je filtrovat podle nastavených příznaků, jako je příznak „Doporučujeme“ nebo „Novinka“. Dále je možné produkty řadit podle jejich identifikačního čísla nebo názvu, pro obě možnosti je dostupné jak vzestupné, tak i sestupné řazení. Prvky filtrace jsou dostupné v dolní liště modulu. Všechny tyto funkce lze navíc kombinovat, a to včetně vyhledávání.

U produktů je také implementován dynamický princip vyhledávání. Při zadání prostého textu je výraz porovnáván pouze s názvy jednotlivých produktů. Pokud však řetězec zadaný do vyhledávacího pole obsahuje symbol lomítko, je tento text podle něj rozdělen na dvě části. První část (před lomítkem) je porovnávána s výrobci jednotlivých produktů, pomocí druhé části (za lomítkem) je vyhledáváno mezi názvy položek katalogu. Metoda vyhledávání se tedy automaticky mění v závislosti na zadaném vstupu.



Obrázek 18 – stránka objednávky před přidáním jednotlivých položek

Hlavním zaměřením eKatalogu je tvorba objednávek. Ty může prodejce vytvářet v modulu *Objednávka*. Pokud není právě žádná objednávka k dispozici, nabízí se zde možnost jejího vytvoření. Uživatel je následně přesměrován do agendy zákazníků, kde vybere prostřednictvím tlačítka *Dodat* toho zákazníka, pro kterého bude objednávka vytvořena. Následně se aplikace

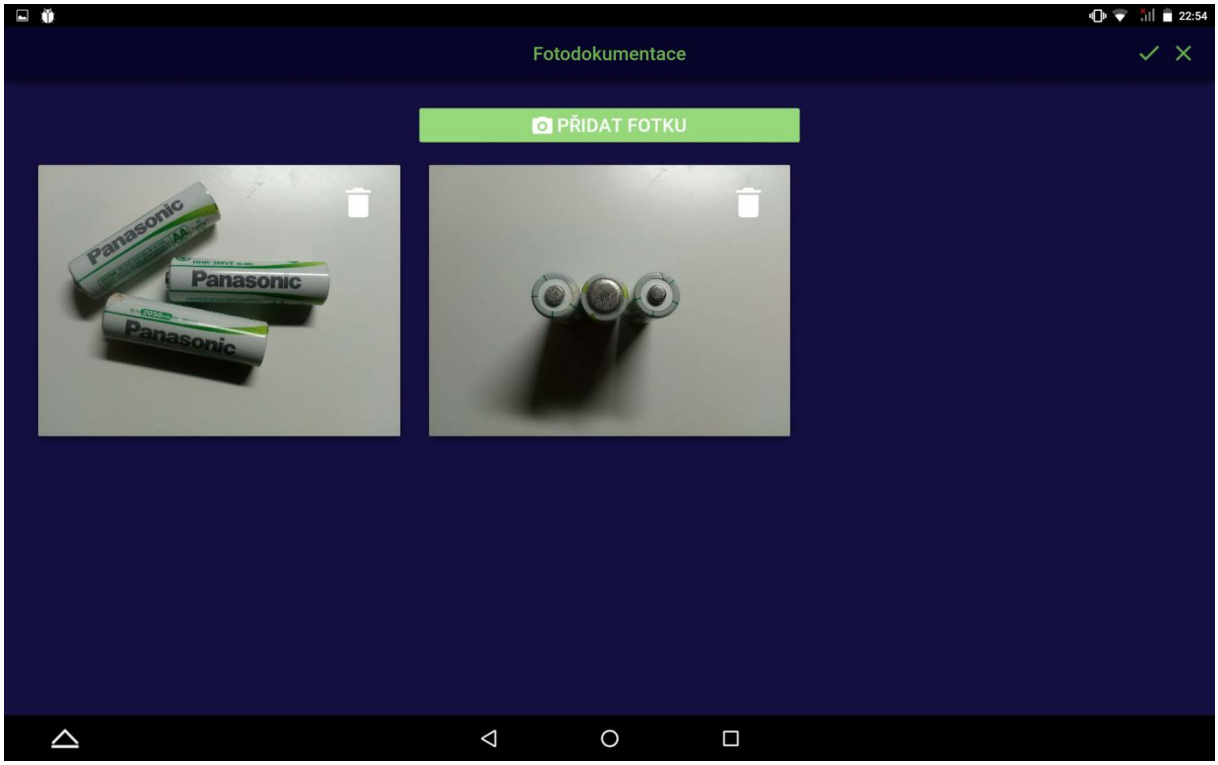
vrátí na stránku objednávky (obrázek 18), kde se nově nachází údaje o celkové objednávce, jako je například forma úhrady. Tyto informace jsou vyplněny podle preferencí vybraného zákazníka, lze je zde však upravit. Zároveň je možné změnit i výběr samotného zákazníka.

