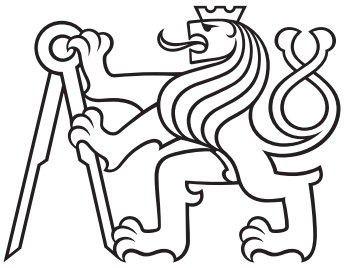
České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky
Obor: Kybernetika a robotika



Multiplatformní interaktivní objednávkový systém pro kavárenský sektor

BAKALÁŘSKÁ PRÁCE

Vypracoval: Viktor Valík
Vedoucí práce: Ing. Jiří Šebek
Rok: 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Valík** Jméno: **Viktor** Osobní číslo: **492350**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Multiplatformní interaktivní objednávkový systém pro kavárenský sektor

Název bakalářské práce anglicky:

Multiplatform interactive order processing system for coffee shop sector

Pokyny pro vypracování:

1) Seznamte se s problematikou objednávání produktů skrze mobilní aplikaci a s problematikou predikce poptávky.
2) Navrhněte řešení, jak propojit síť zákazníků se sítí kaváren skrze jednu aplikaci, která bude multiplatformní - bude fungovat na platformě iOS i Android na mobilu i tabletu. Navrhněte informační systém s predikcí poptávky.
3) Vytvořte MVP, který bude zákazníkovi prezentovat produkty a kavárny a umožní mu tvorbu objednávky na základě preferencí. Dále bude prodejci umožňovat zpracování příchozích objednávek a komunikaci s centrálou.
4) Otestujte aplikaci z hlediska softwarového návrhu a implementace (stabilita, škálování). Změřte její přínos. (Uživatelské testování)

Seznam doporučené literatury:

[1] Flutter architectural overview. Flutter [online]. Dublin: Google [cit. 2022-01-14]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview
[2] Going global at Google Pay with Flutter. Flutter [online]. Dublin: Google [cit. 2022-01-14]. Dostupné z: https://flutter.dev/showcase/google-pay
[3] Learn Firebase fundamentals. Firebase [online]. San Francisco: Google [cit. 2022-01-14]. Dostupné z: https://firebase.google.com/docs/guides
[4] DELIA, Lisandro, et al. Multi-platform mobile application development analysis. In: 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS). IEEE, 2015. p. 181-186.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek kabinet výuky informatiky FEL

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

Datum zadání bakalářské práce: **12.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce:
do konce letního semestru 2022/2023

Ing. Jiří Šebek	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne .....

.....
Viktor Valík

Poděkování

Děkuji Ing. Jiřímu Šebkovi za vedení mé bakalářské práce a za cenné rady, které ji obohatily.

Viktor Valík

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací systému, který dokáže propojit síť zákazníků se sítí kaváren. Účelem práce je implementovat systém jako mobilní aplikaci, která bude fungovat v mobilu i tabletu a v rámci operačních systémů iOS i Android.

V teoretické části se čtenář seznámí s teorií týkající se zkoumané problematiky. Následně je proveden rozbor požadavků a konkurenčních systémů spolu s analýzou nástrojů a platforem umožňujících implementaci systému.

Praktická část je zaměřena na návrh mobilní aplikace s použitím rozhraní Flutter, platformy Firebase a platební brány PayU. Dále je předložen postup implementace a možnost integrace predikce poptávky do aplikace. Na konci práce jsou uvedeny výsledky testování.

Klíčová slova: multiplatformní, mobilní aplikace, iOS, Android, HCI, Flutter, Firebase, predikce

Abstract

This thesis deals with the analysis, design and implementation of a system that can connect a network of customers with a network of cafes. The purpose of the thesis is to implement the system as a mobile application that will work on mobile and tablet and within the iOS and Android operating systems.

In the theoretical part, the reader is introduced to the theory related to this issue. Subsequently, an analysis of the requirements and competing systems is made, along with an analysis of the tools and platforms enabling the implementation of the system.

The practical part focuses on the design of a mobile application using the Flutter interface, the Firebase platform and the PayU payment gateway. Furthermore, the implementation procedure and the possibility of integrating demand forecasting into the application are presented. At the end of the paper, testing results are presented.

Keywords: multiplatform, mobile application, iOS, Android, HCI, Flutter, Firebase, prediction

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato práce se zabývá analýzou a návrhem systému, který dokáže propojit síť zákazníků se sítí kaváren, a jejím účelem je implementace systému coby multiplatformní mobilní aplikace. Ze zadání vyplývá, že bude fungovat v mobilu i tabletu v rámci operačních systémů iOS i Android.

V úvodní kapitole je představena motivace vývoje systému a předložena struktura bakalářské práce.

1.1 Motivace

V dnešní době existuje mnoho služeb, které zajišťují propojení zákazníka s prodejcem přes mobilní aplikaci. Přestože se omezíme na gastronomický sektor, existuje stále nespočet zprostředkovacích služeb. Nevýhodná bývá situace, kdy má daná služba mnoho partnerství s různými provozovnami. Na straně zákazníka potom často nastává paralýza volby, jelikož je mu prezentováno příliš mnoho produktů a prodejen.

V takovém případě se však zároveň otevírá prostor na trhu pro menší platformy, které by dokázaly dokonale ovládnout jeden menší podsektor a usnadnit zákazníkovi volbu produktu, ušetřit čas při nákupu a celkově zefektivnit fungování daného gastronomického řetězce.

Zprostředkovací služba popsaná v této bakalářské práci se konkrétně zaměřuje na kavárenský sektor. Oproti konkurenčním službám [1] bude fungovat poněkud odlišně. Nebude zde figurovat kurýr pro rozvoz objednávek. Když si zákazník bude chtít objednat produkt přes aplikaci, systém mu nabídne nejbližší provozovny. Zákazník si vybere čas vyzvednutí a provede platbu. Zákazník si poté může objednávku bez čekání a bez starostí v provozovně vyzvednout.

Cílem projektu je v podstatě vytvořit ekosystém, ve kterém se zákazník bude cítit dobře a bude jej pravidelně využívat. Každý podnik, který se do ekosystému připojí, získá novou síť klientů a navíc nástroje poskytované aplikací.

1.2 Struktura práce

Teoretická část zahrnuje rešerši a analýzu, která je následně převedena do návrhu aplikace. Nejprve se zaměříme na sběr požadavků, které vyplývají ze zadání a z teorie HCI (Human Computer Interaction). K vysvětlení pojmů jsou použity

zejména zdroje [2, 3, 4, 5, 6]. Následuje vysvětlení teorie týkající se predikce poptávky na základě [7, 8] a průzkum konkurenčních systémů z hlediska jejich architektury a mikroservisů. Nakonec je v této části provedena analýza nástrojů a platforem umožňujících implementaci systému podobného charakteru.

Praktická část je zaměřena na návrh datového modelu a uživatelského rozhraní společně s návrhem predikce poptávky a jeho způsobem integrace do mobilní aplikace. Popsán je také postup implementace mobilní aplikace. Nakonec je čtenář seznámen s výsledky testování.

Kapitola 2

Teoretický rozbor

První část této kapitoly bude zaměřena na teorii týkající se tvorby uživatelských rozhraní. Vysvětlíme základní principy HCI a faktory použitelnosti systému, které nám pomohou vytvořit lepší návrh. Ve druhé části kapitoly předložíme teoretické poznatky k problematice predikce poptávky. Souhrn potom propojuje teoretickou a praktickou část práce.

2.1 Interakce mezi člověkem a počítačem

Human Computer Interaction (HCI) je interdisciplinární obor, který řeší výzkum, plánování a návrh interakce mezi počítači a lidmi (uživateli). HCI je koncept, který by měl vytvářet vhodné propojení uživatele, stroje a služeb, aby bylo dosaženo požadované výkonnosti jak v kvalitě, tak v optimálnosti služeb. Posouzení kvality konceptu HCI je většinou subjektivní a závisí na kontextu [2].

Pojem HCI byl představen společně s příchodem počítačů, nebo obecněji strojů jako takových. Důvod je jasný, většina sofistikovaných strojů je bezcenných, pokud je člověk nemůže správně a efektivně používat. Tento základní argument vede ke dvěma hlavním pilířům, které je třeba zohlednit při návrhu HCI: funkčnost a použitelnost [3].

Funkčnost systému je definována souborem činností nebo služeb, které systém poskytuje svým uživatelům. Hodnota, kterou funkčnost přináší je však viditelná teprve tehdy, když ji uživatel může efektivně využít [2].

Použitelnost systému s určitou funkčností je rozsah, v jakém lze systém efektivně a adekvátně využívat k dosažení určitých cílů pro určité uživatele. Skutečné efektivity je dosaženo právě tehdy, když existuje rovnováha mezi funkčností a použitelností systému [2].

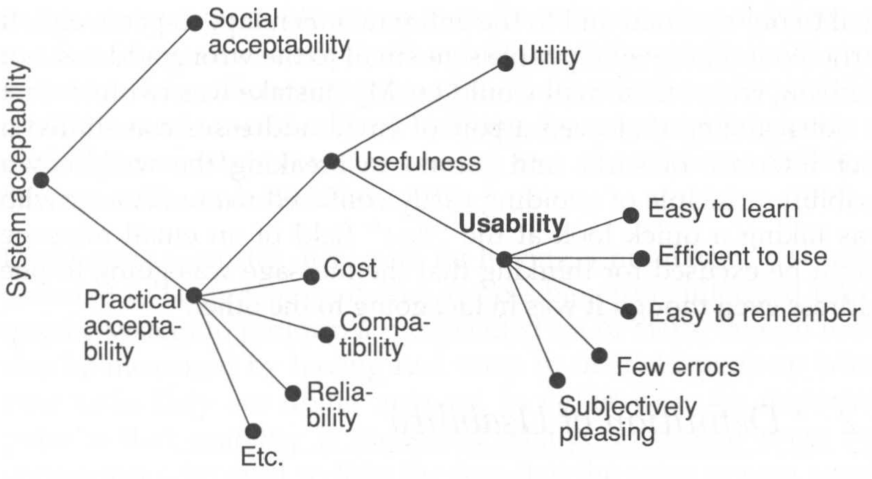
Interakce člověka a počítače zasahuje do mnoha vědních disciplín a lze na ni nahlížet z různých úhlů. Z pohledu uživatele se může jednat o faktory, jako například spokojenost, estetika, komunikace, kognitivní vjemy apod. Ze strany počítače se můžeme na návrh HCI dívat jako na informatickou úlohu (programovací platforma, programovací jazyk apod.). Při návrhu HCI bychom měli mít na paměti, že kontext úlohy je poměrně dynamický a proměnlivý, ať už se jedná o změny trendů a vzorců chování ze strany člověka nebo o rychlý vývoj technologií ve světě počítačů [4].

2.2 Použitelnost systému podle Jakoba Nielsena

Jakob Nielsen je světově uznávaný odborník v oblasti HCI a díky své práci v 90. letech se řadí mezi hlavní průkopníky v tehdy ještě poměrně nové oblasti použitelnosti počítačových systémů a uživatelských rozhraní [5].

Použitelnost je užší problém v rámci širšího tématu *přijatelnosti* systému (system acceptability, obrázek 2.1). Jedná se v podstatě o otázku, zda je systém dostatečně kvalitní, aby uspokojil všechny potřeby a požadavky uživatelů a dalších zainteresovaných stran. Celková přijatelnost počítačového systému je kombinací jeho společenské a praktické přijatelnosti (social and practical acceptability). Na základě skutečnosti, že je systém společensky přijatelný, lze dále analyzovat jeho praktickou přijatelnost v rámci různých kategorií, včetně těch tradičních, jako jsou náklady (cost), spolehlivost (reliability), kompatibilita (compatibility) se stávajícími systémy atd. [5].

Můžeme také zkoumat kategorii užitečnosti (usefulness). Jedná se o faktor, který definuje, zda lze systém použít k dosažení požadovaného cíle. Opět jej lze rozdělit na dvě kategorie: funkčnost (utility) a použitelnost (usability). Funkčnost udává, zda systém dělá to, co je potřeba, a použitelnost vyjadřuje, jak dobře mohou uživatelé funkční prvky používat. Všimněte si, že pojem *užitečnost* nemusí být nutně omezen na oblast určitého typu práce. Např. vzdělávací software (courseware) má vysokou míru užitečnosti, pokud se mohou studenti s jeho pomocí efektivně učit. Zábavní produkt má vysokou užitečnost, pokud je jeho užívání zábavné [5].



Obrázek 2.1: Model přijatelnosti systému [5]

Jak vyplývá z obrázku, přijatelnost systému má několik složek a použitelnost musí být ve vyvíjeném projektu kompromisem mnoha dalších hledisek. Na základě teorie Jakoba Nielsena je použitelnost charakterizována pěti vlastnostmi [5]:

- naučitelnost (easy to learn)
- efektivita (easy to use)
- zapamatovatelnost (easy to remember)
- malá chybovost (few error)
- uspokojení (subjectively pleasing)

2.2.1 Naučitelnost

Tento pojem znamená, že by mělo být pro uživatele jednoduché se naučit systém používat tak, aby s ním mohl rychle začít pracovat. Naučitelnost je v určitém smyslu nejzákladnějším atributem použitelnosti. Většina systémů musí být snadno naučitelná, protože první zkušenost většiny lidí s novým systémem je právě proces učení. Existují systémy, u nichž lze uživatele intenzivně školit, aby složité rozhraní pochopili, ale ve většině případů je třeba, aby osvojení systému bylo jednoduché [5].

V případě aplikace, na niž je naše bakalářská práce zaměřena, se například jedná o rozdíl mezi zákazníkem a prodejcem. Uživatelské rozhraní pro zákazníka musí být velmi jednoduché a stručné, jelikož na zachycení jeho pozornosti míváme zpravidla málo času, tudíž musíme klást důraz na první dojem. Pokud jde o představení systému prodejci, může například proběhnout diskuze s vývojářem. Prodejce bude pravděpodobně ochoten se učit pracovat se systémem o něco déle, protože si je předem vědom toho, jaké benefity mu to do budoucna přinese.

2.2.2 Efektivita

Význam tohoto pojmu spočívá v tom, že by měl systém být při používání efektivní v tom smyslu, aby uživatel po jeho osvojení dosáhl co nejvyšší produktivity. Jedná se o vlastnost poněkud hůře testovatelnou a měřitelnou, jelikož je potřeba testovat uživatele, kteří mají se systémem dlouhodobou zkušenost. Testovat lze například čas, který je potřebný k vykonání určitého úkonu [5].

2.2.3 Zapamatovatelnost

Systém by měl být snadno zapamatovatelný, aby se k němu mohl vrátit i běžný uživatel po určité době nepoužívání, aniž by se musel vše učit znovu. Tato vlastnost systému je důležitá pro tzv. příležitostné uživatelé, kteří představují třetí hlavní kategorií vedle začínajících a zkušených. Příležitostní uživatelé jsou lidé, kteří nepoužívají systém až tak často, jak se předpokládá u uživatelů expertních. Na rozdíl od těch začínajících však příležitostní uživatelé systém používali již dříve, tím pádem se jej nemusí učit od začátku. Stačí, když si na základě předchozích zkušeností na něj vzpomenou. Navíc by systém měl být navržený tak, aby navedl uživatele k cíli, aniž by si byl jistý přesným postupem. Testování zapamatovatelnosti může probíhat tak, že zadáme určité úkoly skupině příležitostných uživatelů a skupině začátečníků a sledujeme, jak rychle dokážou úkoly splnit [5].

2.2.4 Chybovost

Systém by měl být navržen tak, aby uživatelé při používání systému dělali jen málo chyb, případně je mohli snadno napravit v případě, že se jich dopustí. Míru chybovosti lze měřit v rámci experimentů, které slouží k měření dalších atributů použitelnosti. Nesmí také docházet ke katastrofickým chybám, které vedou ke zmaření celé práce uživatele a které je obtížné napravit. Katastrofické chyby by měly být vyhodnocovány odděleně od těch drobných a mělo by být vyvinuto zvláštní úsilí k minimalizaci jejich četnosti [5].

2.2.5 Uspokojení

Systém by měl působit příjemně, aby uživatelé byli při jeho užívání subjektivně spokojení a aby se jim líbil. Tato vlastnost se dá měřit pomocí dotazníků [5]. Míra uspokojení se dá také sledovat na základě recenzí, jedná-li se o aplikaci, která je distribuovaná přes internetový obchod, jako například App Store nebo Google Play.

2.3 Použitelnost systému podle Steva Kruga

Velmi respektovanou osobností v oblasti HCI, použitelnosti systémů a uživatelského prožitku je Steve Krug. Jeho nejznámějším dílem je publikace s názvem *Nenuťte uživatele přemýšlet! (Don't Make Me Think!)* [6], jehož se na celém světě prodalo přes 300 tisíc výtisků. Oproti Jakobu Nielsenovi, který definuje pojmy spíše formálně, se Krug pokouší své zkušenosti předat jednodušší a stručnější formou, aby problematiku pochopil úplně každý. Skutečnost, že cílí na laické publikum, dokazuje jeho definice použitelnosti z knihy uvedené výše: *„Použitelnost přece znamená, že něco dobře funguje a že osoba s průměrnými (ba dokonce podprůměrnými) schopnostmi a zkušenostmi může používat určitou věc – ať už se jedná o webovou stránku, bojový stíhací letoun nebo otočné dveře – k účelu, ke kterému je určena, aniž by musela být beznadějně mučena.“* [6].

Pokusme se nyní shrnout základní poznatky Steva Kruga k problematice použitelnosti.

2.3.1 Nenutit uživatele přemýšlet

Základním principem při tvorbě uživatelských rozhraní je jednoduchost a stručnost. Každá obrazovka by měla být intuitivně rozvržena a na první pohled vystihovat podstatu sdělení. Uživateli by měl být jednoznačně jasný obsah stránek a měl by být schopen systém používat bez většího úsilí. Princip je dobře vidět u klikatelných tlačítek, pomocí kterých se uživatel může v systému pohybovat. Musí být jasně vymezeno, které prvky obrazovky mají funkci tlačítek a které ne, také musí být z názvu tlačítka patrné, kam nás nasměruje. Každá nejasnost, která vyvolává v uživateli otázky namísto pokroku v řešení problému, zbytečně odvádí pozornost. Tato drobná rozptýlení mohou rychle kumulovat až do té míry, že frustrovaný uživatel systém opustí [6].

2.3.2 Uživatelé stránky nečtou, ale prohlížejí

Při tvorbě systému se může zdát jako efektivní postup poskytnout uživateli co nejvíce informací, aby se mohl pečlivě rozhodnout, než klikne na odkaz či tlačítko. Uživatelé však nepostupují systematicky, nýbrž letmo prohlédnou text, a když spatří tlačítko, které by mohlo splnit jejich cíl, neváhají kliknout. Design obrazovky by měl tedy být spíše minimalistický a směřovat rovnou k věci [6].

2.3.3 Vynechat nadbytečná slova

Užívání zdlouhavých popisů a velkého množství slov působí na uživatele demotivačně, jelikož má pocit, že musí přečíst velký objem textu, aby se dostal k cíli. Tento fakt souvisí s předchozím odstavcem, který tvrdí, že uživatelé stránky spíše

prohlížejí, takže je lepší redukovat míru šumu, kterou může rozsáhlý popis zvyšovat. Podle Steva Kruga je možné při revizi návrhu na každé stránce redukovat text na polovinu, aniž by se ztratila významová hodnota [6].

2.3.4 Uživatel musí vědět, kde se právě nachází

Při navigaci v systému je důležité, aby uživatel chápal strukturu systému. Míra větvení a zanoření obrazovek není natolik podstatná v případě, kdy uživatel nemusí přemýšlet a má jasnou představu, kde se nachází. Uživatel se v systému cítí komfortně jen tehdy, pokud je hierarchie intuitivní a systém obsahuje chytré navigační prvky [6].

2.4 Predikce poptávky

Predikce poptávky je poměrně známý problém, který je třeba řešit takřka v jakémkoliv systému, v němž figuruje nabídka a poptávka. Existuje několik způsobů, jak z historických dat extrapolovat budoucnost. Kapitola vychází ze zdrojů [7] a [8].