Protože objednávky typicky obsahují nějaké položky, bude dalším krokem prodejce nejspíše přidání produktů do nově vytvořené objednávky. Při výběru produktů je možné využít již dříve popsané metody filtrace katalogu. Produkty lze přidat jednak jako položky nákupu, a to prostřednictvím tlačítka zobrazujícího zároveň cenu produktu, ale také jako položky reklamace pomocí tlačítka zobrazujícího počet produktů skladem.

Obrázek 19 – přidávání nákupní položky do objednávky

Ceny jednotlivých produktů se odvíjejí od cenové úrovně zákazníka, pro nějž je objednávka vytvářena. Dále je možné prostřednictvím nákupního dialogu na obrázku 19 poskytnout zákazníkovi jistou slevu, ať už ve formě procentuální slevy z celkové částky dané položky, nebo formou akce, kdy zákazník při nákupu daného počtu kusů získá jeden kus zdarma. Lze také manipulovat se samotnou cenou jednoho kusu produktu, ovšem pouze do té míry, dokud nepřekročí minimální cenu výrobku stanovenou obchodní společností. Celý prodejní formulář je rovněž ověřován validačními nástroji, aby nevznikaly logické chyby (jako například nákup záporného počtu kusů výrobku).

Pokud je položka objednávky vytvářena jako reklamační, slouží k její dokumentaci jednak popis dané položky, je však dostupná i možnost fotodokumentace, díky které je možné viditelné závady později ověřit (viz obrázek 20). Počet snímků, které lze k jedné reklamaci připojit je z důvodu snížení objemu přenášených dat omezen na tři.



Obrázek 20 – ukázka práce s multimédii v rámci reklamační položky objednávky

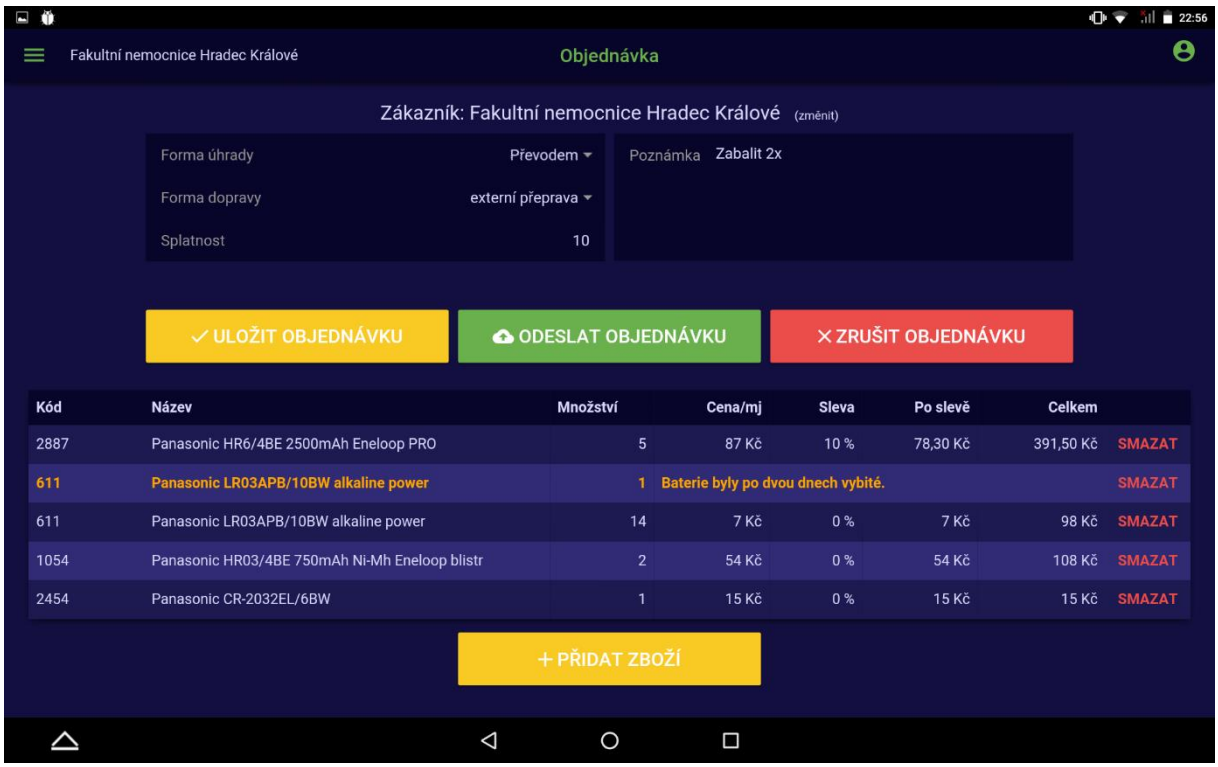
V průběhu přidávání jednotlivých položek do objednávky je možné sledovat počet již přidaných položek, stejně jako současnou cenu celé objednávky. Obě tyto informace jsou dostupné pomocí ikony v pravém rohu horní lišty aplikace.

Po přidání všech požadovaných položek do objednávky je třeba se vrátit zpět do modulu *Objednávka*, jehož aktualizovaná podoba je zobrazena na obrázku 21. Tam lze jednotlivé položky zkontrolovat a v případě potřeby upravit. Dále jsou zde možnosti k dokončení objednávky. Je možné ji smazat, uložit pro pozdější odeslání nebo přímo odeslat. Pokud pokus o odeslání skončí neúspěchem je objednávka uložena do lokální databáze.

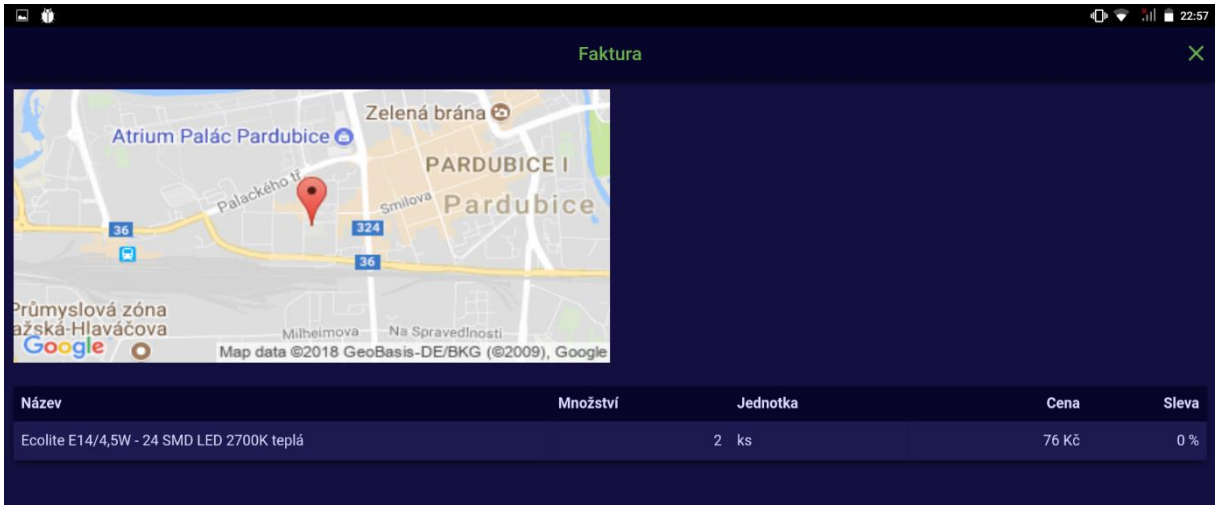
K uloženým neodeslaným objednávkám lze přistoupit pomocí modulu *Neodeslané objednávky*. Zde je možné objednávky prohlížet, mazat nebo se znovu pokusit o jejich odeslání na vzdálený server.

Po úspěšném odeslání objednávky k ní může prodejce přistoupit přes modul *Faktury* (obrázek 22), kde jsou dostupné všechny objednávky uložené ve vzdálené centrální databázi

společnosti, včetně jejich položek a GPS souřadnic místa, odkud byla objednávka vytvořena[34]. Ke stažení těchto objednávek je ovšem třeba připojení k internetové síti.