Prvním způsobem je tzv. naivní přístup. Předpokládá, že poptávka bude v následujícím období stejná jako dříve. To je velmi statický způsob predikce a už z jeho podstaty je jasné, že vede k velkým odchylkám. Dalším přístupem jsou klasické statistické metody jako autoregrese (AR) a klouzavý průměr (MA) nebo jejich kombinace (ARMA, ARIMA). Tyto metody jsou známé, vcelku jednoduché a dokážou poskytnout určitou míru vhledu.

Nejmodernějším přístupem je použití neuronových sítí, konkrétně rekurentních neuronových sítí (RNN). Hlavním přínosem RNN oproti klasickým metodám je možnost dynamicky aktualizovat parametry, učit se a pamatovat si předchozí stavy. RNN mají jeden nedostatek, a to krátkodobou paměť. Pokud je zpracovávaná sekvence příliš dlouhá, má RNN problém přenést informace z dřívějších časových úseků do pozdějších. Pokud je tedy například úkolem zpracovat odstavec textu a provést předpověď, může RNN vynechat důležité informace a myšlenky ze začátku. Hlavní důvod, který tento jev způsobuje, je problém mizejícího gradientu při jeho zpětné propagaci. Gradienty jsou hodnoty užívané k aktualizaci vah neuronových sítí. Problém mizejícího gradientu nastává, když se gradient při zpětné propagaci v čase zmenšuje. Pokud se hodnota gradientu zmenší na extrémně malou hodnotu, k učení příliš nepřispívá. V RNN se tedy dřívější vrstvy, které dostanou malou aktualizaci gradientu, přestanou učit, a tím pádem může v delších sekvencích RNN zapomenout, co viděla.

Tento problém řeší síť LSTM (Long-Short Term Memory). Je vylepšena o vnitřní mechanismy, takzvané *brány*, které mohou regulovat tok informací. Podrobněji se jedná o vstupní, výstupní a zapomínající bránu. Mohou se naučit, která data v sekvenci je důležité zachovat a která je vhodné zapomenout. Díky tomu může síť LSTM předávat relevantní informace dále v dlouhém řetězci sekvencí a vytvářet tak smysluplné předpovědi.

2.5 Shrnutí rozboru

V teoretické části byly vysvětleny základní principy HCI a známé faktory použitelnosti systému. Podklady z této kapitoly využijeme zejména při návrhu uživa-

telského rozhraní a při jeho následném testování.

Jelikož je jedním z požadavků na tuto práci vytvořit návrh informačního systému s predikcí poptávky, byla v této kapitole také předložena teorie týkající se předpovědi časových řad. Toto téma je poměrně komplexní a bude mu dále věnována kapitola 4.3, která zmíněné teoretické postupy ukáže na příkladu existujícího systému.

Kapitola 3

Sběr požadavků

V této kapitole je provedena analýza zadání, na jejímž základě budou definovány potřeby uživatele. Z těchto potřeb vyplyne soubor požadavků, které bychom měli zohlednit při návrhu aplikace.

3.1 Analýza zadání

Na základě nároků obsažených v zadání je zřejmé, že do systému bude potřeba zahrnout celkem čtyři entity, z nichž tři základní jsou zákazník, prodejce a centrála. Podle toho bude postavena konstrukce funkčních požadavků. Pro účely další pohodlnější analýzy a pozdějšího návrhu si přejmenujeme roli prodejce na pracovníka a roli centrály na administrátora. Role zákazníka je poměrně jasná, bude to nezávislá entita, která tvoří poptávku po produktu. U dalších dvou rolí je třeba se zamyslet nad principem fungování kavárenské společnosti. Jedná se většinou o organizaci, která spravuje několik provozoven. V rámci jedné společnosti bude tedy jeden centrální uzel v roli administrátora a několik závislých uzlů v rolích pracovníků.

Čtvrtou entitou je platební brána. Tu na první pohled není tak jednoduché dopředu identifikovat, protože její vliv na systém není tak explicitní jako u ostatních entit. Každý obchodník musí své transakce provádět skrze platební bránu, protože sám o sobě nemá oprávnění manipulovat s bankovními údaji zákazníka (pokud sám platební bránu neprovozuje).

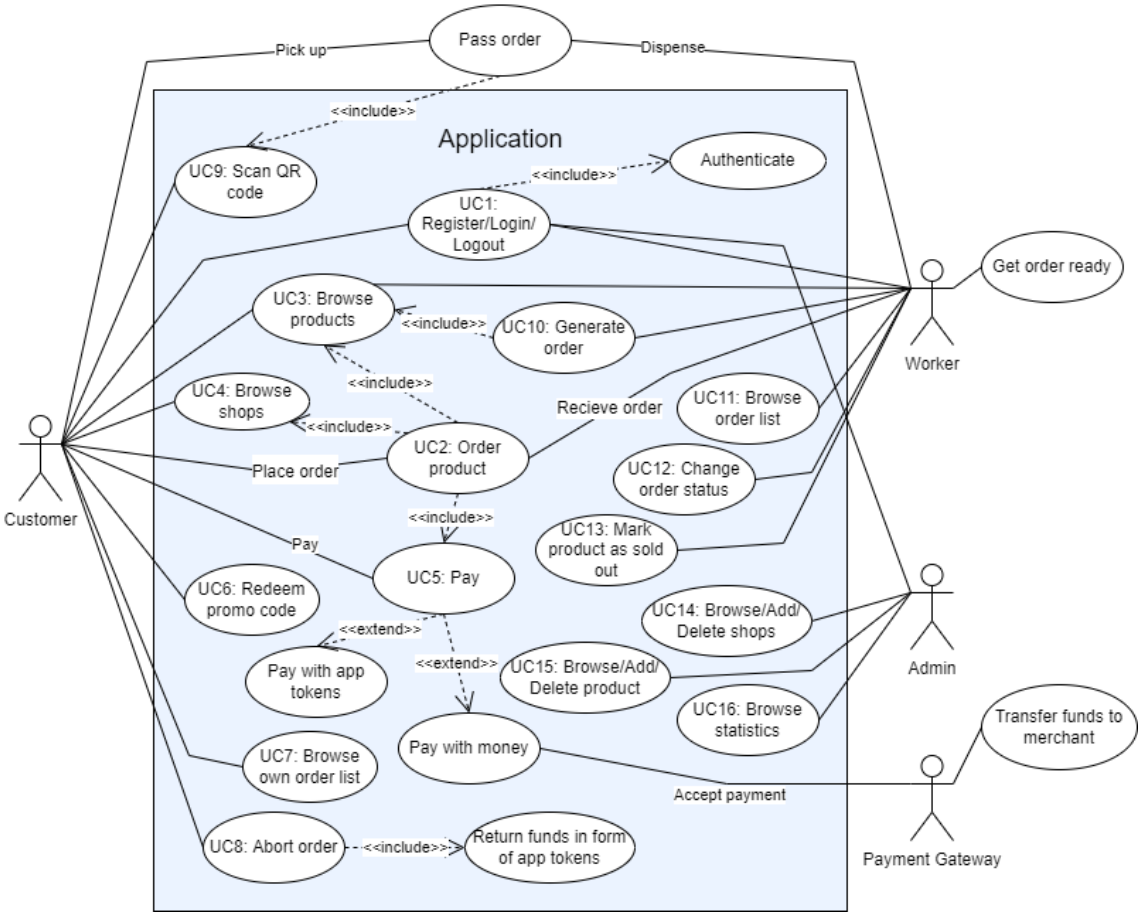
Platební brána je speciální aplikace, která umožňuje rychlé, bezpečné a jednoduché placení na internetu, je licencovaná centrální bankou a řídí se jejími regulacemi. Platební brána by měla splňovat příslušné bezpečnostní standardy a využívat moderní bezpečnostní systémy pro platby kartou [9].

V zadání se objevují také nefunkční požadavky, kterými jsou multiplatformnost a responzivita.

3.2 Funkční požadavky

V předchozí sekci jsme identifikovali několik entit. Každá z nich má odlišnou roli, a tudíž i specifické nároky na funkčnost aplikace. Proto bude potřeba identifikovat případy užití aplikace z několika různých pohledů. Nejprve se zaměříme na společné požadavky všech uživatelů aplikace a následně na potřeby a požadavky zákazníka, pracovníka a administrátora. Entita, která představuje platební bránu, v podstatě žádné požadavky nemá, využíváme ji pouze jako službu třetí strany. Obrázek 3.1

slouží jako grafický podklad pro vysvětlení vztahů mezi entitami (a s nimi spojenými případy užití aplikace).



Obrázek 3.1: Diagram zobrazující případy použití aplikace (UC = Use Case)

3.2.1 Společné požadavky

Funkce, která bude všem entitám společná, je možnost registrace, přihlášení a odhlášení. Samotný proces registrace se bude pravděpodobně trochu lišit, protože každá entita jakožto třída bude mít jiné atributy.

3.2.2 Požadavky zákazníka

Hlavním požadavkem na zákaznickou část aplikace je možnost rychlého a jednoduchého vytvoření a zaplacení objednávky. Tomu předchází potřeba prohlédnout si dostupné provozovny a jejich produkty. V souladu s teorií z kapitol 2.2 a 2.3 by měly být funkce splňující tyto potřeby prezentovány stručně a jasně. Proto se nabízí realizovat přehled provozoven jako mapu. Ta by však byla poměrně složitá na implementaci, a proto zvolíme spíše představení pomocí seznamu. U nabídky produktů nejlépe identifikujeme preference zákazníka vizuálně pomocí obrázků. Nakonec si bude moci zvolit, za jak dlouho si objednávku vyzvedne na provozovně.

Proces objednávky je také nutně spojen s platbou, která by měla proběhnout co nejjednodušeji, nejlépe jedním tlačítkem, aby zákazník neměl tendenci koupi zvažovat. Kromě platby penězi bude vhodné zavést i možnost platby interní aplikační měnou (kredity). Tuto funkci zavádíme hlavně z důvodu zjednodušení platebního

systému. Uživatel by totiž měl mít možnost zrušit objednávku, za což by se mu slušelo vrátit peníze. Refundace je ovšem nemalý implementační úkol, který zahrnuje správnou komunikaci s platební bránou a manipulaci s penězi, což může do celého systému zavést zbytečné chyby. Máme-li zavedené kredity, můžeme je při zrušení objednávky jednoduše přičíst zákazníkovi na účet. Ten si pak za ně může koupit následující objednávku. Kredity se také hodí pro zavedení promo kódů, které by zákazníkovi odemykaly přístup k dalším kreditům.

Na závěr objednávky dochází k jejímu převzetí (fyzický akt probíhající mimo aplikaci, jak je naznačeno na obrázku 3.1), při němž je potřeba prokázat oprávnění k vyzvednutí objednávky. Toho se dá velmi dobře dosáhnout pomocí skenování QR kódu.

Sadu funkcí bude završovat možnost prohlédnout si své aktivní objednávky a historii objednávek. Historické objednávky mohou posloužit jako stvrzenky za provedené nákupy.

3.2.3 Požadavky administrátora

Hlavní prioritou administrátorské části je sloužit jako centrála, což zahrnuje potřebu přístupu k přehledu svých prodejen a produktů nebo přístup ke statistickým údajům a predikcím.

První obrazovka, kterou bude administrátor potřebovat, je přehled prodejen. Zde bude možné přidat a odebrat provozovny nebo upravit jejich údaje. Obdobně bude vypadat druhá obrazovka zaměřená na produkty. Třetí obrazovka bude podávat shrnující informace týkající se objednávek a provozoven. Mohou se zde vyskytnout například informace, jako je počet zrušených objednávek nebo počet a druh prodaných produktů s predikcí budoucí poptávky.

3.2.4 Požadavky prodejce

Hlavní potřebou prodejce je možnost pohodlně a rychle vyřizovat objednávky. Obrazovka, na které bude prodejce pravděpodobně trávit nejvíce času, je tudíž fronta objednávek. Pro zvýšení přehlednosti by se měly urgentní objednávky zobrazovat jako první, dále by zde měla být možnost filtrování objednávek podle toho, v jakém jsou stavu. Každou objednávku by mělo být možné rozkliknout k zobrazení podrobných informací důležitých k přípravě (ta samozřejmě probíhá mimo aplikaci, jak je naznačeno na obrázku 3.1). Vhodná funkce s tímto spojená bude možnost označit objednávku jako připravenou, čímž je zákazník informován, že si objednávku může převzít.

Prodejců může být více, přičemž každý musí být schopen si zobrazit frontu objednávek korespondující s provozovnou, ve které právě pracuje. Zároveň by však bylo poměrně nepohodlné, aby každý prodejce měl v aplikaci vlastní účet, jelikož s každou novou provozovnou by administrátor musel zakládat nový prodejní účet. Musí tedy fungovat pouze jeden účet, skrze který bude moci vyřizovat objednávky několik prodejců naráz. Ke zmíněné problematice se blíže vyjadřuje kapitola 7 zabývající se návrhem.

Jedna funkce, která nevyplývá přímo z naší analýzy zadání, je možnost generování objednávek prodejcem. Mohla by být užitečná zejména za předpokladu, že je naše aplikace jediným systémem, skrze který provozovna vyřizuje objednávky. Ne

každý uživatel bude pravděpodobně chtít vytvářet objednávky vzdáleně a mohl by si přát vytvořit objednávku na místě.

V provozovně může nastat případ, že dojdou suroviny nebo se pokazí zařízení nutné k přípravě objednávky. V tom případě bude mít prodejce možnost v aplikaci deaktivovat produkt vyžadující tyto suroviny, což se projeví v aplikaci zákazníka, který si nebude moci daný produkt objednat; a v aplikaci administrátora, který uvidí chybějící produkt nebo suroviny a může problém vyřešit.

3.3 Nefunkční požadavky

Nefunkční požadavky jsou takové, které nepředstavují konkrétní uživatelské funkce. Jde spíše o vlastnosti samotného systému. Ze zadání víme, že systém musí být multiplatformní a plně responzivní. Oba tyto požadavky bude potřeba testovat.

3.3.1 Multiplatformnost

Ze softwarového hlediska potřebujeme zajistit, aby aplikace fungovala v operačních systémech Android a iOS. Způsoby, které tento nárok řeší, se bude zabývat kapitola 5 zaměřená na analýzu nástrojů.

3.3.2 Responzivita

Responzivita v tomto případě znamená schopnost reagovat na změnu velikosti displeje a adekvátně přizpůsobit velikost prvků uživatelského rozhraní. Ke splnění tohoto požadavku je třeba zaručit, aby byla aplikace použitelná na obrazovkách o co největším spektru velikostí. Tohoto problému se dotkne kapitola 8.1 zabývající se responzivitou.

Kapitola 4

Analýza podobných aplikací

Tato kapitola bude zacílena na průzkum konkurenčních aplikací z hlediska mikroservisů, architektury a predikce poptávky. V jejím závěru je přiloženo shrnutí, vysvětlující přínosy analýzy a její spojitost s touto prací. Jelikož nebyly nalezeny společnosti, které provozují aplikace podobné té naší a byly by zároveň transparentní ohledně své architektury, zaměříme se na co možná nejpodobnější společnost – Uber.

4.1 Mikroservisy společnosti Uber

Služby Uberu jsou poměrně známé. Uživatel si může prostřednictvím aplikace objednat jízdu a během několika minut přijede poblíž čekající řidič, který ho odveze do cíle. Dříve byl Uber postaven na modelu „monolitické“ softwarové architektury. Měla backendovou i frontendovou službu a jedinou databázi. Architektura fungovala dobře pro malý počet jízd v několika městech, když se ale služba rozšířila do dalších metropolí, potýkal se tým Uberu s problémy. Po roce 2014 se jeho členové rozhodli přejít na servisně orientovanou architekturu a nyní Uber kromě dopravy osob zajišťuje také rozvoz jídla [10].

Architektura služby Uber se skládá z mnoha mikroservisů (příloha 1). Nabídku zajišťují řidiči (drivers) a poptávku představují zákazníci (riders). Všechna volání, která tyto dvě entity provedou, musí nejdříve projít přes Web Application Firewall (WAF), která slouží k blokování určitých IP adres a botů. Následně prochází přes Load Balancer (LB), který rozděluje zátěž mezi jednotlivé servery. KAFKA REST API [11] zpracovává data o polohách řidičů, DISCO (Dispatch Optimization) zajišťuje propojení na základě zeměpisné polohy. V systému figuruje mnoho dalších servisů, jako například zabezpečení proti platebním podvodům nebo analytika. Všechny servisy spolupracují a jsou důležité. Výhodou podobné architektury je možnost aktualizace jednotlivých servisů bez nutnosti opětovného nasazování celého systému [12].

4.2 Databáze společnosti Uber

Při návrhu databáze Uber zvažoval následující kritéria [12]:

- Horizontální škálovatelnost – lze přidávat více uzlů v různých regionech a dohromady se pak chovají jako jedna databáze.

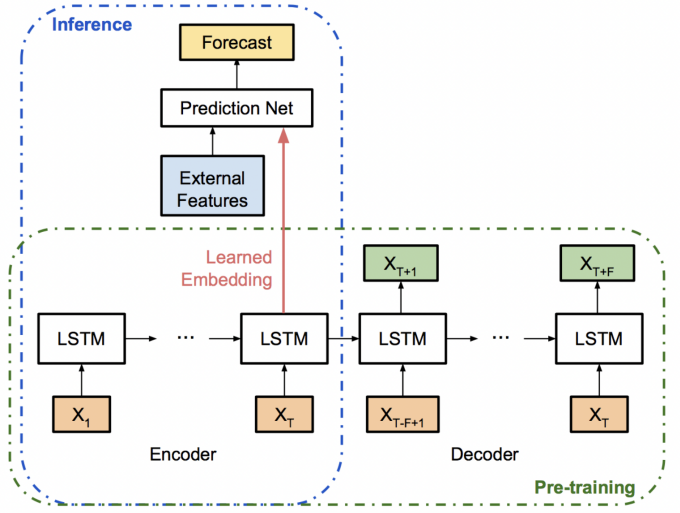
- Dostupnost zápisů a čtení – každé čtyři sekundy odesílá řidič svou polohu, takže se v systému odehrává velké množství čtení a zápisů.
- Žádné prostoje – systém by měl být dostupný i v době, kdy se provádí údržba.
- Nejbližší datová centra – když se přidají nová města, Uber se snaží ukládat data do nejbližších datových center, aby služba fungovala co nejrychleji.

Dříve byla používána k ukládání dat souvisejících s profily, body GPS a dalšími údaji tradiční relační databáze RDBMS (Relational Database Management System). Bylo však zjištěno, že systém nelze škálovat, když přibývají uživatelé a města. Uber tedy přešel na databázi na principu NoSQL (nebo tzv. schemaless), která je postavená na MYSQL [12].

4.3 Predikce poptávky společnosti Uber

Predikce poptávky se řadí mezi problémy predikce časových řad. Přesné předpovídání a spolehlivý odhad nejistoty jsou zásadní pro detekci anomálií, optimální alokaci zdrojů, plánování rozpočtu atd. Předpověď je náročná zejména v období vysoké variability (např. svátky, sportovní události). Tyto proměnné je velmi obtížné zahrnout do klasických modelů časových řad. Uber proto používá modelování založené na modelu LSTM, který byl zmíněn v kapitole 2.4. Jedná se o poměrně populární model díky komplexnímu modelování a snadnému začlenění exogenních proměnných. Síť LSTM také může modelovat složité nelineární interakce některých rysů, což je rozhodující pro modelování složitých extrémních událostí [13].

U LSTM se objevuje problém odhadu nejistoty předpovědi, která je důležitá pro posouzení toho, nakolik lze důvěřovat předpovědi vytvořené pomocí sítě. Uber používá Bayesovskou neuronovou síť (BNN), která se řídí podmíněnou pravděpodobností namísto bodového odhadu [13]. Zjednodušená architektura sítě je naznačena na diagramu na obrázku 4.1.