Obrázek 21 – stránka aktuální objednávky po přidání jejích položek



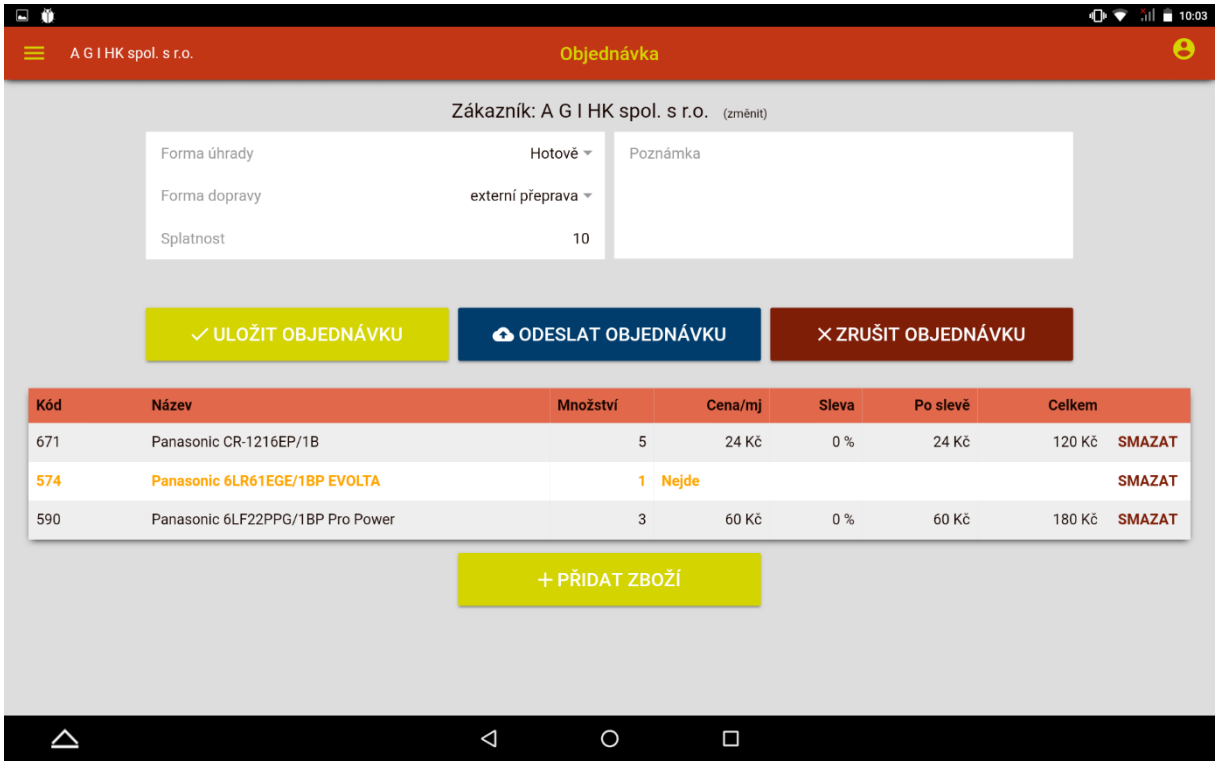
Obrázek 22 – ukázka práce s geolokačními údaji

Tímto krokem končí postup vytváření a odeslání objednávky. Prodejce se může odhlásit, ukončit aplikaci bez odhlášení a přesunout se k dalšímu zákazníkovi, nebo vytvořit další objednávku pro stejného zákazníka.

[34] Pokud se nepodařilo při vytváření objednávky získat informace o poloze uživatele, je zobrazen zástupný obrázek.