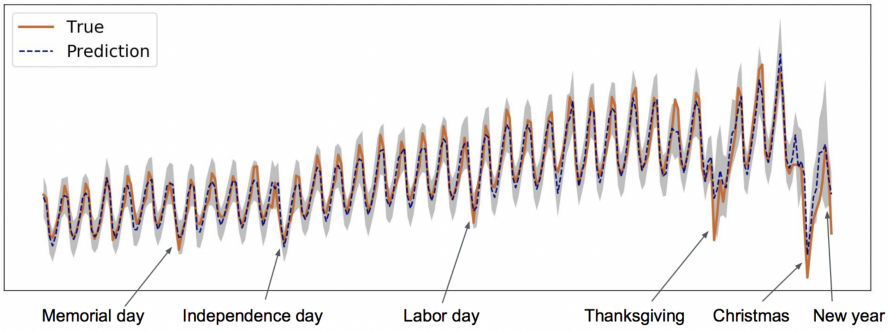


Obrázek 4.1: Architektura modelu společnosti Uber pro předpověď poptávky [13]

Výsledky fungování tohoto modelu jsou patrné z tabulky 4.1, která porovnává naivní přístup (Last-Day), quantile random forest (QRF), LSTM a bayesovský model (Our Model). Hodnoty v tabulce představují symetrickou absolutní střední procentuální odchylku (Symmetric mean absolute percentage error, SMAPE), která je výsledkem evaluace modelů na testovacích datech. Obrázek 4.2 pak ukazuje porovnání skutečných hodnot a hodnot předpovědi naměřených během testovacího období v San Franciscu [13].

City	Last-Day	QRF	LSTM	Our Model
Atlanta	15.9	13.2	11.0	7.3
Boston	13.6	15.4	10.0	8.2
Chicago	16.0	12.7	9.5	6.1
Los Angeles	12.3	10.9	8.5	4.7
New York City	11.5	10.9	8.7	6.1
San Francisco	10.7	11.8	7.3	4.5
Toronto	15.2	11.7	10.0	5.3
Washington D.C.	13.0	13.3	8.2	5.2
Average	**13.5**	**12.5**	**9.2**	**5.9**

Tabulka 4.1: SMAPE čtyř různých predikčních modelů [13]



Obrázek 4.2: Porovnání reálných dat a predikce bayesovského modelu [13]

4.4 Shrnutí analýzy společnosti Uber

Zkoumání konkurenčních řešení je důležitá část vývoje nového systému. Můžeme se průzkumem nechat inspirovat na cestě k optimálnímu řešení nebo se naopak vyvarovat nevhodných návrhů. Jak jsme se dozvěděli, Uber má velmi komplexní mikroservisní architekturu. Přestože v projektu naší škály nepotřebujeme takto složitý návrh, směřování k tomuto typu architektury nám v budoucnu může ušetřit problémy se škálováním aplikace. S ohledem na budoucnost projektu bychom se také mohli inspirovat NoSQL databází, která je dynamická a dobře škálovatelná [12]. Z předchozí kapitoly 4.3 lze na základě předložených dat usoudit, že neuronové sítě dokážou dobře předpovídat časové řady a že se jedná o složitou, ale optimální cestu, kterou se lze ubírat.

Kapitola 5

Analýza technologií

V této kapitole se zaměříme na průzkum nástrojů pro implementaci systému, jenž budeme vyvíjet na základě poznatků z teoretické části a sběru požadavků. V první části se zaměříme na možnosti multiplatformního vývoje mobilních aplikací. Ve druhé potom na nástroje, které máme k dispozici, abychom mohli navrhnout a implementovat prezentační vrstvu (frontend) v souladu s teorií z kapitol 2.2, 2.3 a požadavky z kapitoly 3.2. Ve třetí části se budeme zaobírat vhodnými nástroji pro návrh vrstvy operující s daty (backend). V závěru vybereme platební bránu se zaměřením na minimální poplatky a jednoduchost integrace do aplikace. Vybrána bude sada nástrojů nejvhodnější pro dosažení cíle naší mobilní aplikace.

5.1 Multiplatformní vývoj mobilních aplikací

Vývoj a údržba mobilní aplikace mohou být velmi nákladné, a proto je potřeba si hned na začátku zvolit vhodné vývojové metody, které šetří práci a prostředky. V dnešní době jsou nejpoužívanější operační systémy v mobilních zařízeních Android (71 % uživatelů) a iOS (28 %) [14]. Abychom získali co největší počet uživatelů, je třeba aplikace vyvíjet pro oba tyto operační systémy. Porovnání jednotlivých přístupů je shrnuto v tabulce 5.1 v kapitole (5.1.5).

5.1.1 Nativní aplikace

Nativní vývoj představuje opak vývoje multiplatformního. Je zde uveden pro porovnání a také jde o stále hojně využívanou metodu. Nativní aplikace jsou psány v programovacím jazyce, který je určen pro daný operační systém (Java nebo Kotlin pro Android a Objective-C nebo Swift pro iOS). Výhodou tohoto přístupu je, že poskytuje maximální výkon a možnost plného využití hardwarových možností zařízení, což je vhodné pro vývoj her a dalších náročných aplikací. Velkou nevýhodou je nutnost vývoje separátní aplikace pro každý operační systém, což klade dvojnásobné nároky na vývojářský tým, co se týče času, prostředků i znalostí [15].

5.1.2 Webové aplikace

PWA (Progressive Web App) jsou v podstatě internetové stránky běžící uvnitř webového prohlížeče. K vývoji posloužily stejné nástroje jako pro běžné internetové stránky, tedy HTML, CSS, JavaScript, případně další knihovny a frameworky.

Oproti klasickým webovým stránkám jsou zde využívány technologie webového prohlížeče HTML5, který poskytuje mnoho funkcí pro vývoj mobilních aplikací. Výhodou tohoto vývojového přístupu jsou nízké nároky na úložiště telefonu a možnost využít jeden zdrojový kód pro oba operační systémy. K nevýhodám patří minimální přístup k hardwarovým funkcím a nedostupnost aplikace v internetových obchodech dané platformy. Tento typ aplikací se hodí například pro zpravodajství, články, vzdělávání apod. [15]

5.1.3 Hybridní aplikace

Hybridní aplikace jsou v základu opět webové stránky vyvíjené pomocí HTML, CSS a JavaScriptu. Výsledný kód je ale vložen do WebView, které se nachází v samotné nativní aplikaci. Výhodou takového přístupu je možnost nahrát aplikaci do obchodu dané platformy, kde je přístupná pro instalaci. Dalším benefitem je fakt, že kromě WebView je zdrojový kód použitelný pro jakýkoliv operační systém. Díky pluginům také můžeme přistupovat k většině hardwarových funkcí. Nevýhodou je běh uvnitř WebView, ve kterém aplikace nedosahuje zdaleka takového výkonu jako v případě ostatních vývojových přístupů [15, 16].

5.1.4 Multiplatformní aplikace

Multiplatformní aplikace (někdy nazývány jako cross-platform) jsou ze všech možností nejbližší těm nativním. Jsou psány pomocí jednoho programovacího jazyka, který je při tvorbě instalačního souboru zkompilován do nativního kódu dané platformy. Výhody tohoto přístupu spočívají hlavně v rychlosti vývoje, jelikož není nutné zdrojový kód psát dvakrát. Aplikace dosahují vysokého výkonu a mohou přistupovat k nativním API. Jedinou nevýhodou může být opožděná dostupnost nejnovějších funkcí nativního vývoje [15].

5.1.5 Porovnání a volba přístupu k vývoji aplikace

Výhody multiplatformního přístupu jsou pro náš projekt nejpřínosnější. V další sekci tedy rozebereme multiplatformní technologie a následně opět zvolíme tu nejvýhodnější. Závěrečné srovnání metod vývoje je shrnuto v tabulce 5.1.

Metoda vývoje	Rychlost vývoje	Rychlost aplikace	Znalost progr. Jazyků	Podpora hardwarových funkcionalit
Nativní	pomalý	nejrychlejší	Java, Kotlin, Objective-C, Swift	plná podpora
PWA	nejrychlejší	srovnatelná s nativní	JavaScript, HTML, CSS	omezená, vyžaduje HTML 5
Hybridní	rychlý	znatelně pomalejší	JavaScript, HTML, CSS, AngularJS	srovnatelná s nativními
Multiplatformní	**rychlý**	**srovnatelná s nativní**	**JavaScript, Angular, React Native, C#, Dart, Vue.js**	**srovnatelná s nativními**

Tabulka 5.1: Srovnání přístupů k vývoji multiplatformních aplikací [15]

5.2 Rozbor multiplatformních frontendových technologií

V roce 2022 patří k nejpopulárnějším multiplatformním rozhraním React Native, Xamarin a Flutter [17, 18, 19, 20]. V následujícím rozboru porovnáme jejich výhody a nevýhody a vybereme rozhraní, které se nejlépe hodí pro požadavky naší aplikace. Porovnání jednotlivých přístupů je shrnuto v kapitole 5.2.4.

5.2.1 React Native

React Native, který se poprvé objevil v roce 2015, je JavaScript open-source framework od společnosti Meta (dříve známé pod názvem Facebook). Výhodou tohoto rozhraní je poměrně vysoká stabilita a veliká komunita vývojářů. Velmi přesvědčivá také může být skutečnost, že React Native používají kromě Mety také společnosti Tesla nebo Discord. Nevýhodou je nižší výkon, pokud se jedná o hry nebo jinak graficky náročné aplikace. Často je také nutné při vývoji používat knihovny třetích stran [17, 18, 19].

5.2.2 Xamarin

Xamarin je open-source rozhraní postavené na programovacím jazyce C#. První verze byla uvedena v roce 2011, v roce 2016 Xamarin koupila společnost Microsoft. Hlavní výhodou tohoto rozhraní je velmi vysoký výkon. Mezi nevýhody patří malá komunita, složitější syntax a občasná nutnost psát nativní kód, vyžaduje-li aplikace velké množství funkcí. Rozhraní navíc není tolik využíváno velkými světovými společnostmi [17, 18, 19].

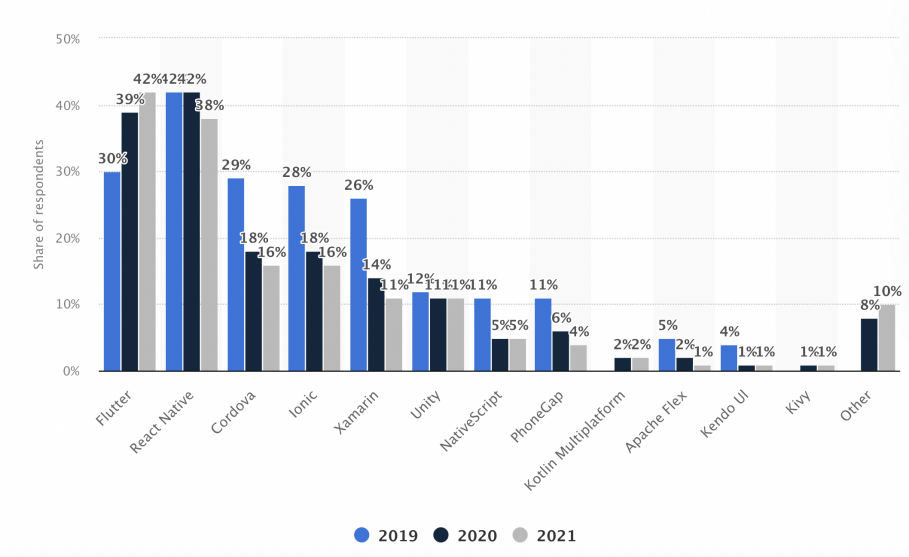
5.2.3 Flutter

Flutter je open-source rozhraní pro vývoj softwaru vytvořené společností Google v roce 2017. Programovacím jazykem je Dart, což je objektově orientovaný jazyk syntaxí velmi podobný JavaScriptu. K výhodám Flutteru patří vysoký výkon, jednoduchá a přehledná syntax (zejména struktura widgetů), velká komunita a kvalitní dokumentace. Menší nevýhodou může být nutnost učit se nový jazyk Dart, což ovšem není tak náročné, protože se hodně podobá JavaScriptu. Dále se také může projevit nižší stabilita, jelikož se stále jedná o poměrně novou technologii [17, 18, 19].

Je sice fakt, že kromě Googlu tohle rozhraní nepoužívá tolik světových společností, dá se ale předpokládat, že to Google s Flutterem myslí vážně, například v roce 2020 byla do něj přeprogramována značně používaná aplikace Google Pay [21].

5.2.4 Porovnání a volba rozhraní

Všechny tři technologie jsou takřka srovnatelné a těžko se z nich vybírá jedna jediná. Pro tento projekt byl nakonec zvolen Flutter, a to na základě preference z hlediska syntaxe a také trendu popularity, který se zdá být rostoucí (obrázek 5.1).



Obrázek 5.1: Multiplatformní frameworky používané vývojáři po celém světě mezi lety 2019 a 2021 [22]

5.3 Rozbor backendových technologií

Při volbě backendu máme v podstatě dvě možnosti: naprogramovat si vlastní, nebo využít tzv. backend-as-a-service (BaaS). První možnost pro náš projekt nepřipadá v úvahu, jelikož si z časových důvodů pro splnění požadavků na vyvíjený systém nemůžeme dovolit programovat backend od nuly. BaaS je servisní model, který obsahuje předpřipravené funkce a komponenty, tudíž je nachystaný k okamžitému použití [23]. Nevýhodou tohoto přístupu může být nižší flexibilita, zabezpečení dat v rukou třetí strany nebo vyšší cena při extrémním škálování [24]. Žádná z těchto nevýhod nepředstavuje pro náš MVP překážku. V následujících segmentech budou porovnány nejvyužívanější BaaS řešení, kterými jsou Amplify, Azure a Firebase na základě dat [25], která vykazují nejvyužívanější cloudové platformy.

5.3.1 Amplify

Amplify je sada účelově vytvořených nástrojů a funkcí, které umožňují frontendovým webovým a mobilním vývojářům vytvářet full-stack aplikace. Jedná se o produkt společnosti Amazon, konkrétně Amazon Web Services (AWS). Služba Amplify je integrovatelná s rozhraními iOS, Android, Flutter, React Native a Vue a nabízí SQL i NoSQL databázi s řadou užitečných funkcí, ke kterým patří autentizace, úložiště, push notifikace, analytika, machine learning nebo cloudové funkce. Službu lze po dobu 12 měsíců používat zdarma, poté je spuštěn platební model pay-as-you-go [26]. Nevýhodou Amplify může být delší čas strávený učením se s platformou zacházet.

5.3.2 Azure

Microsoft Azure, dříve známý jako Windows Azure, je veřejná cloudová výpočetní platforma společnosti Microsoft. Poskytuje řadu cloudových služeb, včetně

výpočetních, analytických, úložných a síťových. Azure poskytuje SQL i NoSQL databázi s nabídkou funkcí velmi podobnou jako u Amplify. Stejně jako u Amplify lze Azure po dobu 12 měsíců používat zdarma, poté následuje platební model pay-as-you-go [27]. Nevýhodou této služby je absence Flutter SDK nebo jiného řešení v podobě oficiální knihovny. Při vývoji by bylo potřeba se spolehnout na neoficiální knihovny nebo nutné psát platformně specifický kód.

5.3.3 Firebase

Firebase je univerzální a mnohostranný backend, který poskytuje specializovanou cloudovou NoSQL real-time databázi a k ní řadu funkcí a nástrojů, mezi které patří autentizace, push notifikace, analytika, cloudové funkce nebo machine learning. Backendové funkce, které by byly jinak náročné na implementaci, lze díky Firebase realizovat pár řádky kódu. Informace se ukládají do kolekcí a dokumentů stejně jako u ostatních NoSQL databází. Z hlediska zabezpečení má tato platforma své rozhraní, ve kterém lze definovat libovolně komplexní zabezpečovací pravidla. Služby Firebase lze využít skrze SDK [28].

Co se týče ceny, Firebase funguje na modelu pay-as-you-go, což znamená, že měsíční cena se přímo úměrně odvíjí od toho, kolikrát se aplikace dotázala serveru, kolik nahrála na úložiště dat, kolik provedla autentizací a kolikrát aktivovala cloudové funkce. Zároveň do určitého počtu invokací dané služby je tato platforma zdarma, např. méně než 10 tisíc autentizací měsíčně [28].

5.3.4 Porovnání a volba backendu

Všechny tři alternativy BaaS, které byly popsány výše, nabízejí velmi podobné možnosti. Finální volbou pro nás bude Firebase, jelikož se nachází ve stejném ekosystému Googlu jako zvolené frontendové rozhraní Flutter, má jednoduchou strukturu a neomezený free tier. Srovnání jednotlivých řešení BaaS je v tabulce 5.2.

Platforma	Ekosystém	Integrace s rozhraním Flutter	Free tier	Uživatelské rozhraní	Typ databáze
Amplify	Amazon	jednoduchá	12 měsíců	náročné	SQL, NoSQL
Azure	Microsoft	složitá	12 měsíců	průměrně náročné	SQL, NoSQL
Firebase	**Google**	**jednoduchá**	**nižší počet requestů, doba neomezená**	**jednoduché**	**NoSQL**

Tabulka 5.2: Porovnání BaaS alternativ

5.4 Volba platební brány

Volba platební brány není úplně jednoduchý úkol. Existuje jich velké množství, často se od sebe liší malými nuancemi, ne všechny mají zcela transparentní ceník, některé zase komplikovanou integraci s mobilní aplikací. Naším hlavním cílem je najít bránu, která má co nejnižší poplatek z transakcí a je zároveň propojitelná s mobilní aplikací. Pro porovnání jednotlivých služeb vyjdeme z tabulky 5.3.

Platební brána	Poplatek z trasakcí		Měsíční poplatek	Aktvační poplatek	Převod na bankovní účet
Stripe	1,4 až 2,9%	6,50 Kč	od 50 Kč	0 Kč	8 Kč + 0,25%
Global Payments	1%	0,75 Kč	0 až 499 Kč	–	–
Pays.cz	1,50%	1 Kč	0 Kč	600 Kč	0 až 39 Kč
ThePay	0,79 až 1,99%	0 až 2 Kč	0 až 279 Kč	0 Kč	0 až 10 Kč
ComGate	0,59 až 0,79%	1,80 až 2,40 Kč	0 až 149 Kč	0 Kč	0 Kč
GoPay	0,9 až 2,2%	3 Kč	0 až 190 Kč	0 Kč	10 Kč
PayU	**1,60%**	**1 Kč**	**0 Kč**	**999 Kč**	–

Tabulka 5.3: Porovnání platebních bran [29]

Když to vezmeme shora, Stripe je velmi dobře propojitelný s rozhraním Flutter (existuje pro něj přímo propojovací knihovna), ovšem cena služby není akceptovatelná. Global Payments vyžaduje uzavření smlouvy s bankou před samotnou integrací, cena navíc závisí na bance, což je pro náš případ nepřijatelné. Pays.cz nemá veřejnou dokumentaci, proto taky nepřipadá v úvahu. ThePay nemá stejně jako ComGate žádný rozumný způsob integrace s mobilní aplikací. GoPay má příliš vysoký absolutní poplatek z transakce.

Nejlépe nám vychází služba PayU, má poměrně nízký poplatek, ačkoliv není nejnižší. Dále má solidní dokumentaci a možnosti integrace. Aktivační poplatek je zanedbatelný. Jedná se o polskou platební bránu, která funguje v rámci střední Evropy, takže jsme si zvolili právě ji.

Kapitola 6

Analýza struktury aplikace

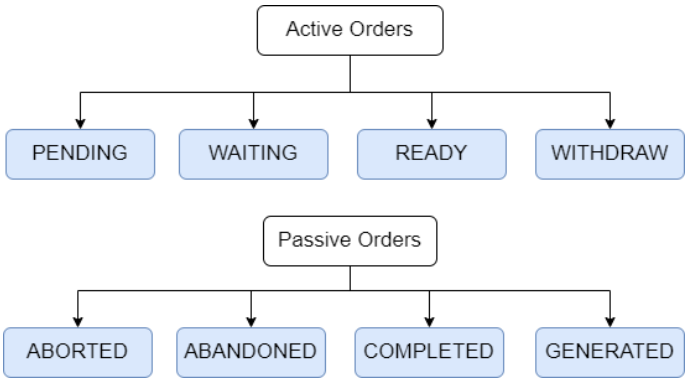
Cílem této kapitoly je analyzovat aplikaci z hlediska toku dat a relací datových tříd. Nejdříve představíme životní cyklus objednávky coby největšího přenosového média informací v systému a poté bude analyzován datový model.

6.1 Životní cyklus objednávky

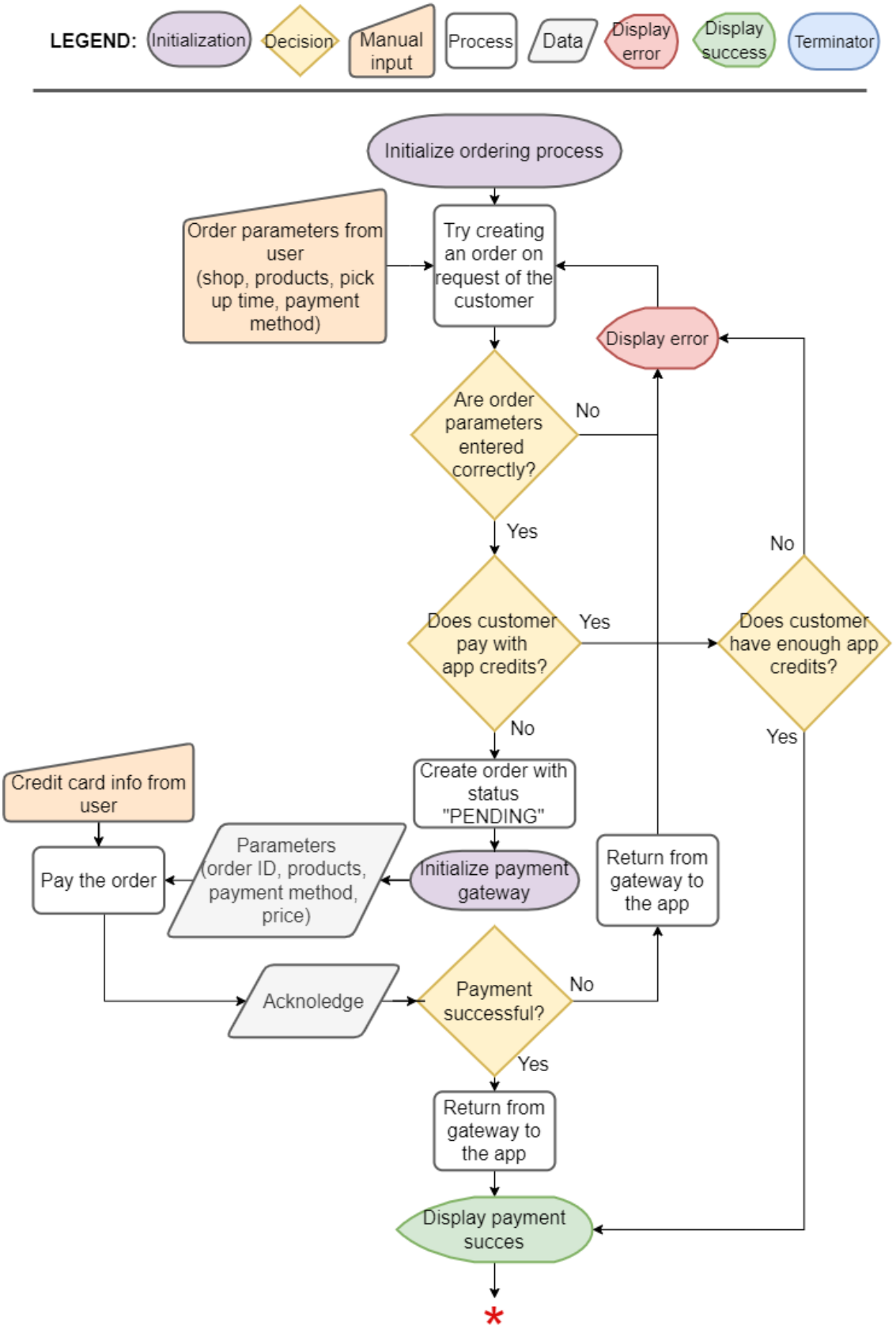
Objednávka je základní stavební kámen informační infrastruktury aplikace z následujících důvodů:

- Slouží jako komunikační kanál mezi zákazníkem a prodejcem.
- Dokument s objednávkou představuje pro zákazníka účtenku.
- Objednávky jsou agregovány podle vlastností do datových bloků, které jsou důležité pro zobrazení statistických údajů v profilu administrátora.

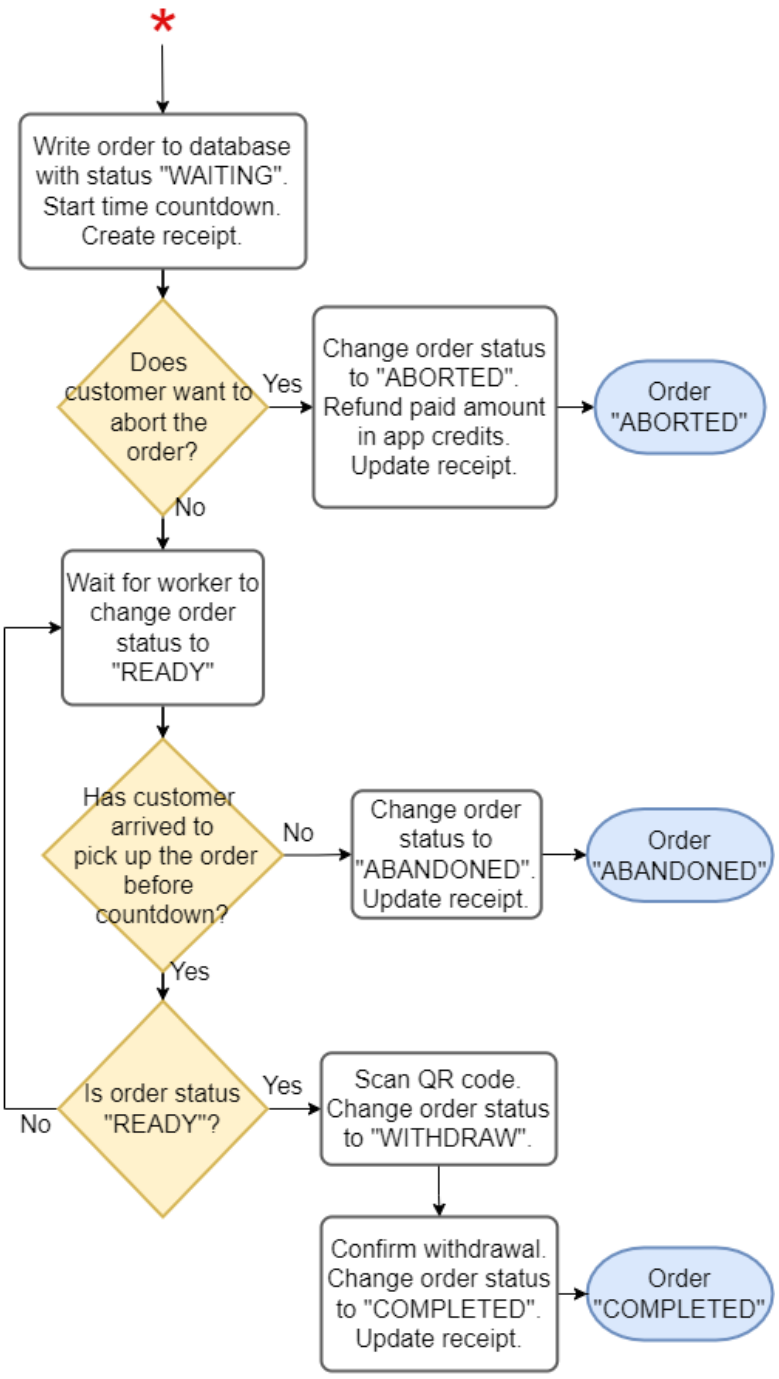
V této podkapitole se zaměříme na životní cyklus objednávky od jejího vzniku až po různé způsoby zakončení (obrázky 6.2, 6.3). Pro pochopení cyklu jsou důležité všechny stavy objednávky, které jsou uvedeny na obrázku 6.1.



Obrázek 6.1: Druhy objednávek a jejich možný stav



Obrázek 6.2: Životní cyklus objednávky část 1.



Obrázek 6.3: Životní cyklus objednávky část 2.

6.1.1 Vznik objednávky

Objednávka může vzniknout dvěma cestami. První cestou projde v případě, že ji vytvoří a zaplatí zákazník vzdáleně ve své aplikaci. Druhou možností je tvorba objednávky pracovníkem, načež je zaplacena zákazníkem fyzicky v prodejně. Za standardní vznik objednávky (jak je prezentována na obrázcích 6.2 a 6.3) budeme považovat tvorbu zákazníkem. Druhý případ vzniku objednávky z profilu pracovníka bude vysvětlen později.

První krok objednávacího procesu zahrnuje výběr provozovny. Lze si vybrat pouze otevřenou provozovnu (podle otevíracích hodin, které řídí administrátor). Ve

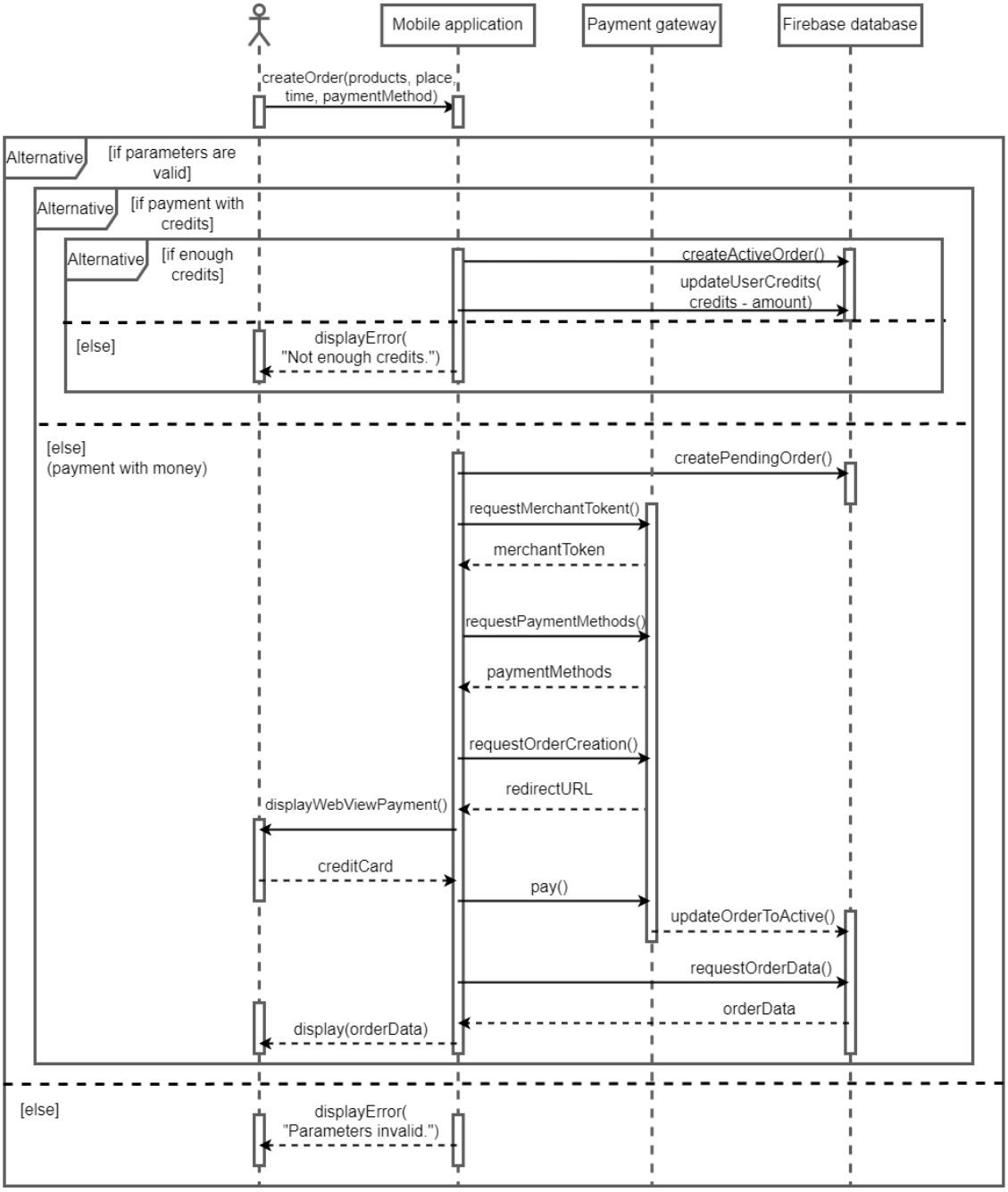
druhém kroku se zákazník dostane do menu, kde si vloží požadované produkty do virtuálního nákupního košíku. Produkty si lze objednat, jen pokud jsou dostupné (dostupnost řídí pracovník v prodejně ve své aplikaci), dále v košíku musí být více než jeden produkt, v opačném případě se zobrazí chybové hlášení při snaze potvrdit objednávku a ta se nezapíše do databáze. Ve třetím kroku lze posuvníkem zvolit čas vyzvednutí a tlačítkem způsob platby (peníze nebo kredity). Pokud zákazník zvolí platební kartu, tlačítkem „zaplatit“ se mu otevírá platební brána, což je čtvrtý a poslední krok tvorby objednávky. Po zaplacení dojde k přesměrování z platební brány zpět do aplikace na obrazovku obsahující detail objednávky. Proces v pozadí, který zajišťuje vlastní transakci skrz platební bránu, je vysvětlen v následující kapitole 6.1.2. Pokud zákazník zvolí platbu v kreditech a má jich dostatek, pak je přeskočen proces platby přes platební bránu. Zákazníkovi jsou odečteny kredity a po úspěšném zápisu objednávky do databáze je zákazník přesměrován na detail evidované objednávky. Pokud zákazník nemá na účtu dostatek kreditů, zobrazí se chybové hlášení a objednávka se do databáze nezapíše.

Případ založení objednávky v profilu pracovníka je mnohem jednodušší proces. Není potřeba vybrat provozovnu, protože objednávka je automaticky vytvořená v provozovně, ve které právě vzniká. Pokud je v košíku alespoň jedna položka, lze umístit objednávku do databáze kliknutím na tlačítko „vytvořit“. Platební proces neprobíhá skrz platební bránu ani pomocí kreditů, nýbrž fyzicky mimo aplikaci.

6.1.2 Komunikace s platební bránou

V momentě, kdy zákazník potvrdí objednávku, u které je nastavená platba platební kartou, spustí se několik funkcí:

1. Proběhne kontrola košíku (jestli obsahuje aspoň jednu položku).
2. Daná objednávka se zapíše do databáze ve stavu `PENDING`. Objednávka v tomto stavu se zatím zobrazuje jen v rozhraní zákazníka, nikoliv v rozhraní pracovníka. Nezaplacené objednávky nemá smysl pracovníkovi ukazovat.
3. Spustí se funkce, která naváže spojení s platební bránou. Tato funkce je popsána sekvenčním diagramem na obrázku 6.4 spolu s dalšími okolnostmi vzniku objednávky. Funkce je také popsána v následujících odstavcích.



Obrázek 6.4: Systémový sekvenční diagram tvorby objednávky zákazníkem

V tomto projektu je použita platební brána PayU. K započetí komunikace je potřeba mít u této brány vedený obchodnický účet a znát tzv. `client_id` a `client_secret` spojené s naším účtem. Pomocí těchto údajů prokážeme svou identitu při komunikaci, která začíná skrze `http post request`, který odešle naše údaje z mobilní aplikace na server platební brány. Pakliže máme správné identifikační údaje, server nám odpoví zprávou, která obsahuje tzv. `bearer_token`. Ten použijeme v dalším dotazu na server, tentokrát se bude jednat o `http get request`, pomocí kterého získáme sadu dostupných platebních metod. Ve finálním požadavku typu `http post request` odešleme typ platební metody, `bearer_token`, `client_id`, cenu a seznam produktů. Ve zprávě budou dále parametry `notifyUrl`, `continueUrl`, `extOrderId` (význam těchto parametrů osvětlíme v dalším popisu). Server nám odpoví zprávou obsahující URL adresu, která slouží k přesměrování zákazníka na platební bránu.

Adresu otevřeme nikoliv v internetovém prohlížeči, nýbrž ve WebView, a to z následujícího důvodu. Jakmile zákazník zadá své platební údaje a potvrdí platbu, je potřeba se nějakým způsobem dostat zpět do aplikace. K tomu slouží parametr `continueUrl`, který představuje URL adresu, na kterou zákazníka přesměruje platební brána po úspěšné platbě. Ve WebView můžeme definovat podmínku, která ho ukončí, pokud se otevře námi specifikovaná URL adresa (což je právě adresa `continueUrl`). V případě, že by se platební brána otevřela v internetovém prohlížeči, nemáme žádný elegantní způsob jak zažádat o návrat z prohlížeče do aplikace.

Nyní přichází na řadu platforma Firebase a její užitečná funkce Cloud Functions. Po návratu z platební brány je objednávka stále vedena v naší databázi se stavem `PENDING` a je nutné nějakým způsobem zjistit, že prošla platební bránou, zaevidovat tuto skutečnost do naší databáze a zobrazit ji uživateli. K tomu nám poslouží parametr `notifyUrl`, na který posílá server platební brány informaci o objednávce vždy, když se změní její stav. Informaci posílá ve formě `http post request`. Na server Firebase umístíme cloudovou funkci, která tyto požadavky umí zpracovat. Firebase ji přiřadí URL adresu, kterou poté přiložíme při každé tvorbě objednávky (hrazené platební kartou) právě jako parametr `notifyUrl`.

Nyní bude vysvětlen parametr `extOrderId`. V něm posíláme platební bráně ID objednávky z naší databáze. Brána nám poté posílá informace o stavu objednávky nesoucí toto ID. Parametr `extOrderId` reálně nese nejen identifikátor objednávky, ale také identifikátor společnosti, u níž je objednávka vytvořena a identifikátor zákazníka, který objednávku odeslal. Tyto tři údaje se pro platební bránu zakódují do jednoho dlouhého řetězce a jsou oddělené podtržítkem. Je potřeba to tímto způsobem provést, jelikož parametr `extOrderId` je jediné unikátní pole, prostřednictvím kterého lze tuto informaci přeposílat.

Tělo zprávy, kterou nám pošle platební brána, je zakódované jako JSON, požadovaný parametr z něj vytáhne naše cloudová funkce. Extrahovaný parametr rozštěpí na jednotlivé identifikátory a najde podle nich záznam duální objednávky v databázi (dualita je vysvětlena v kapitole 6.2.2). Pokud platební brána odešle oznámení o úspěšném zpracování, přepíše cloudová funkce v databázi stav objednávky na `WAITING`.

Tímto procesem se stala objednávka aktivní, což se automaticky projeví v uživatelském rozhraní zákazníka i pracovníka.