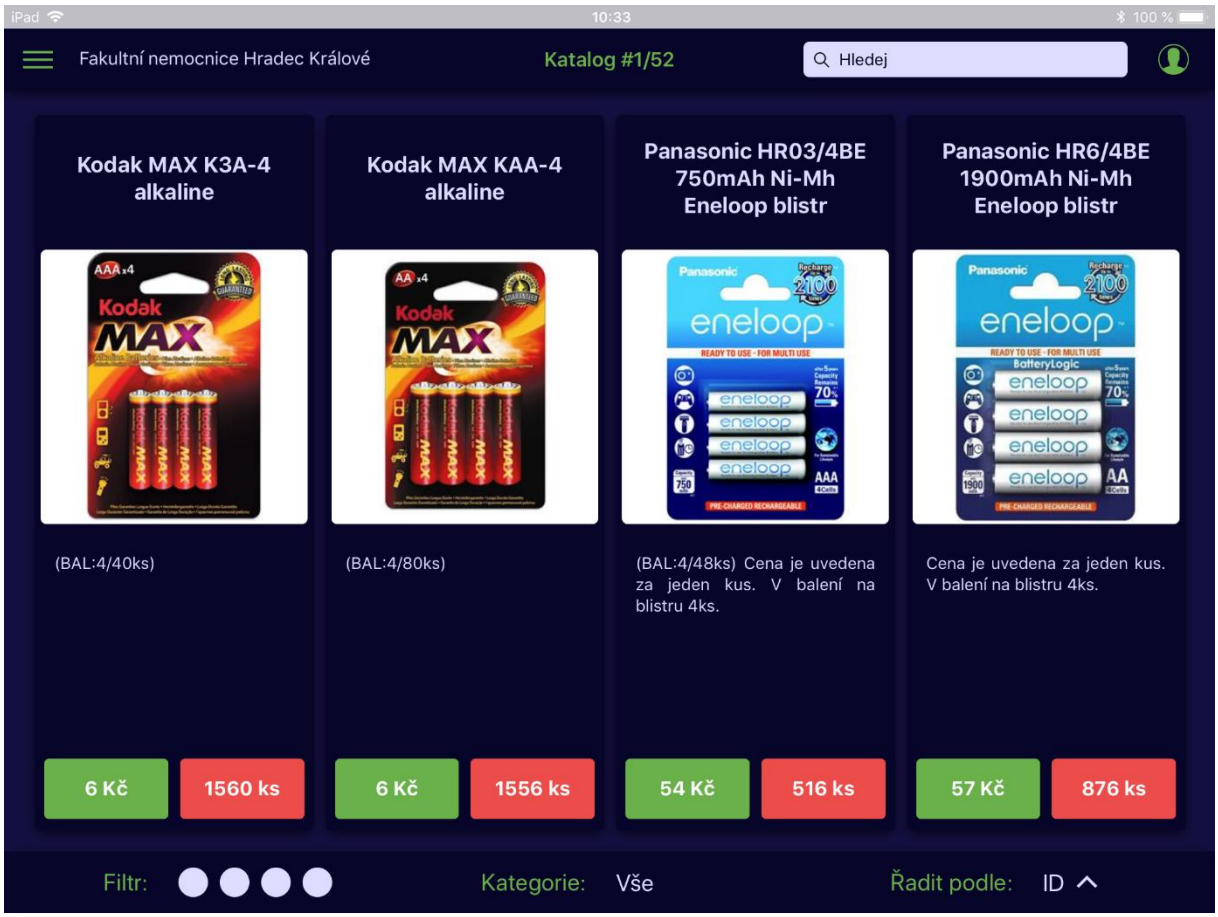
4.2 Další možnosti užití aplikace

V rámci modulu *Zákazníci* je možné sledovat také doposud neuhrazenou částku z minulých objednávek zákazníka. Po kliknutí na danou sumu se otevře stránka aplikace s detailnějšími informacemi, jako jsou čísla vydaných dokladů nebo data splatnosti. Tyto informace poslouží prodejci k rozhodnutí, zda neplatícímu zákazníkovi vytvářet novou objednávku, u které by existovalo riziko, že rovněž nebude splacena.



Obrázek 23 – stránka objednávky v alternativní barevné variantě

Dále je v aplikaci dostupný modul nastavení. V něm lze získat informace o právě přihlášeném uživateli a změnit motiv aplikace. K dispozici je jeden motiv tmavý a jeden světlý (demonstrovaný na obrázku 23). Aplikace je však implementována tak, aby bylo možné v případě potřeby stávající motivy upravit nebo doplnit. Díky tomu je možné snadno přizpůsobit barevnou paletu na míru konkrétní obchodní společnosti. Aplikace je také rozdílně vizualizována na jednotlivých platformách. Zatímco všechny ostatní obrázky této kapitoly ukazují vzhled aplikace na platformě Android, obrázek 24 prezentuje její verzi nasazenou na iOS. Jde o modul katalogu produktů, jehož verze z Androidu je pro porovnání na obrázku 17.



Obrázek 24 – vzhled katalogu produktů na platformě iOS

Jak již bylo několikrát zmíněno, aplikace umožňuje také odhlášení uživatele. Při odhlášení prostřednictvím záložky *Odhlásit* v levém menu je uživatel přesměrován na přihlašovací obrazovku aplikace a z lokální databáze jsou odstraněny veškeré jeho informace.

ZÁVĚR

V úvodní kapitole diplomové práce byly představeny vybrané metody vývoje mobilních aplikací. Jednalo se konkrétně o nativní, hybridní a kompilovaný přístup.

Nativní aplikace vynikají svým výkonem a možnostmi implementace. Jejich hlavní nevýhodou je nutnost vývoje více samostatných aplikací, pokud by celý projekt cílil na uživatele různých platforem. Při jejich vývoji jsou používány jazyky určené společnostmi spravujícími dané platformy.

Hybridní aplikace umožňují do značné míry sjednotit vývoj aplikací více platforem do jedné multiplatformní. Využívají k tomu princip zapouzdření vyvíjené webové aplikace do transparentního prohlížeče – kontejneru WebView. Uvnitř tohoto kontejneru jsou pak webové aplikace spouštěny a uživateli se jeví podobně jako aplikace nativní. Takovou aplikaci lze ovšem nasadit na jakoukoliv platformu, která obsahuje zmiňovaný kontejner.

Hybridní aplikace se potýkají s několika nedostatky, z nichž nejvýraznější je jejich nižší výkon v porovnání s aplikacemi nativními a závislost na technologiích třetích stran, nikoliv pouze na dané platformě. Z těchto důvodů je důležité zvolit vhodný nástroj vývoje hybridních aplikací. Příkladem může být framework Ionic pracující s nástrojem Cordova. Obě tyto technologie jsou stále vyvíjeny a podporovány, navíc také postupně optimalizují svůj výkon, čímž se přibližují aplikacím nativním.

Největší výhodou hybridních aplikací, kromě již zmíněné přenositelnosti mezi platformami, je skutečnost, že se vnitřně jedná stále o aplikace webové. Lze tedy využít jednak znalosti webových kodérů, kteří by chtěli přejít na vývoj mobilních aplikací, ale i dostupných knihoven široce podporovaného jazyka JavaScript.

Kompilované aplikace spojují některé výhody obou předchozích skupin. Jsou vyvíjeny pomocí jiných než nativních technologií (například webových), následně jsou však částečně překládány na aplikace nativní. Uchovávají si tedy výhodu přenositelnosti, jsou ovšem výkonnější než aplikace hybridní. Vývojáři jsou však opět závislí na technologiích třetích stran, které musí například zajistit přístup aplikace k nativním prvkům mobilního zařízení, jako je fotoaparát nebo datové uložiště.

Nelze jednoznačně doporučit jediný správný přístup k vývoji mobilních aplikací. Výběr metody implementace se vždy odvíjí od konkrétního projektu. Aplikace náročné na grafické výpočty je vhodné realizovat jako nativní z důvodu jejich nároků na využití výkonu mobilního zařízení.

Méně náročné aplikace mohou těžit z přenositelnosti aplikací hybridních nebo kompilovaných. Svou roli při výběru postupu mohou sehrát také předchozí znalosti a zkušenosti vývojáře.

Další části práce se věnují vývoji mobilní aplikace pro podporu tvorby objednávek. Tato aplikace je určena prodejcům obchodní společnosti cestujícím za jejich zákazníky. Usnadňuje nabídku produktů, tvorbu objednávek a jejich odeslání do informačního systému společnosti k dalšímu zpracování. Protože se nejedná o aplikaci požadující maximální využití výkonu mobilního zařízení, a zároveň je vhodné implementovat přenositelnou aplikaci, aby si sama společnost mohla zvolit platformu, na které bude aplikaci využívat, byla zvolena implementace formou hybridní aplikace s využitím zmíněných nástrojů Ionic a Cordova.

Před dokumentací samotného vývoje aplikace jsou v práci stručně představeny další technologie a nástroje, které byly při vývoji použity. Popsány jsou jazyky TypeScript, JavaScript a HTML, framework Angular, kaskádové styly včetně jejich preprocesoru Sass nebo databázové technologie SQLite a MySQL. Zmíněny jsou rovněž vývojová prostředí a další nástroje usnadňující celý vývoj, například VS Code nebo MySQL Workbench.

Poté je přiblížen postup návrhu a vývoje aplikace. Popsány jsou některé metody související s vývojem, které byli při implementaci využívány. Jde například o nasazení aplikace na mobilní zařízení nebo její následné testování. Poslední kapitola demonstruje jednotlivé funkcionality implementované aplikace na příkladu jejího užití z pohledu prodejce.

V rámci práce byly splněny všechny stanovené cíle. Po představení a porovnání jednotlivých přístupů k vývoji mobilních aplikací byla zvolena vhodná metoda implementace obchodní aplikace vyvíjené v rámci praktické části této práce. Aplikace využívá požadované technologie, pracuje s multimédii ve formě fotek reklamovaných produktů, ukládá časové i GPS údaje o místě pořízení objednávky a uchovává požadované informace v lokálním uložišti mobilního zařízení pro zajištění jejich dostupnosti bez nutností aktivního připojení k internetové síti.

Zdrojové kódy přiložené aplikace jsou přehledně strukturované podle doporučení daných nástrojů. Kvůli zajištění vyšší míry čitelnosti a udržitelnosti kódu byly při vývoji použity nástroje pro zlepšení kvality kódu Prettier a TSLint. Barevné schéma a některé další prvky aplikace byly koncipovány tak, aby bylo snazší je modifikovat na míru konkrétní společnosti, která by se rozhodla aplikaci pro tvorbu svých objednávek využít. Jak vývoj, tak i samotná aplikace byly v rámci této práce patřičně zdokumentovány.

Při implementaci aplikace byly řešeny různé typy problémů od modulárního návrhu přes implementaci jednotlivých komponent až po optimalizaci výkonu animovaných prvků.

Příkladem komplexnějšího problému je trojitý stránkovač spravující kromě aktuální stránky produktů rovněž stránku předchozí a následující. Při jeho implementaci bylo třeba správně propojit jednotlivé spolupracující komponenty tak, aby uchovávaná data korespondovala s prezentační vrstvou aplikace. Podle informací získaných z analýzy kódu nástrojem GitHub, pomocí nějž byly jednotlivé verze aplikace spravovány a zálohovány, byl při vývoji mobilní aplikace nejvíce využíván programovací jazyk TypeScript (téměř 68 % zdrojového kódu), následovaný kaskádovými styly (přibližně 20 %) a jazykem HTML (okolo 12%).

Celá aplikace vyvinutá v rámci této práce byla úspěšně nasazena a testována na zařízeních obou cílových platforem. Jednalo se konkrétně o tablet Lenovo TAB 3 a iPad Pro. Pro účely ověření jednotlivých funkcionalit mobilní aplikace byla serverová část nasazena a spuštěna s využitím internetové platformy Microsoft Azure.

Práci by bylo možné dále rozšiřovat podle potřeb konkrétní společnosti. Vhodným doplňkem by bylo například využití statistik prodeje jednotlivých produktů konkrétním zákazníkům, což by umožnilo efektivnější nabídku těchto produktů. Dalším rozšířením by mohl být sběr statistik samotných prodejců, který by managementu společnosti posloužil při jejich hodnocení.