6.1.3 Vyřízení objednávky

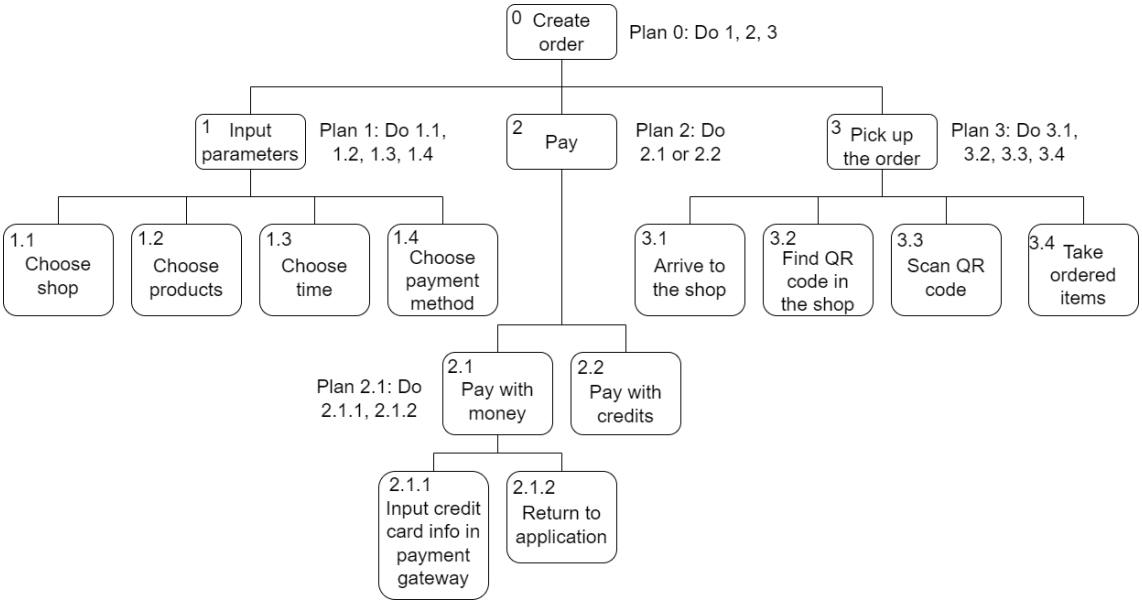
Cyklus objednávky může být ukončen několika způsoby, dva jsou negativní a jeden pozitivní. Ve všech případech nedojde v databázi ke změně dat objednávky, ale k jejímu smazání z kolekce aktivních objednávek a k jejímu zápisu do kolekce pasivních objednávek. Takto rozdělíme všechny existující objednávky na dvě kolekce. Kolekce aktivních objednávek je zpravidla menší, je v ní prováděno více zápisů a čtení. Kolekce pasivních objednávek slouží spíše jako archiv, má větší datový objem a je v ní prováděno méně zápisů a čtení.

První negativní vyřízení objednávky nastává, když se zákazník rozhodne objednávku zrušit. Taková skutečnost se projeví tím, že objednávka v databázi získá stav `ABORTED`. Aktem zrušení se zákazníkovi přičte na jeho účet v aplikaci suma kreditů odpovídající ceně objednávky (vracení kreditů místo peněz je vysvětleno v kapitole 3.2.2). Kredity lze pak využít k zaplacení příští objednávky.

Další negativní vyřízení nastane, když se zákazník pro svou objednávku ne-

dostaví do určitého časového intervalu. Po jeho uplynutí objednávka získá stav `ABANDONED`, přičemž zákazník ztrácí na objednávku nárok bez možnosti refundace.

Ideální ukončení objednávky nastane v případě, že si zákazník objednávku vyzvedne. Možnost vyzvednutí je ovšem podmíněna tím, že pracovník v dané provozovně nejdříve fyzicky zboží objednávku připraví a poté ji v aplikaci eviduje jako připravenou, čímž se změní její stav z `WAITING` na `READY`, což se projeví v aplikaci zákazníka. Pokud je objednávka připravena, zákazník si ji může jednoduše vyzvednout po naskenování QR kódu, který je fyzicky umístěn v provozovně poblíž místa výdeje produktu. V QR kódu je zakódovaný název kavárenské společnosti a adresa provozovny. Pokud adresa získaná z QR kódu souhlasí s adresou objednávky, kterou se zákazník pokouší vyzvednout, změní se v databázi stav objednávky na `WITHDRAW`. V aplikaci pracovníka se zobrazí, že daná objednávka právě podléhá procesu vyzvednutí, načež je pracovník povinen zboží fyzicky vydat. Poté pracovník označí objednávku jako vydanou a ta se dostane do stavu `COMPLETED`. Pokud objednávku nevytvoří zákazník, ale pracovník v prodejně, objednávka získá status `GENERATED`. Hierarchická analýza úloh spojených s celým procesem objednávky je ukázána na obrázku 6.5.

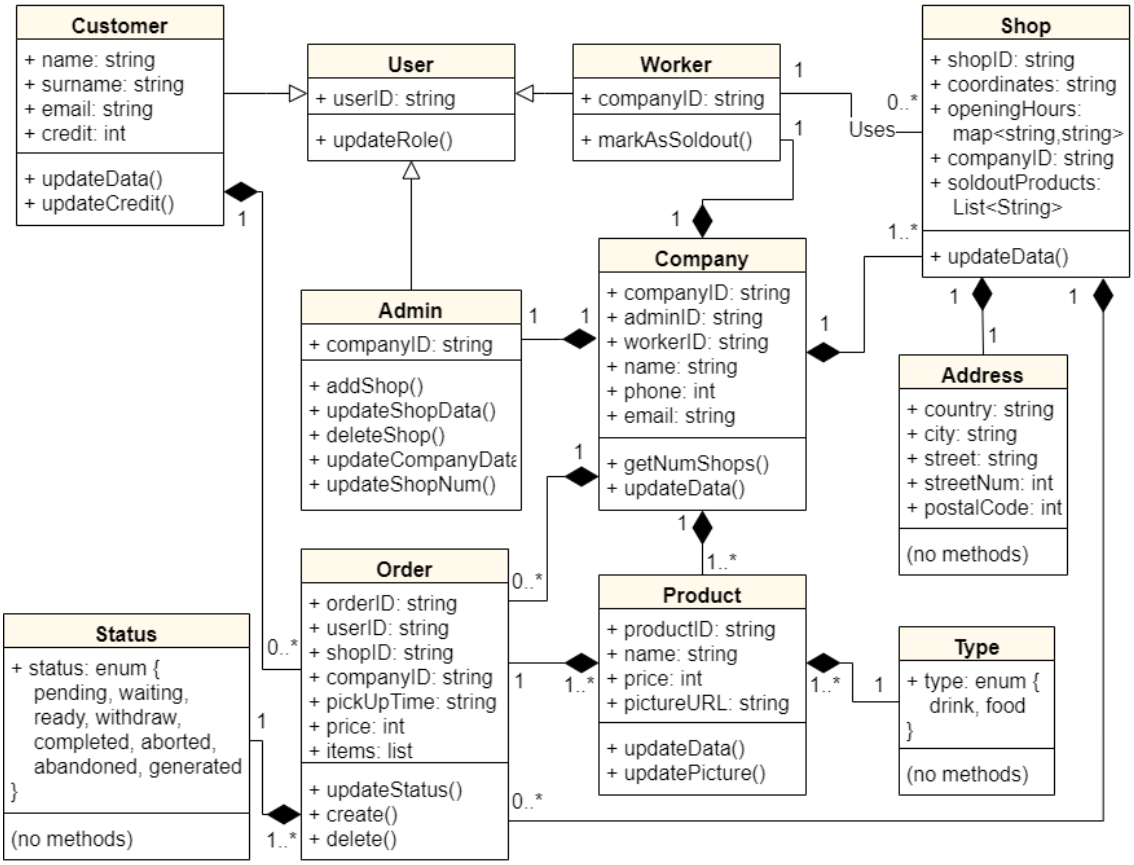


Obrázek 6.5: HTA diagram procesu objednávky

Zákazník by teoreticky mohl provést vyzvednutí objednávky stisknutím tlačítka, aniž by cokoliv skenoval. Technologie QR kódu je zde však využita proto, aby zákazník nemohl omylem ani úmyslně odkliknout objednávku, pokud se fyzicky nenachází v provozovně. Odkliknutí objednávky mimo provozovnu by mohlo způsobit zmatek, tudíž QR kódy v tomto případě zvyšují snižují pravděpodobnost chyby.

6.2 Datový model

Datový model je důležitým podkladem pro tvorbu návrhu aplikace. Na základě diagramu tříd (obrázek 6.6) bude vysvětlena volba jednotlivých tříd, jejich asociace a dědičnost. Všechna ID, která budou v popisu zmíněna, jsou řetězce (string) o délce 20 znaků, skládající se z čísel a písmen. Výjimkou je uživatel, který má ID o délce 28 znaků.



Obrázek 6.6: Diagram tříd

6.2.1 Uživatel

Každý uživatel (user) má své unikátní ID a svou roli, která může být: zákazník (customer), pracovník (worker) nebo administrátor (admin). Tyto role jsou potomky uživatele, dědí jeho vlastnosti a přidávají navíc další. Zákazník má oproti uživateli navíc kontaktní údaje a kredity, za které si může nakoupit produkty. Je také asociován se třídou objednávek, kterých může mít nula a více. Admin má ID společnosti, ke které patří, a širší privilegia v oblasti funkcí. Pracovník je poměrně strohá role, je důležitá hlavně ohledně informace o uživatelském rozhraní, které je pro každou roli jiné.

6.2.2 Společnost

Další třídou je společnost (company), která tvoří protipól k uživateli. Má ID své, účtu administrátora a účtu pracovníka. Obsahuje také kontaktní informace, jako je název, telefonní číslo a e-mail. Každá společnost musí mít právě jeden účet administrátora a pracovníka, jeden a více obchodů, jeden a více produktů. Stejně jako zákazník může mít společnost také nula a více objednávek. Záznamy o objednávkách jsou duplicitně vedeny u zákazníka i u společnosti z důvodu efektivnějšího čtení z databáze.

6.2.3 Objednávka

Objednávka (order) je nejrozsáhlejší třída, přes ID je spojena se společností, zákazníkem a provozovnou. Velmi důležitým parametrem je stav (status), který řídí, kde a komu se objednávka zobrazí. Jelikož nabývá jen omezeného počtu hodnot, jedná se o datový typ enum. Dále objednávka obsahuje cenu, seznam produktů a čas vyzvednutí.

6.2.4 Provozovna

Další třídou je provozovna (shop), má ID své i společnosti, ke které patří, adresu, souřadnice a otevírací dobu, vedenou jako mapa, kde klíč je den v týdnu a hodnota je čas od–do. Provozovny mohou přidávat, upravovat a odebírat administrátoři. Zákazníci si z nich mohou pak vybírat při tvorbě objednávky. K dané provozovně jsou asociovány objednávky, pracovník má tedy možnost vyřizovat pouze ty, které patří k provozovně, v níž pracuje.

6.2.5 Produkt

Poslední třída je produkt, který má opět své ID, název, cenu a typ. Typ udává, zda se jedná o jídlo nebo pití a z toho důvodu je datového typu enum. Obsahuje také URL adresu obrázku, který je uložen v databázi. Produkty mohou přidávat, upravovat a odebírat administrátoři. Zákazníci si je pak mohou přidávat do objednávky.

Kapitola 7

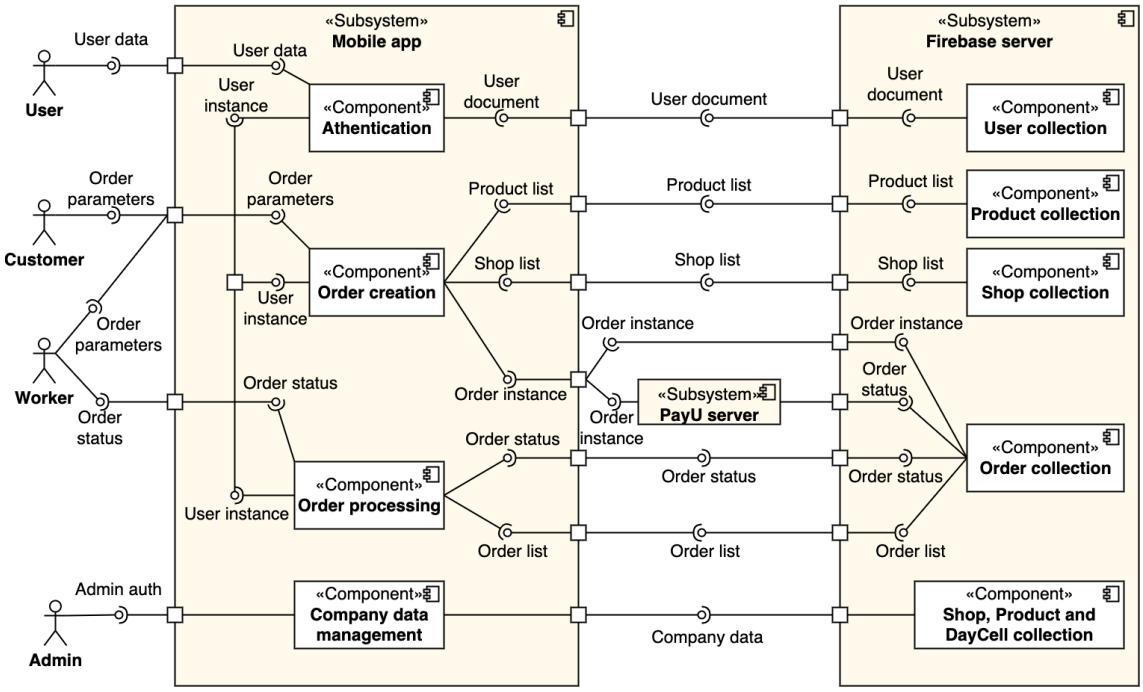
Návrh

Obsahem této kapitoly je návrh mikroservisů, propojení komponent a uživatelského rozhraní na základě rešerše, sběru požadavků a analýzy. Nejdříve vytvoříme koncept uživatelského rozhraní s cílem vytvořit co nejlepší uživatelský zážitek. Poté bude tento koncept otestován na respondentech a na základě zpětné vazby zrealizujeme finální podobu rozhraní.

7.1 Mikroservisy a komponenty

Mikroservisní architektura je opakem monolitické. Jedná se o moderní architekturu, která se skládá z několika nezávislých specializovaných softwarových služeb, jak bylo popsáno v kapitole 4 na příkladu společnosti Uber. Analogií k této architektuře může být lidský organismus, který obsahuje mnoho nepostradatelných specializovaných orgánů, jež dohromady tvoří jeden celek těla. Mezi orgány probíhá komunikace pomocí nervových impulsů, krevního oběhu nebo hormonální soustavy. Stejně tak musí probíhat komunikace mezi mikroservisy softwarové architektury, například pomocí REST API.

Architektura našeho projektu má poměrně málo mikroservisů a blíží se spíše architektuře monolitické, ale může být přece jen rozdělena na tři nezávislé mikroservisy, které mezi sebou komunikují. Prvním je samotná aplikace Flutter, druhým je backendová platforma Firebase a třetím server platební brány PayU. Celá architektura je představena na obrázku 7.1.



Obrázek 7.1: Diagram komponent

Komunikace mezi aplikací a Firebase probíhá pomocí Firebase Flutter SDK, ve kterém jsou zabaleny jednotlivé Firebase APIs. Komunikace mezi aplikací a serverem PayU probíhá pomocí REST API stejně jako komunikace mezi serverem Firebase a PayU.

7.2 Návrh a testování uživatelského rozhraní

Návrh uživatelského rozhraní byl vytvořen pomocí nástroje Figma. Postup návrhu probíhal na základě teorie popsané v kapitole 2 s cílem vytvořit co nejlepší uživatelský dojem. K testování byla zvolena metoda heuristické evaluace sloužící k odhalování chyb spojených s použitelností [5]. Sestavený klikatelný návrh byl předložen čtyřem respondentům, kteří měli za úkol odhalit chyby v návrhu a zhodnotit návrh z hlediska uživatelské přívětivosti. V rámci zpětné vazby bylo spolu s pozitivními ohlasy objeveno i několik drobných chyb a připomínek. Následuje jejich výčet:

Feedback 1: Málo atraktivní design

Dva z respondenti poznamenali, že je potřeba udělat design trochu hravější zakomponováním většího množství trefných obrázků a ikon.

Feedback 2: Špatná dostupnost tlačítek

Jeden z respondentů navrhl odzkoušet design na větších zařízeních, aby se ověřila a případně vylepšila dostupnost tlačítek, která v první iteraci návrhu nebyla podle všeho zcela optimální.

Feedback 3: Špatně viditelná informace o stavu

Během testování byla sdělena ohledně splývající informace o stavu objednávky. Stav objednávky je v původním návrhu zobrazen stejným stylem jako zbytek textu na stránce detailu objednávky. Vezmeme-li v úvahu, že stav objednávky je jedna z nejdůležitějších informací pro orientaci v objednávacím procesu, měla by být náležitě zvýrazněna. Nejlépe by toho šlo dosáhnout vhodným barevným označením jednotlivých stavů objednávky, aby uživatel v ideálním případě poznal stav podle barvy, aniž by četl text.

Feedback 4: Font

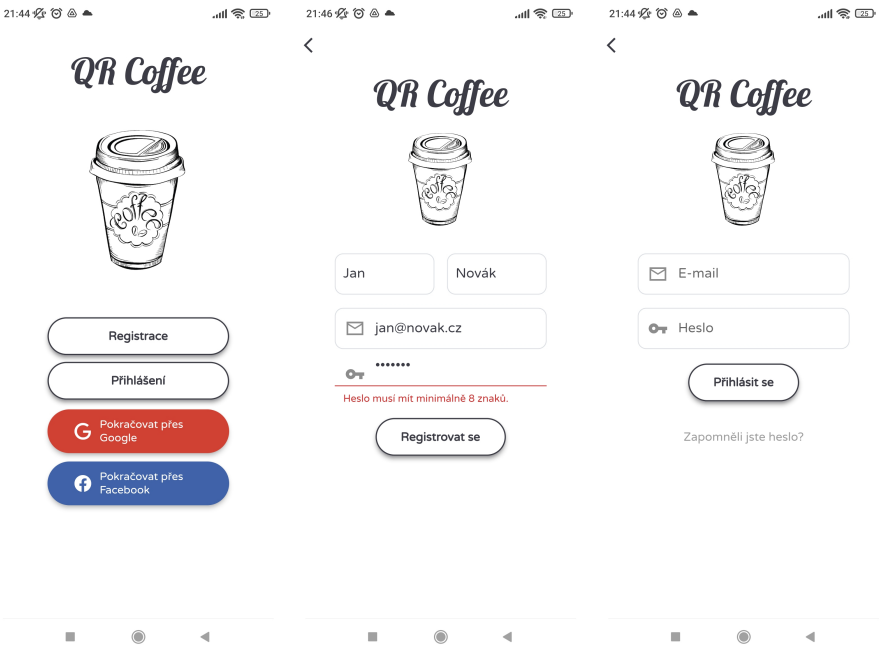
Font písma je poměrně významný detail, který dokáže ovlivnit celkový dojem z designu. V první iteraci designu byl zvolen futuristický font Play, který působil na respondenty moc „sci-fi“ vzhledem k tomu, k čemu je aplikace určená. Ve druhé iteraci byl tedy zvolen mírumilovnější font Varela Round.

7.3 Realizace uživatelského rozhraní

Obsahem této kapitoly je ukázka, jak se podařilo převést návrh uživatelského rozhraní do funkční aplikace. V následujících odstavcích se postupně podíváme na autentizaci a obrazovku zákazníka, pracovníka a administrátora. Screenshoty byly snímány v telefonu Xiaomi Mi 9 SE s rozlišením obrazovky 1080 × 2340 pixelů.

7.3.1 Registrace a přihlášení

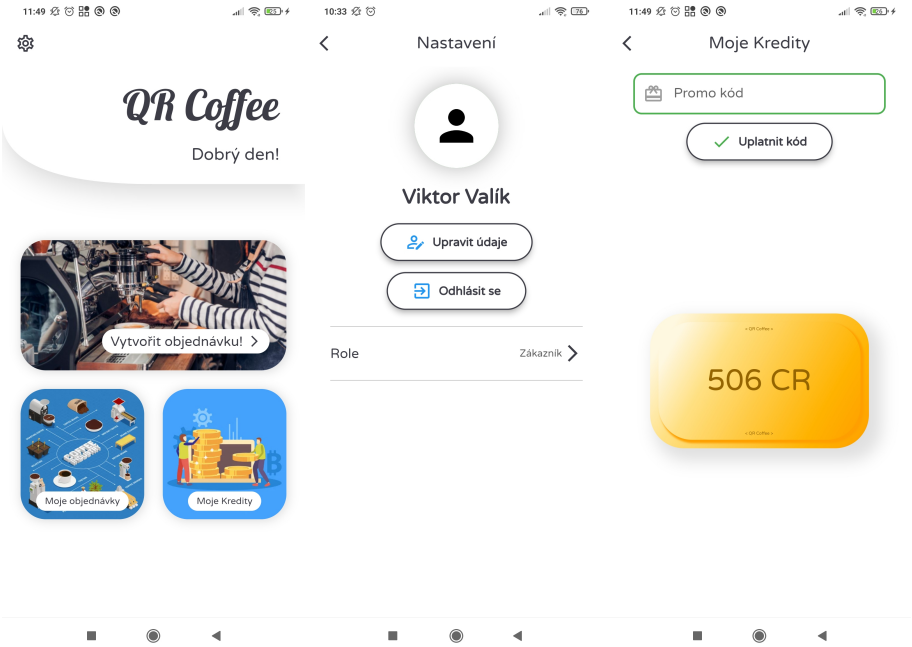
První obrazovka, kterou uživatel uvidí po načtení aplikace, je možnost registrace a přihlášení (obrázek 7.2). Tlačítka pro zapomenuté heslo a přihlášení přes sociální sítě zatím nefungují a slouží jako placeholder pro budoucí zavedení těchto funkcí.



Obrázek 7.2: (snímky zleva) 1. Úvodní obrazovka, 2. Registrace s demonstrací kontroly textových polí, 3. Přihlášení

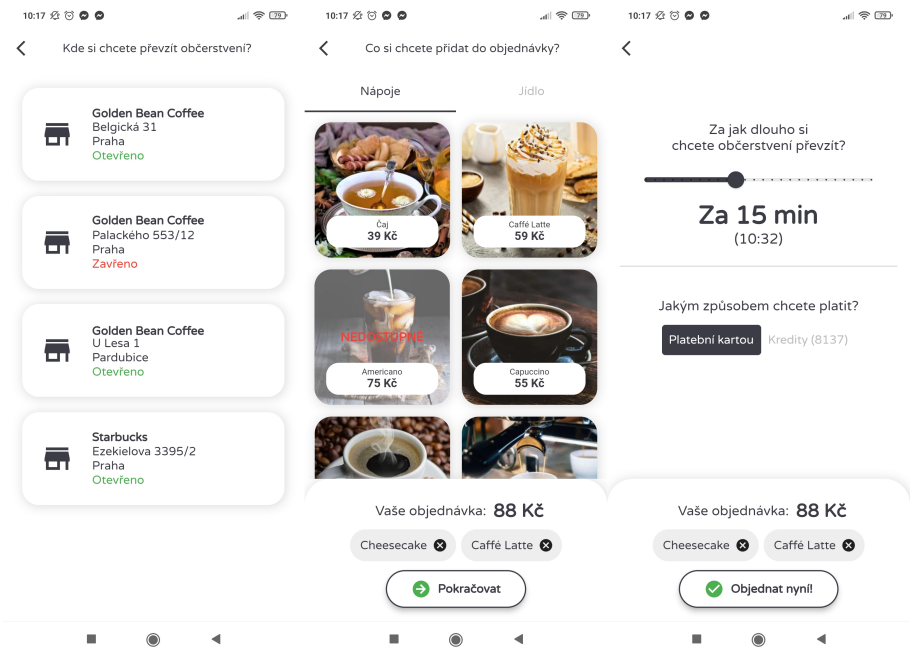
7.3.2 Zákaznické rozhraní

Jakmile se zákazníkovi podaří přihlásit, dostane se na úvodní obrazovku (obrázek 7.3, snímek 1). Vlevo nahoře je možné přejít do nastavení, kde si lze upravit osobní údaje (obrázek 7.3, snímek 2), pravým dolním tlačítkem se dostaneme do přehledu o kreditech (obrázek 7.3, snímek 3).



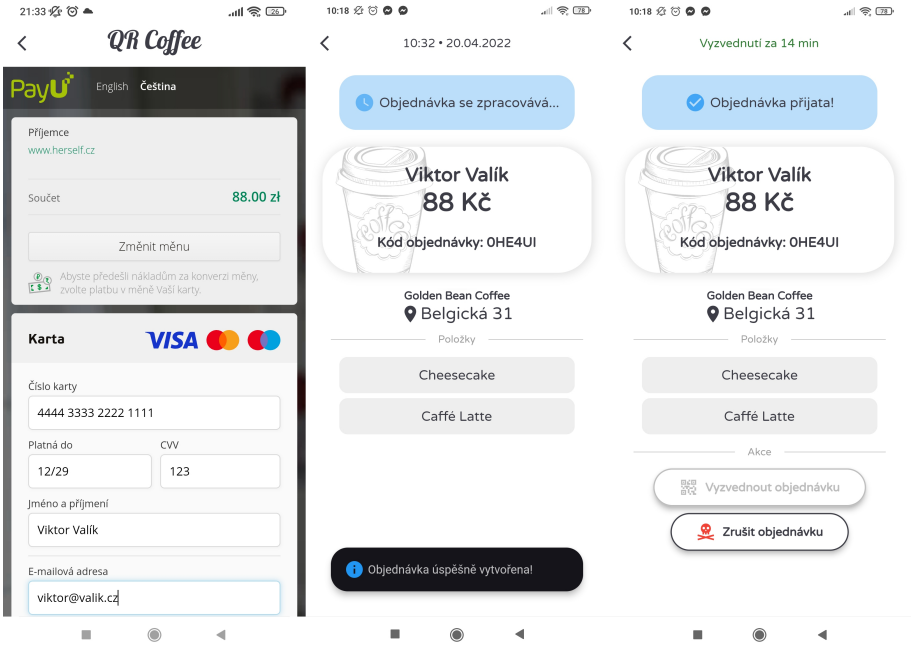
Obrázek 7.3: (snímky zleva) 1. Úvodní obrazovka, 2. Nastavení, 3. Přehled o kreditech

Hlavní funkcí zákaznického rozhraní je tvorba objednávky kliknutím na tlačítko „Vytvořit objednávku!“ (obrázek 7.3, snímek 1). Zákazník se následně dostane na obrazovku s výběrem provozoven (obrázek 7.4, snímek 1), rozkliknout lze pouze provozovnu, která je označena zeleným nápisem „Otevřeno“. Dále si lze přidat do košíku produkty, které daná provozovna nabízí (obrázek 7.4, snímek 2). Po selekci produktů zákazník vybere dobu vyzvednutí a způsob platby (obrázek 7.4, snímek 3).



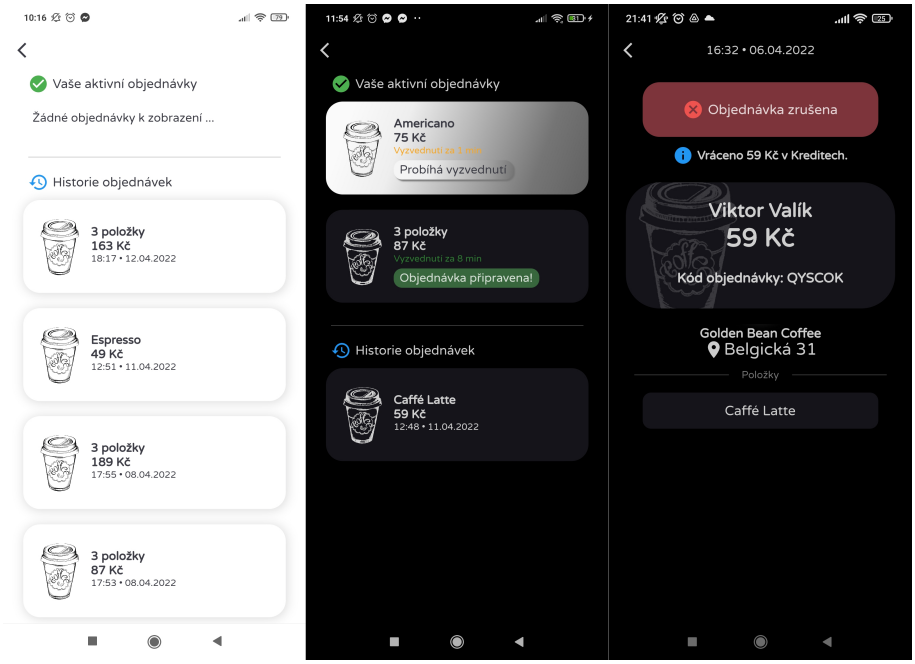
Obrázek 7.4: (snímky zleva) 1. Přehled dostupných provozoven, 2. Výběr produktů, 3. Nastavení doby vyzvednutí a platební metody

Jestliže zvolí platbu kartou, otevře se WebView s platební bránou (obrázek 7.5, snímek 1). Na snímku lze vidět údaje testovací platební karty PayU. Po zaplacení zákazníka aplikace přesune na detail zpracovávající se objednávky (obrázek 7.5, snímek 2). Pakliže projde objednávka platební bránou, zákazník je o tom informován dynamickou změnou obrazovky (obrázek 7.5, snímek 3) a může objednávku zrušit nebo vyzvednout pomocí QR kódu. Tlačítko „Vyzvednout objednávku“ však není klikatelné, dokud pracovník v provozovně nepotvrdí, že je objednávka připravena.



Obrázek 7.5: (snímky zleva) 1. Platební brána, 2. Detail objednávky před zpracováním, 3. Detail objednávky po zpracování

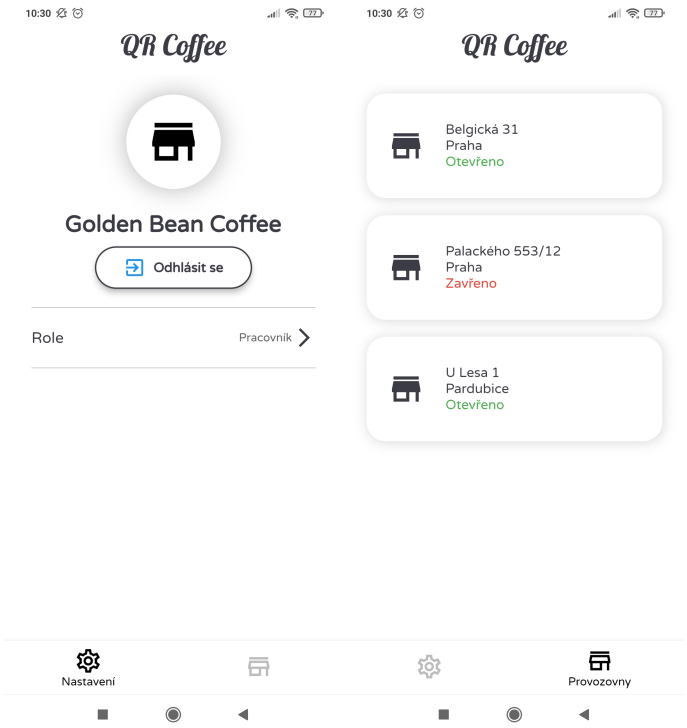
Kliknutím na tlačítko „Moje objednávky“ (obrázek 7.3, snímek 1) se zákazníkovi zobrazí všechny aktivní a ukončené objednávky (obrázek 7.6, snímky 1 a 2). Jestliže objednávku zruší, je informován o množství vrácených kreditů (obrázek 7.6, snímek 3).



Obrázek 7.6: 1. Seznam s prázdnými aktivními objednávkami, 2. Seznam s různorodými typy objednávek v tmavém režimu, 3. Detail zrušené objednávky v tmavém režimu

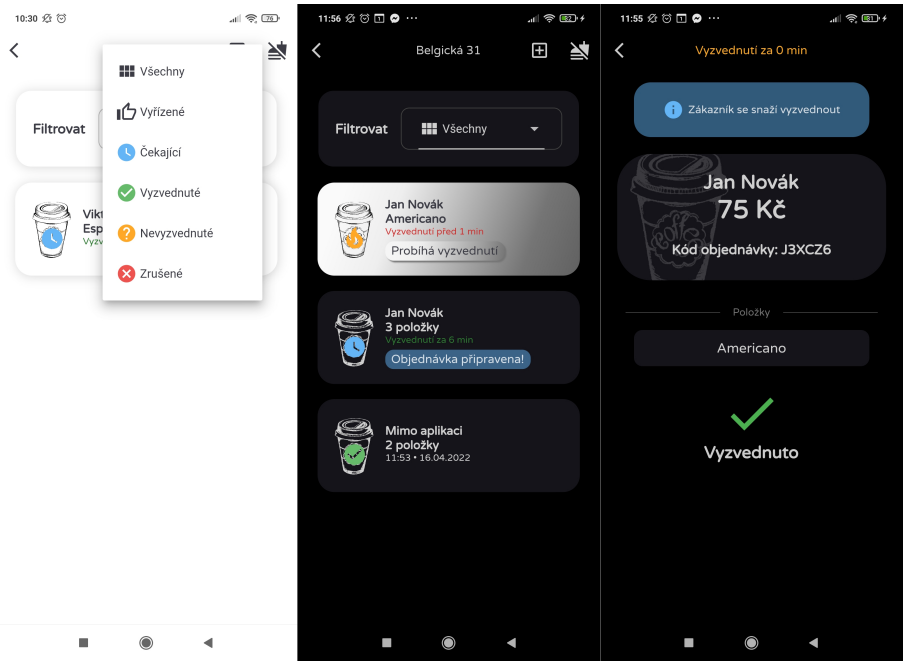
7.3.3 Rozhraní pracovníka provozovny

Hlavní prioritou rozhraní pracovníka provozovny je možnost přijímat objednávky. Nejdříve si pracovník zvolí provozovnu, v níž právě daný den pracuje (obrázek 7.7, snímek 2). Všichni pracovníci dané společnosti operují pod jedním uživatelským účtem, každý však působí v jiné „buňce“ reprezentované dlaždicí dané provozovny.



Obrázek 7.7: (snímky zleva) 1. Nastavení, 2. Selekce provozoven

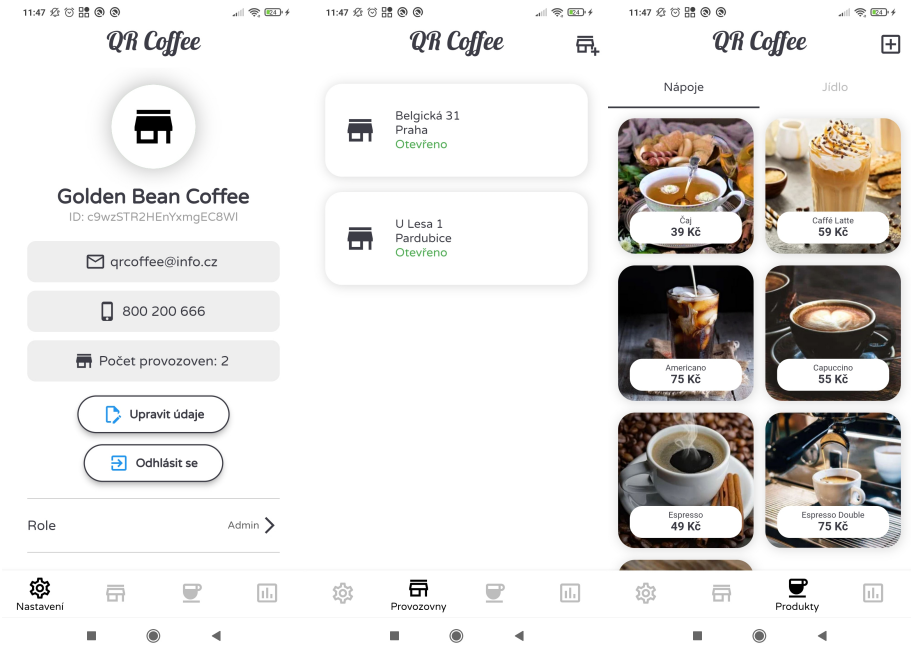
Po rozkliknutí dlaždice s provozovnou, se dostáváme k frontě objednávek (obrázek 7.8, snímek 2), které lze jednoduše filtrovat podle stavu (obrázek 7.8, snímek 1). Na obrazovce s frontou je také možné pomocí tlačítek vpravo nahoře vytvořit objednávku přímo v prodejně, nebo označit vyprodané produkty. Po rozkliknutí detailu objednávky (obrázek 7.8, snímek 3) lze provádět různé akce, které mění stavy objednávky a dochází tak k interakci se zákazníkem.



Obrázek 7.8: (snímky zleva) 1. Filtrování objednávek, 2. Fronta různých typů objednávek v tmavém režimu, 3. Detail objednávky s akčním tlačítkem v tmavém režimu

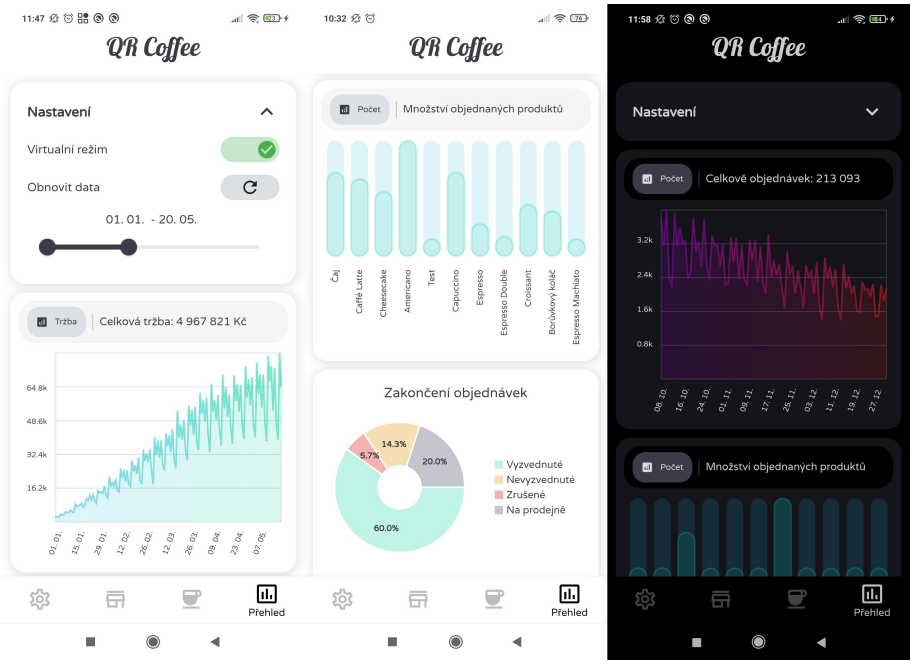
7.3.4 Administrátorské rozhraní

Hlavní prioritou administrátorské části je role centrály pro řetězec provozoven. Na záložce se selekcí provozoven (obrázek 7.9, snímek 2) může administrátor přidávat nové provozovny (kliknutím na ikonu vpravo nahoře) nebo rozkliknout detail provozovny s možností jejího odstranění nebo úpravy. Stejný proces probíhá u produktů (obrázek 7.9, snímek 3).



Obrázek 7.9: (snímky zleva) 1. Nastavení, 2. Selekce provozoven, 3. Selekce produktů

Na poslední záložce má administrátor přístup ke statistickým údajům. První graf (obrázek 7.10, snímek 1) ukazuje tržbu za jednotlivé dny, kliknutím na tlačítko „Tržba“ se hodnoty změní na počet objednávek a tlačítko dostane název „Počet“. Druhý graf ukazuje tržbu za jednotlivé produkty a opět se zde nachází tlačítko „Tržba“ / „Počet“ se stejným efektem jako v předchozím případě. Třetí graf (obrázek 7.10, snímek 2) ukazuje procentuální zastoupení konečných stavů objednávek. V nastavení grafů (obrázek 7.10, snímek 1) se nachází možnost přepínat mezi normálním a virtuálním režimem (bližší informace o rozdílu poskytuje kapitola 8.4). Je zde také oboustranný posuvník, který určuje počet dní zastoupených v grafu a tlačítko vyvolávající nejnovější data.