POUŽITÁ LITERATURA

[1] Mobile Application (Mobile App). In: *Techopedia: The IT Education Site* [online]. [cit. 2018-05-12]. Dostupné z: https://www.techopedia.com/definition/2953/mobile-application-mobile-app

[2] BROWN, MARCEL. The "Lost" Steve Jobs Speech from 1983: Foreshadowing Wireless Networking, the iPad, and the App Store. In: *Life, Liberty, and Technology* [online]. 2.10.2012 [cit. 2018-05-10]. Dostupné z: http://lifelibertytech.com/2012/10/02/the-lost-steve-jobs-speech-from-1983-foreshadowing-wireless-networking-the-ipad-and-the-app-store/

[3] STRAIN, Matt. 1983 to today: a history of mobile apps. In: *The Guardian* [online]. 13.2.2015 [cit. 2018-05-22]. Dostupné z: https://www.theguardian.com/media-network/2015/feb/13/history-mobile-apps-future-interactive-timeline

[4] IPhone 3g Commercial. In: *Youtube* [online]. 4.2.2009 [cit. 2018-05-09]. Dostupné z: https://www.youtube.com/watch?v=szrsfeyLzyg

[5] Platform. In: *Techopedia: The IT Education Site* [online]. [cit. 2018-05-12]. Dostupné z: https://www.techopedia.com/definition/3411/platform

[6] LACKO, Ľuboslav. Mistrovství Android. Přeložil Martin HERODEK. Brno: Computer Press, 2017. ISBN 978-80-251-4875-4.

[7] KARCH, Marziah. What Is Google Android? In: *Lifewire: tech untangled* [online]. 2.10.2017 [cit. 2018-05-12]. Dostupné z: https://www.lifewire.com/what-is-google-android-1616887

[8] NATIONS, Daniel. What Is the iPhone OS (iOS)? In: *Lifewire: tech untangled* [online]. 15.2.2018 [cit. 2018-05-10]. Dostupné z: https://www.lifewire.com/what-is-ios-1994355

[9] ACADEMIND. React Native vs Ionic vs NativeScript vs Android/ iOS Native Apps. In: *Youtube* [online]. 10.12.2017 [cit. 2018-04-15]. Dostupné z: https://www.youtube.com/watch?v=rb8smP_xTTY

[10] Hybrid vs. Native: An Introduction to cross-platform hybrid development for architects and app development leaders [online]. 2017 [cit. 2018-05-15]. Dostupné z: https://ionicframework.com/books/hybrid-vs-native

[11] Native Mobile App. In: *Techopedia: The IT Education Site* [online]. [cit. 2018-05-12]. Dostupné z: https://www.techopedia.com/definition/27568/native-mobile-app

[12] MILLER, Paul. Google is adding Kotlin as an official programming language for Android development. In: *The Verge* [online]. 17.5.2017 [cit. 2018-05-15]. Dostupné z: https://www.theverge.com/2017/5/17/15654988/google-jet-brains-kotlin-programming-language-android-development-io-2017

[13] *Swift.org* [online]. 2014 [cit. 2018-05-12]. Dostupné z: https://swift.org/

[14] BRISTOWE, John. What is a Hybrid Mobile App? In: *Telerik: Developer Network* [online]. 25.3.2015 [cit. 2018-05-09]. Dostupné z: https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/

[15] GLOBETROTTER. Hybrid Applications And Android Native Browser. In: *MyShadesOfGray* [online]. 15.4.2014 [cit. 2018-05-09]. Dostupné z: https://myshadesofgray.wordpress.com/2014/04/15/hybrid-applications-and-android-native-browser/

[16] Foundation Project. *The Apache Software Foundation* [online]. [cit. 2018-05-15]. Dostupné z: https://www.apache.org/foundation/

[17] LEROUX, Brian. PhoneGap, Cordova, and what's in a name? In: *PhoneGap* [online]. 19.3.2012 [cit. 2018-05-09]. Dostupné z: https://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/

[18] Architectural overview of Cordova platform. *Apache Cordova* [online]. [cit. 2018-05-06]. Dostupné z: http://cordova.apache.org/docs/en/latest/guide/overview/index.html

[19] Ionic Documentation. *Ionic Framework* [online]. [cit. 2018-05-17]. Dostupné z: https://ionicframework.com/docs/

[20] GAURAV, Saini. Hybrid mobile development with Ionic: build high performance hybrid applications with HTML, CSS, and JavaScript. Birmingham, UK: Rackt Publishing, 2017. ISBN 978-1-78528-605-6.