Obrázek 7.10: (snímky zleva) 1. Nastavení grafů a první z nich, 2. Druhý a třetí graf, 3. Grafy v tmavém režimu

Kapitola 8

Implementace

Tato kapitola je zaměřena na proces tvorby mobilní aplikace, která je zhmotněním analýzy a návrhu našeho systému. Obsahem je princip responzivity, tok dat, state management (neboli správa stavových proměnných), vizualizace dat v administrátorské části aplikace a systém predikce poptávky.

8.1 Responzivita

V následujících podkapitole se podíváme na to, jak byla v aplikaci vyřešena responzivita. Naprogramována byla ručně, jelikož v době programování uživatelského rozhraní měla ještě nejpoužívanější knihovna `responsive_framework` [30] pár nedostatků v některých speciálních případech škálování widgetů.

Vytvořili jsme třídu `Responsive`, ve které jsme definovali několik metod na principu media query, které jsou používány napříč aplikací. Nejpoužívanějšími funkcemi jsou `width` a `height` (obrázek 8.1).

```
static double width(double p, BuildContext context) {
  return MediaQuery.of(context).size.width * (p / 100);
}

static double height(double p, BuildContext context) {
  return MediaQuery.of(context).size.height * (p / 100);
}
```

Obrázek 8.1: Funkce `width` a `height`

Tyto funkce přijímají jako parametr `context`, který udává, v jakém widgetu se nacházíme a desetinné číslo p, které je v rozmezí 0–100 a udává procentuální velikost. Návratová hodnota funkcí je velikost v pixelech. Tyto funkce nám pomáhají škálovat widgety relativně vůči velikosti zařízení.

Další funkce jsou `isLargeDevice` a `isSmallDevice` (obrázek 8.2).

```
static bool isLargeDevice(BuildContext context) {
  return Responsive.deviceWidth(context) >
  kDeviceUpperWidthTreshold ? true : false;
}

static bool isSmallDevice(BuildContext context) {
  return Responsive.deviceWidth(context) <
  kDeviceLowerWidthTreshold ? true : false;
}
```

Obrázek 8.2: Funkce `isLargeDevice` a `isSmallDevice`

Tyto funkce přijímají `context` jako parametr a vrací booleovskou hodnotu `true` nebo `false` na základě toho, jestli je šířka daného zařízení větší než předem nastavená konstanta, která byla experimentálně určena jako nejvhodnější. V aplikaci pak můžeme na základě těchto funkcí rozhodovat o zobrazování a schovávání určitých widgetů s ohledem na velikost zařízení.

8.2 Načítání dat z databáze a ochrana proti napadení

V následující sekci si ukážeme, jak probíhá získávání dat z databáze. Náš frontend je psán v JavaScriptovém frameworku, je tedy dynamicky typovaný, stejně jako je dynamicky typovaná i naše databáze.

8.2.1 Tok dat

Data z databáze Firebase lze v reálném čase streamovat přímo do aplikace. Ukážeme si to na příkladu extrakce pasivních objednávek. Nejdříve je potřeba získat referenci na požadovanou kolekci (obrázek 8.3).

```
CollectionReference _getPassiveOrderCollection() {
  String date = DateFormat('yyyy_MM_dd')
    .format(DateTime.now());
  return FirebaseFirestore.instance
    .collection('companies').doc(companyID)
    .collection('passive_orders').doc('$date')
    .collection('orders');
}
```

Obrázek 8.3: Funkce k získání reference na kolekci

Poté vytvoříme getter funkci, která vrací stream v podobě pole objednávek. Funkce `_OrderListFromSnapshot` ještě navíc návratovou hodnotu mapuje na jednotlivé instance třídy `Order` (obrázek 8.4).

```
Stream<List<Order>> get passiveOrderList {
  return _getPassiveOrderCollection()
    .snapshots().map(_OrderListFromSnapshot);
}
```

Obrázek 8.4: Funkce k získání streamu instancí třídy `Order`

Následně lze tento stream konzumovat v hierarchii widgetů pomocí konstrukce `StreamBuilder`. Na další ukázce kódu (obrázek 8.5) je patrné, že jako stream hodnot použijeme námi vytvořenou getter funkci `passiveOrderList` z předchozího bloku kódu. `builder` je slot pro funkci, která vrací strom widgetů, její hlavní parametr je `snapshot`, ve kterém jsou ukryta data ze streamu. Před sestavením widgetů je třeba zkontrolovat, jestli `snapshot` vůbec nějaká data obsahuje. Pokud ne, musíme poskytnout nějaký fallback widget, nejčastěji se jedná o načítací obrazovku. Pokud data obsahuje, lze z funkce vrátit požadovaný strom widgetů. Jestliže se data v databázi změní, `StreamBuilder` uslyší změnu a přestaví celý strom widgetů, který se v něm nachází.

```
StreamBuilder<List<Order>>(
  stream: OrderDatabase(companyID: admin.companyID)
    .passiveOrderList,
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      List<Order> orders = snapshot.data!;
      return WidgetTree(orders: orders);
    } else {
      return Loading();
    }
  },
);
```

Obrázek 8.5: Konzumace streamu hodnot pomocí konstrukce `StreamBuilder`

8.2.2 Ochrana dat

Ochrana dat proti nedovolenému čtení a zápisu je vyřešena pomocí pravidel definovaných přímo v konzoli Firebase. Pravidla jsou psána ve speciálním syntaxu podobném JavaScriptu. V rámci našeho projektu jsme definovali podmínku, která dovolí provádět změny v databázi, jen pokud má uživatel správný autentizační token (obrázek 8.6). Nikdo mimo aplikaci tudíž nemůže provádět změny. Přímo v aplikaci jsou pak konkrétní akce omezeny rolí uživatele.

```
service cloud.firestore {
  match /databases/{database}/documents {

    // true if user is signed in
    function userSignedIn(){
      return request.auth != null;
    }

    match /{document=**}{
      allow read, write: if userSignedIn();
    }
  }
}
```

Obrázek 8.6: Ochranné pravidlo v konzoli Firebase

8.3 State management

Často potřebujeme přistupovat k určitým důležitým stavovým proměnným napříč systémem. Správný state management, neboli správa stavových proměnných, je tím pádem zásadní. Rozhraní Flutter, ve kterém je naše aplikace naprogramována, má pro tento fenomén několik poměrně šikovných technik. Často používanou konstrukcí pro state management je třída `ChangeNotifier`, která poskytuje notifikaci o změnách pomocí funkce `notifyListeners()`. Následující úryvek kódu (obrázek 8.7) ukazuje konkrétní implementaci správy stavu nákupního košíku v naší aplikaci.

```
class ShoppingCart extends ChangeNotifier {
  List<Product> selectedItems = [];

  void addItem(Product item) {
    selectedItems.insert(0, item);
    notifyListeners();
  }

  void removeItem(int index) {
    selectedItems.removeAt(index);
    notifyListeners();
  }
}
```

Obrázek 8.7: Správa stavu nákupního košíku

K instanci třídy `ShoppingCart` má přístup pouze určitý strom widgetů. Pokud chceme stavové proměnné přenášet mezi různými stromy, potřebujeme `ChangeNotifierProvider` jak je ukázáno na následujícím příkladu (obrázek 8.8).

```
ChangeNotifierProvider(
  create: (context) => ShoppingCart(),
  child: CreateOrderScreen(),
)
```

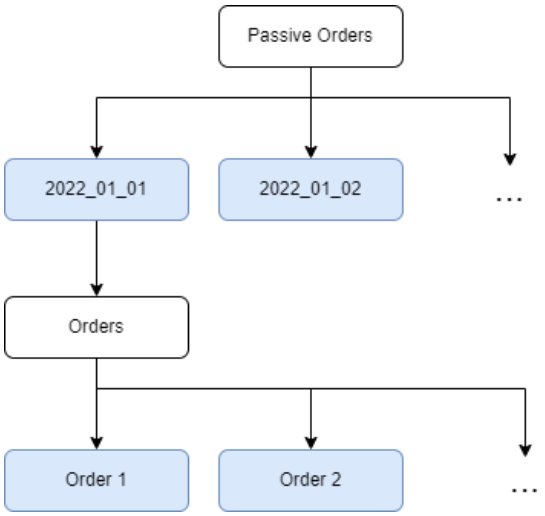
Obrázek 8.8: Ukázka konstrukce `ChangeNotifierProvider`

8.4 Vizualizace dat v administrátorské části

Následující popis se týká postupu implementace grafů a dalších vizualizačních prvků na administrátorském panelu. Popis obsahuje způsob shromažďování a třídění dat a jejich zobrazení pomocí náležité knihovny.

8.4.1 Agregace dat

Jakmile skončí proces objednání, objednávka se dostane do jednoho ze čtyř stavů (`COMPLETED`, `ABORTED`, `ABANDONED`, `GENERATED`). Poté je v databázi uložena v kolekci pasivních objednávek do subkolekce reprezentované datem aktuálním (obrázek 8.9).



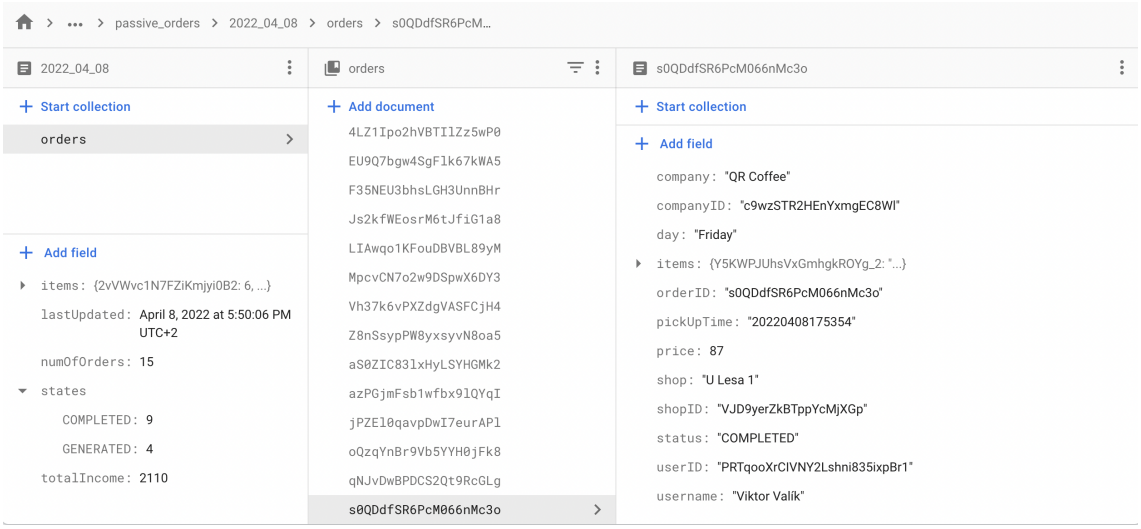
Obrázek 8.9: Rozdělení objednávek v databázi

Každá tato subkolekce reprezentuje buňku, která odpovídá právě jednomu dni. Třída představující buňku je ve zdrojovém kódu označena jako `dayCell` a obsahuje jednu a více objednávek, seznam produktů a jejich počet napříč všemi objednávkami v daném dni, stejně tak seznam konečných stavů objednávek i jejich počet (`items` a `states`). Důležitou informací v buňce je také celková tržba (`totalIncome`) a celkový počet objednávek (`numOfOrders`). Doplňkovou informací je čas poslední aktualizace buňky (`lastUpdated`). Ukázka jedné buňky `2022_04_08` ve Firebase databázi je na obrázku 8.10.

Nyní máme k dispozici časovou řadu, kde osa `x` reprezentuje čas ve dnech a osa `y` tržbu, počet konečných stavů nebo objednávek a produktů. Nyní je potřeba tato data zobrazit v administrátorské části aplikace. K tomu slouží knihovna `fl_charts`, která je zdarma a dokáže na obrazovku vykreslovat poměrně rozmanité druhy grafů.

Pro testovací účely byly také do systému zavedeny kromě pasivních objednávek i virtuální. Tyto objednávky jsou vygenerovány tak, aby zhruba odrážely reálný provoz. Vygenerovaná časová řada má rostoucí charakter s týdenní periodou a malým náhodným šumem. Mezi grafem s reálnými a virtuálními hodnotami lze v aplikaci libovolně přepínat (viz obrázek 7.10 v kapitole 7.3.4).

Třídění objednávek je navrženo tak, aby bylo možné v případě potřeby data dále granulovat. Do subkolekce reprezentující jeden den se dají vložit další subkolekce představující hodiny, minuty atd.



Obrázek 8.10: Ukázka třídění objednávek v databázi

8.4.2 Problémy s knihovnou

Při práci s knihovnou jsme narazili na dva hlavní problémy. Prvním z nich byla vysoká latence a zpomalené scrollování, pokud byla do instance grafu poslána informace o hodnotách na ose `y`, které se pohybovaly nad 100 jednotek. Nepodařilo se odhalit zdroj tohoto problému, pravděpodobně je to způsobeno špatnou optimalizací knihovny. Přistoupili jsme tedy na jednoduché řešení. Hodnoty vydělíme škálovací konstantou, která je vhodným násobkem desítky, aby se výsledná hodnota pohybovala vždy v rozmezí 0–100. Při zobrazení popisků osy `y` pak hodnoty škálovací konstantou vynásobíme, aby řád hodnot souhlasil s původními daty.

Druhý problém se týká dní, ve kterých nebyla vytvořena žádná objednávka. Aby takové dny byly v grafu správně zobrazeny, musí být explicitně jasné, že ten den bylo provedeno nula objednávek. Ovšem v databázi tento den chybí úplně, protože subkolekce příslušející určitému datu se vytvoří až s první objednávkou, která ale nemusí vůbec přijít. Proto bylo potřeba vytvořit funkci, která vytvoří seznam datumů od prvního zaznamenaného dne, který nám přijde z databáze, až po poslední datum. Před vyobrazením dat do grafu funkce `syncCells` (obrázek 8.11) zkontroluje časovou řadu z databáze a doplní chybějící dny nulovými hodnotami.

```
List<DayCell> syncCells(
List<DayCell> dayCells, List<String> dates) {
  List<DayCell> syncedCells = [];
  bool included;
  DayCell includedCell = DayCell.initialData();

  for (String date in dates) {
    included = false;
    for (DayCell dayCell in dayCells) {
      if (date == dayCell.date) {
        included = true;
        includedCell = dayCell;
      }
    }
    if (included) {
       syncedCells.add(includedCell);
     }else{
       syncedCells.add(DayCell.initialData());
    }
  }
  return syncedCells;
}
```

Obrázek 8.11: Ukázka funkce `syncCells`

8.4.3 Backendové funkce shromažďující data

Jakmile je vytvořena první objednávka v daném dni, dojde v databázi k vytvoření jedné buňky `dayCell`. Backendové funkce s každou přidanou objednávkou aktualizují data dané buňky (počet objednávek, tržba atd.). Bez těchto funkcí by v buňce vznikal pouze dlouhý seznam objednávek a počítání by se muselo řešit na frontendu.

Všechny funkce jsou volány nezávisle a asynchronně a spouští se, pokud se zapíše dokument do následující cesty: `'/companies/{companyID}/passive_orders/{date}/orders/{orderID}'`. Složené závorky indikují tzv. wildcard, která označuje dokument s jakýmkoliv ID. Když se tedy zapíše objednávka s libovolným ID do buňky s libovolným datem k libovolné společnosti, funkce se spustí a zapíší do dané buňky informace o objednávce. Jednou z funkcí je `agregateOrderData`, která inkrementuje počet objednávek, celkovou tržbu a zapisuje datum a čas poslední aktualizace (obrázek 8.12).

```
return cell.update({
  'numOfOrders': admin.firestore.FieldValue
                  .increment(1),
  'totalIncome': admin.firestore.FieldValue
                  .increment(snap.data().price),
  'lastUpdated': admin.firestore
                  .FieldValue.serverTimestamp(),
}).catch(err => {
  console.log("Document does not exist.", err)
})
```

Obrázek 8.12: Hlavní část funkce `agregateOrderData`

Další funkcí je `agregateOrderProducts`, jež s každou objednávkou inkrementuje počet jednotlivých produktů objednaných v daném dni (obrázek 8.13).

```
const orderItems = snap.data().items;
var mapItems = {};
if (doc.data().items != null) {
  mapItems = doc.data().items;
}

for (var key of Object.keys(orderItems)) {
const newKey = key.substring(0, 20);
  if (mapItems.hasOwnProperty(newKey)) {
    mapItems[newKey] = mapItems[newKey] + 1;
  } else {
    mapItems[newKey] = 1;
  }
}

return cell.update({ 'items': mapItems }).catch(err => {
  console.log("Document does not exist.", err)
})
```

Obrázek 8.13: Hlavní část funkce `agregateOrderProducts`

Poslední funkce `agregateOrderStates` aktualizuje počet konečných stavů objednávek v daném dni. Funguje velice podobně jako předchozí funkce, proto neuvádíme ukázku kódu.

8.5 Systém predikce poptávky

Systém navazuje na předchozí sekci o vizualizaci dat. Představíme vám jen možnost implementace (návod k implementaci), systém však není zatím v projektu implementován z důvodu složitosti a časové náročnosti. Popis je inspirován postupem [31]. Budeme potřebovat Datalab [32], Firebase Firestore s trénovacími daty a přístup ke cloudovým Firebase funkcím.

Nejdříve je potřeba nastavit Datalab, což je prostředí běžící na Google Compute Engine, které umožňuje streamovat cloudová data přímo do pythonského Jupyter Notebooku. Toto prostředí navíc poskytuje téměř neomezené výpočetní zdroje [31]. Následně načteme data z Firestore do Pandas dataframe [33], který je vhodný pro ML.

Dále potřebujeme vytrénovat model, který si můžeme buď zpracovat sami, nebo použít hotový model, například ze softwarové knihovny SciKit [34]. Pro náš příklad (obrázek 8.14) použijeme Multi-layer Perceptron Regressor [35].

```
from sklearn.neural_network import MLPRegressor
sklearn.metrics import mean_absolute_error

model = RandomForestRegressor()
model.fit(X_train, y_train)preds = model.predict(X_test)

print("Model Mean Absolute Error MAE {}"
  .format(mean_absolute_error(y_test, preds)))
```

Obrázek 8.14: Trénování modelu

Následně uložíme model na úložiště Firebase jako Storage Bucket, který může být dále zpracován pomocí ML Enginu (obrázek 8.15).

```
from sklearn.externals import joblib
from firebase_admin import storage

joblib.dump(model, 'model.joblib')
bucket = storage.bucket(name='your-bucket-path')

b = bucket.blob('happy-v1/model.joblib')
b.upload_from_filename('model.joblib')
print('model uploaded!')
```

Obrázek 8.15: Nahrání modelu na Firebase

Nakonec náš model vystavíme jako API endpoint pomocí cloudové Firebase funkce (obrázek 8.16).

```
import { google } from 'googleapis';
const ml = google.ml('v1')

export const predictHappiness = functions.https
    .onRequest(async (request, response) => {

  const instances = request.body.instances;
  const model = request.body.model;

  const { credential } = await google.auth
    .getApplicationDefault();
  const modelName =
'projects/YOUR-PROJECT/models/${model}';

  const preds = await ml.projects.predict({
    auth: credential,
    name: modelName,
    requestBody: {instances}
  } as any);

  response.send(JSON.stringify(preds.data))
});
```

Obrázek 8.16: Cloudová funkce obsluhující náš model

Pokud chceme získat predikci modelu do naší aplikace, stačí poslat na Firebase server request, jehož tělo bude obsahovat název modelu a data ve vhodném formátu. Predikci, kterou dostaneme ve stejném formátu lze poté jednoduše prezentovat v mobilní aplikaci například pomocí knihovny `fl_charts`, kterou jsme již použili pro vizualizaci dat v administrátorské části.

Kapitola 9

Testování

Tato kapitola sumarizuje výsledky testování aplikace. Nejdříve se zaměříme na uživatelské testování, poté na testování responzivity, multiplatformnosti, škálovatelnosti a stability.