[21] Introduction to Mobile Development. *Microsoft Docs* [online]. 28.3.2017 [cit. 2018-05-09]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/cross-platform/get-started/introduction-to-mobile-development

[22] Xamarin vs Ionic vs React Native: differences under the hood. In: *Crux lab* [online]. 2017 [cit. 2018-04-15]. Dostupné z: https://cruxlab.com/blog/reactnative-vs-xamarin/

[23] BROWN, Tiffany B., Kerry BUTTERS a Sandeep PANDA. *HTML5 okamžitě: ovládněte HTML5 za víkend*. Brno: Computer Press, 2014. ISBN 978-80-251-4296-7.

[24] DOMES, Martin. *333 tipů a triků pro CSS*. 2., aktualiz. vyd. Brno: Computer Press, 2011. ISBN 978-80-251-3366-8.

[25] Sass Basics. *Sass: Syntactically Awesome Style Sheets* [online]. [cit. 2018-05-12]. Dostupné z: https://sass-lang.com/guide

[26] GITHUB. *The State of the Octoverse 2017* [online]. 2017 [cit. 2018-05-09]. Dostupné z: https://octoverse.github.com/

[27] JavaScript Tutorial. *W3Schools Online Web Tutorials* [online]. [cit. 2018-05-17]. Dostupné z: https://www.w3schools.com/js/default.asp

[28] *TypeScript: JavaScript that scales* [online]. [cit. 2018-05-12]. Dostupné z: https://www.typescriptlang.org/

[29] Architecture overview. *Angular* [online]. [cit. 2018-05-13]. Dostupné z: https://angular.io/guide/architecture

[30] RABOY, Nic. Use SQLite In Ionic 2 Instead Of Local Storage. In: *The Polyglot Developer* [online]. 27.12.2015 [cit. 2018-05-11]. Dostupné z: https://www.thepolyglotdeveloper.com/2015/12/use-sqlite-in-ionic-2-instead-of-local-storage/

[31] *SQLite Home Page* [online]. [cit. 2018-05-12]. Dostupné z: https://www.sqlite.org/

[32] NGUYEN, Don. *Node.js Okamžitě*. Brno: Computer Press, 2016. ISBN 978-80-251-4820-4.

[33] What is MySQL? *MySQL* [online]. [cit. 2018-05-12]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

[34] JSON: Introduction. *W3Schools Online Web Tutorials* [online]. [cit. 2018-05-17]. Dostupné z: https://www.w3schools.com/js/js_json_intro.asp

[35] BROWN, Korbin. What Is GitHub, and What Is It Used For? In: *How To Geek* [online]. 21.9.2016 [cit. 2018-05-09]. Dostupné z: https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

[36] Getting Started: Git Basics. *Git* [online]. [cit. 2018-05-09]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

[37] Documentation for Visual Studio Code. *Visual Studio Code: Code Editing. Redefined* [online]. [cit. 2018-05-17]. Dostupné z: https://code.visualstudio.com/docs

[38] What is Prettier? *Prettier: Opinionated Code Formatter* [online]. [cit. 2018-05-12]. Dostupné z: https://prettier.io/docs/en/index.html

[39] *TSLint* [online]. 2016 [cit. 2018-05-12]. Dostupné z: https://palantir.github.io/tslint/

[40] MySQL Workbench. *MySQL* [online]. [cit. 2018-05-12]. Dostupné z: https://www.mysql.com/products/workbench/

[41] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[42] HOC, Phan. *Ionic 2 Cookbook: Second Edition*. Packt Publishing, 2016. ISBN 978-1-78646-596-2.

[43] MORONY, Josh. *Joshmorony: Build Mobile Apps with HTML5* [online]. [cit. 2018-05-15]. Dostupné z: https://www.joshmorony.com/blog/

PŘÍLOHY

PŘÍLOHA A – POROVNÁNÍ IMPLEMENTAČNÍCH METOD

	Nativní		Hybridní	Kompilované		
	iOS	**Android**	**Ionic + Cordova**	**React Native**	**NativeScript**	**Xamarin**
Výhody	Nejvyšší výkon Podpora vývoje Nativní vzhled Přístup k funkcím zařízení		Přenositelné Webové technologie Webové knihovny	Přenositelné Jednotné technologie Lepší výkon		
		Nezávislost vývoje na OS	Nativní vzhled Množství nativních pluginů a přístup k nim	JS knihovny Podpora Reactu	JS knihovny	Rychlost Provázanost s nativními technologiemi
Nevýhody	Nepřenositelné aplikace		Pomalejší Závislost na technologii třetí strany	Závislost na technologii třetí strany		
	Vývoj jen na MAC OS X	Fragmentace vývoje		Interpretace JS	Interpretace JS Nižší podpora Menší komunita	
Distribuce aplikace	Nativní obchody aplikací		Nativní obchody aplikací Webové aplikace	Nativní obchody aplikací		
Technologie vývoje	Objective C nebo Swift	Java nebo Kotlin	HTML, CSS, JavaScript nebo TypeScript, Angular	JavaScript, JSX, React	JavaScript, XML	C#