9.1 Uživatelské testování

Uživatelské testování mělo ověřit použitelnost návrhu uživatelského rozhraní a nalézt chyby v návrhu a implementaci aplikace. K testování byla použita metoda zvaná kognitivní průchod, což je inspekční metoda testující použitelnost systému na základě připravených scénářů [36]. Scénáře jsou sepsány v kapitole 9.1.2. Respondentům byly po splnění testovacích scénářů navíc předloženy dotazníky, jejichž obsahem bylo hodnocení vlastností systému. Hodnocené vlastnosti byly sestaveny na základě heuristických pravidel podle Jakoba Nielsena [5].

K testování byly použity vlastní mobilní telefony respondentů. Celkově proběhla dvě kola testování s odstupem tří týdnů se starou a novou verzí aplikace. Nová verze aplikace prošla úpravami vyplývajícími z předchozího kola testování. Úpravy se týkaly jak funkčnosti, tak designu.

9.1.1 Respondenti

Uživatelského testování se zúčastnilo pět respondentů. Byli vybráni, tak aby zhruba pokryli věkovou kategorii od 20 do 60 let. Nepředpokládáme, že by tuto aplikaci využívali děti, středoškoláci pravděpodobně velmi zřídka. Také z testování vylučujeme starší 60 let kvůli předpokladu, že tito lidé nejsou tak zběhlí v používání moderních informačních technologií. Profily respondentů jsou popsány v tabulce 9.1.

Respondent 1	
Pohlaví	muž
Věk	22 let
Povolání	student elektrotechnické fakulty
Zařízení	Xiaomi Mi 9 SE, iPad
Respondent 2	
Pohlaví	muž
Věk	55 let
Povolání	CAD návrhář mechanických části
Zařízení	iPhone 13
Respondent 3	
Pohlaví	muž
Věk	34 let
Povolání	finanční ředitel
Zařízení	iPhone XR
Respondent 4	
Pohlaví	žena
Věk	22 let
Povolání	studentka hornicko-geologické fakulty
Zařízení	iPhone 11
Respondent 5	
Pohlaví	žena
Věk	47 let
Povolání	školní psycholog
Zařízení	Realme C11

Tabulka 9.1: Profily respondentů

9.1.2 Testovací scénáře

Následující testovací scénáře jsou navrženy podle případů užití, které byly identifikovány v kapitole 3.2. Testovací scénáře se snaží co nejlépe tyto případy užití pokrýt. U scénáře je vždy uveden počáteční stav aplikace, úkoly a případy užití, které testovací scénář pokrývá.

Scénář 1: Registrace a přihlášení

- Počáteční stav: úvodní obrazovka, uživatel nepřihlášen
- Úkoly:
  1. Vytvořit účet.
  2. Odhlásit se.
  3. Přihlásit se.
- Pokryté případy užití: UC1

Scénář 2: Tvorba objednávky (platba testovací platební kartou)

- Počáteční stav: úvodní obrazovka, uživatel přihlášen v roli zákazníka
- Úkoly:
  1. Vytvořit objednávku.

2. Zadat údaje testovací karty.
   - Číslo: 4444 3333 2222 1111
   - Datum platnosti: 12/29
   - Bezpečnostní číslo: 123
3. Počkat na přijetí objednávky.

- Pokryté případy užití: UC2, UC3, UC4, UC5

Scénář 3: Tvorba objednávky (platba pomocí kreditů)

- Počáteční stav: úvodní obrazovka, uživatel přihlášen v roli zákazníka
- Úkoly:
  1. Zadat promo kód *qrcoffee* (zisk 250 kreditů).
  2. Vytvořit objednávku a zaplatit pomocí kreditů.
- Pokryté případy užití: UC2, UC3, UC4, UC5, UC6

Scénář 4: Vyzvednutí objednávky

- Počáteční stav: úvodní obrazovka, uživatel přihlášen v roli zákazníka
- Úkoly:
  1. Počkat, až bude objednávka připravena.
  2. Vyzvednout objednávku skrze naskenování QR kódu (v realitě umístěn u pokladny, vzhled kódu na obrázku 9.1).
  3. Převzít objednávku.
- Pokryté případy užití: UC7, UC9

Obrázek 9.1: QR kód sloužící k vyzvednutí objednávky

Scénář 5: Zrušení objednávky

- Počáteční stav: Úvodní obrazovka, uživatel přihlášen v roli zákazníka, uživatel má vytvořenou objednávku
- Úkoly:
  1. Zrušit objednávku.

- Pokryté případy užití: UC7, UC8

Scénář 6: Vyřízení objednávky

- Počáteční stav: obrazovka se seznamem provozoven, uživatel přihlášen v roli pracovníka
- Úkoly:
  1. Vybrat provozovnu, ve které se právě nacházíte.
  2. Připravit jednu objednávku a označit ji jako připravenou.
  3. Počkat, až si zákazník přijde vyzvednout objednávku.
  4. Předat objednávku a označit ji jako vyzvednutou.
- Pokryté případy užití: UC11, UC12

Scénář 7: Označení nedostupného produktu

- Počáteční stav: obrazovka se seznamem provozoven, uživatel přihlášen v roli pracovníka
- Úkoly:
  1. Vybrat provozovnu, ve které se právě nacházíte.
  2. Otevřít výběr produktů s možností nastavení dostupnosti produktu.
  3. Označit produkt jako nedostupný.
- Pokryté případy užití: UC13

Scénář 8: Generování objednávky v prodejně

- Počáteční stav: obrazovka se seznamem provozoven, uživatel přihlášen v roli pracovníka
- Úkoly:
  1. Vybrat provozovnu, ve které se právě nacházíte.
  2. Vytvořit objednávku na základě požadavků zákazníka.
- Pokryté případy užití: UC10

Scénář 9: Přidání, změna, odstranění provozovny

- Počáteční stav: obrazovka se seznamem provozoven, uživatel přihlášen v roli administrátora
- Úkoly:

1. Přidat provozovnu.
2. Upravit údaje provozovny.
3. Odstranit provozovnu.

- Pokryté případy užití: UC14

Scénář 10: Vyčtení dané objednávky ze statistiky

- Počáteční stav: obrazovka se statistikami, uživatel přihlášen v roli administrátora, grafy jsou v normálním režimu (nikoli ve virtuálním)
- Úkoly:
  1. Vyčíst údaje k dnešnímu dni (počet objednávek, tržba, počet objednaných produktů, zastoupení konečných stavů objednávky).
  2. Počkat, až dojde k vytvoření nových objednávek.
  3. Aktualizovat obrazovku a vyčíst údaje znovu.
- Pokryté případy užití: UC16

9.1.3 Výsledky testování

Díky uživatelskému testování byly kromě návrhu na úpravu designu také objeveny dvě závažné chyby. Následuje jejich výčet s popisem, příčinou a řešením:

Chyba 1

- Popis: při placení objednávky kartou se naskytl problém během čekání na potvrzení platby. Někdy se stalo, že se objednávka dostala do stavu WAITING (platba přijata) a pak hned zpět do stavu PENDING (stav platby neznámý).
- Příčina: cloudová funkce mění stav objednávky v databázi pokaždé, když dostane informaci ze serveru platební brány. Informace někdy přijde duplicitně nebo asynchronně, takže může dorazit informace o přijaté platbě a po ní informace, že je stav platby neznámý.
- Řešení: podmínkou lze ošetřit, že pokud jednou objednávka získá stav WAITING, funkce nebude reagovat na další zprávy serveru platební brány.

Chyba 2

- Popis: na obrazovce se statistikami došlo při posunutí obou konců oboustranného posuvníku do maxima nebo do minima k zamrznutí aplikace. Pro připomenutí vzhledu posuvníku slouží obrázku 9.2.
- Příčina: posuvník má o jedničku posunuté indexování oproti poli s datumy a došlo tak k chybnému přístupu na index –1 (levý roh) nebo na index n+1 (pravý roh).

- Řešení: v momentě, kdy se posuvníky dostanou do jednoho rohu, přičte se k poloze jednoho z nich jednička. Při interakci pak v praxi nemůže dojít k překrytí.

Obrázek 9.2: Oboustranný posuvník

Nakonec bylo v rámci uživatelského testování hodnoceno šest vlastností na škále 0–10. Průměrné hodnocení respondentů pro jednotlivé verze aplikace je ukázáno v tabulce 9.2.

Vlastnost	Verze 1	Verze 2
Viditelnost stavu	7,5	8
Konzistence a estetika designu	7,25	9,25
Nápověda a prevence chyb	8	8
Efektivita navigace	7,75	8
Užitečnost	9	9
Jednoduchost používání	7	9

Tabulka 9.2: Průměrné hodnocení verzí aplikace respondenty

9.2 Responzivita a multiplatformnost

Ze zadání víme, že má aplikace být plně responzivní, tedy má správně fungovat na zařízeních o různých velikostech. Tato vlastnost byla testována na zařízeních o velikosti obrazovky od 4 do 10,5 palce (tabulka 9.3). Kontrolováno bylo hlavně správné zalamování textu, škálování obrázků, formulářů, ikon a dalších elementů. Testování proběhlo bez problémů. Princip responzivity této aplikace je vysvětlen v kapitole 8.1.

Další podmínkou ze zadání je správné fungování aplikace na systémech iOS i Android. Tato vlastnost byla testována na několika reálných zařízeních i emulátorech s oběma typy operačních systémů. Testování proběhlo téměř bez problému. Jediným zádrhelem byla nutnost zaplatit vývojářský Apple účet, bez kterého by aplikace nešla nainstalovat na reálné iOS zařízení.

Název zařízení	Velikost displeje	Typ	Operační systém
Xiaomi Mi 9 SE	5,97"	reálné zařízení	Android
Realme C11	6,5"	reálné zařízení	Android
iPhone 11	6,1"	reálné zařízení	iOS
iPhone 13	6,1"	reálné zařízení	iOS
iPad Pro	10,5"	reálné zařízení	iOS
Google Nexus S	4"	emulátor	Android
Google Pixel 2	5"	emulátor	Android
Google Pixel 4 XL	6,3"	emulátor	Android
Google Pixel C	10,2"	emulátor	Android
iPhone 13	6,1"	emulátor	iOS

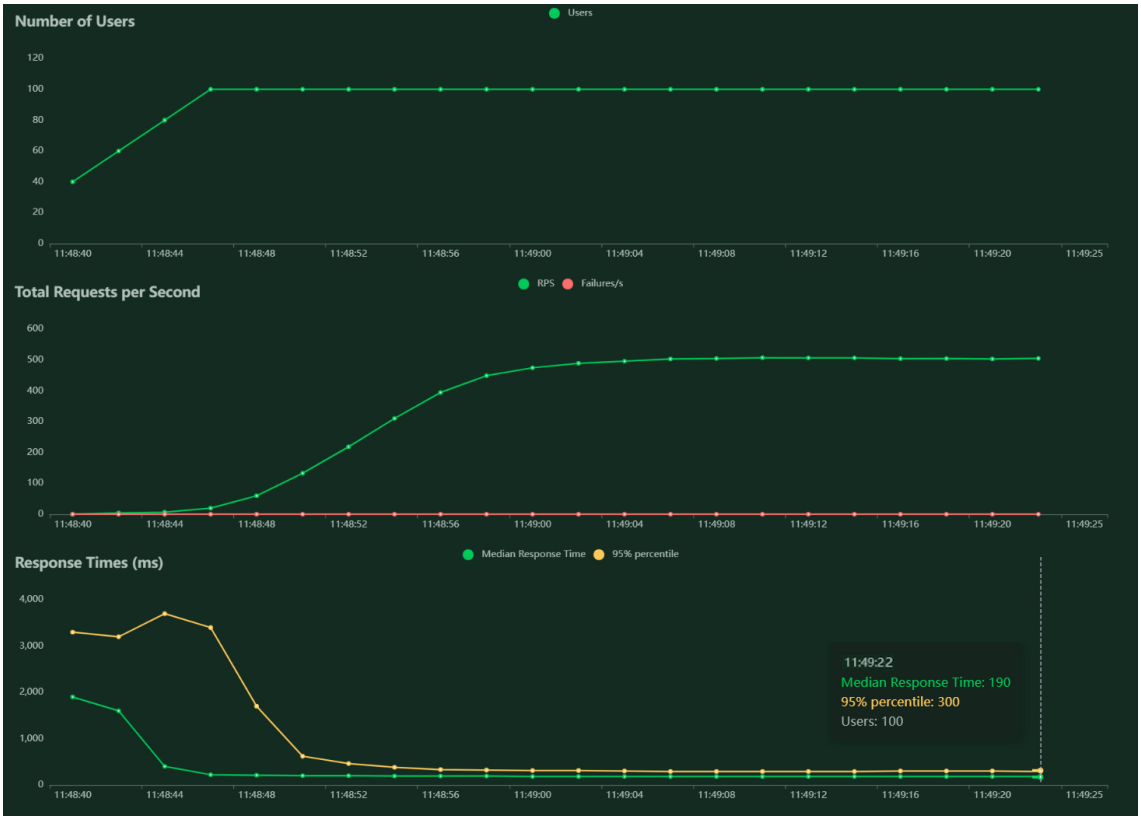
Tabulka 9.3: Přehled testovacích zařízení

Co se týče emulátorů, nebyl zpozorován žádný výrazný rozdíl oproti reálným zařízením. Na obou typech aplikace fungovala ekvivalentně.

9.3 Škálovatelnost a stabilita

Jak bylo již zmíněno, v tomto projektu byl jako backend použita platforma Firebase. Z její dokumentace vyplývá, že se databáze škáluje automaticky podle počtu požadavků. Dostupná data uvádějí, že zvládne až deset tisíc zápisů v tzv. Native Firestore Modu. Pokud je potřeba zpracovat větší škálu požadavků, lze přepnout na tzv. Datastore Mode na úkor omezení real-time funkcí [37].

Přestože jsou dostupná oficiální data, škálovatelnost jsme raději prověřili pomocí nástroje Locust [38]. Jako cíl byla vybrána cloudová funkce `gatewayNotification`, jež slouží jako API endpoint, na který posílá zprávy server platební brány. Zahltili jsme tento endpoint požadavky, jejichž frekvence se zvyšovala od 0 po cca 500 požadavků za sekundu (požadavky odesílalo postupně 40 až 100 virtuálních uživatelů). To bylo maximum, které jsme si mohli dovolit, jelikož bezplatný Firebase účet je omezen na dvacet tisíc zápisů za den, přičemž během jednoho běhu funkce `gatewayNotification` je proveden právě jeden zápis do databáze. Výsledek testování je vidět na obrázku 9.3. Medián doby odezev se drží stabilně kolem 190 ms, přestože zátěž stoupá.



Obrázek 9.3: Graf s výsledky testování pomocí nástroje Locust

Kapitola 10

Závěr

Cílem této práce bylo vytvořit systém, který dokáže propojit síť zákazníků se sítí kaváren a měl by fungovat na platformách iOS i Android a na zařízeních o různých fyzických velikostech. Mezi dílčí požadavky dále patří tvorba a vyřízení objednávky, návrh predikce poptávky a komunikace prodejce s centrálou. V rámci této práce se podařilo vytvořit MVP, který splňuje všechny zadané požadavky.

V první části byl čtenář seznámen s teorií týkající se problematiky HCI. Následně byl proveden rozbor požadavků a konkurenčních systémů spolu s analýzou nástrojů a platforem umožňujících implementaci systému.

Praktická část byla zaměřena na návrh mobilní aplikace s použitím rozhraní Flutter, platformy Firebase a platební brány PayU. Dále byl předložen postup implementace a možnost integrace predikce poptávky do aplikace. Nakonec byl čtenář seznámen s výsledky testování.

Systém není zcela dokonalý a existuje mnoho funkcí, které by bylo vhodné do aplikace doplnit, například zavedení map a notifikací by jistě zlepšilo informovanost a orientaci uživatele v systému. Dále stojí za zvážení několik funkcí, které by mohly značně urychlit dosažení cíle, jako například přihlašování přes sociální sítě, platby pomocí Apple Pay a Google Pay nebo přizpůsobení služeb na základě polohy uživatele.

Literatura

1. *What is a food delivery service and how it works?* [Online]. FarEye [cit. 2022-05-13]. Dostupné z: https://www.getfareye.com/insights/blog/food-delivery.
2. MUKHOPADHYAY, Subhas Chandra. Human-Computer Interaction: Overview on State of the Art. *International Journal on Smart Sensing and Intelligent Systems* [online]. 2017, roč. 1, s. 137–159 [cit. 2022-02-28]. Dostupné z DOI: https://doi.org/10.21307/ijssis-2017-283.
3. TE'ENI, Dov; CAREY, Jane; ZHANG, Ping. *Human-Computer Interaction: Developing Effective Organizational Information Systems*. Hoboken: John Wiley, 2007. ISBN 978-0471677659.
4. HEWETT, Thomas. *Curricula for Human-Computer Interact*. New York: The Association for Computing Machinery, 1992. ISBN 0-89791-474-0.
5. NIELSEN, Jakob. *Usability Engineering*. San Francisco: Morgan Kaufmann, 1993. ISBN 1-12-518406-9.
6. KRUG, Steve. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. New York: New Riders, 2014. ISBN 978-0-321-96551-6.
7. OLAH, Christopher. *Understanding LSTM Networks* [online]. Colah's blog, 2015 [cit. 2022-01-14]. Dostupné z: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
8. PHI, Michael. *Illustrated Guide to LSTM's and GRU's: A step by step explanation* [online]. Towards Data Science, 2018 [cit. 2022-01-14]. Dostupné z: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.
9. *Co je platební brána?* [Online]. Moneta [cit. 2022-03-01]. Dostupné z: https://www.moneta.cz/slovnik-pojmu/detail/co-je-platebni-brana.
10. UPADHYAY, Anu. *System Design of Uber App – Uber System Architecture* [online]. Geeks For Geeks, 2021 [cit. 2022-03-30]. Dostupné z: https://www.geeksforgeeks.org/system-design-of-uber-app-uber-system-architecture/.
11. *Apache Kafka* [online]. Devopedia, 2022 [cit. 2022-04-18]. Dostupné z: https://devopedia.org/apache-kafka.
12. DISSANAYAKE, Kasun. *Uber Architecture and System Design* [online]. Medium, 2021 [cit. 2022-03-30]. Dostupné z: https://medium.com/nerd-for-tech/uber-architecture-and-system-design-e8ac26690dfc.

13. ZHU, Lingxue; LAPTEV, Nikolay. Deep and Confident Prediction for Time Series at Uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* [online]. 2017, s. 103–110 [cit. 2022-01-14]. ISBN 978-1-5386-3800-2. Dostupné z DOI: 10.1109/ICDMW.2017.19.

14. *Mobile Operating System Market Share Worldwide* [online]. Statcounter [cit. 2022-03-03]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile.

15. DOMKAŘ, Petr. *Multiplatformní mobilní aplikace*. Brno, 2018. Diplomová práce. Masarykova univerzita. Vedoucí práce RNDr. Vít Rusňák, Ph.D.

16. DELIA, Lisandro. Multi-platform mobile application development analysis. *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)* [online]. 2015, s. 181–186 [cit. 2022-03-03]. Dostupné z DOI: 10.1109/RCIS.2015.7128878.

17. ALFERD, Sten. *Flutter vs Xamarin vs React Native — Let the Battle Begin!* [Online]. Medium, 2019 [cit. 2022-03-03]. Dostupné z: https://medium.com/@stenalferd/flutter-vs-xamarin-vs-react-native-let-the-battle-begin-d3e783bb4bf1.

18. *Flutter vs React Native vs Xamarin in 2021: Pros, Cons, Examples* [online]. EGO, 2021 [cit. 2022-03-03]. Dostupné z: https://www.ego-cms.com/post/best-framework-for-mobile-app-development.

19. LAM, Chi. *Flutter vs react native vs xamarin: what's best in 2022?* [Online]. InApps, 2022 [cit. 2022-03-03]. Dostupné z: https://www.inapps.net/flutter-vs-react-native-vs-xamarin-whats-best-in-2022/.

20. JAVIN, Paul. *Best Frameworks & Libraries for Cross-Platform Android and iOS Apps in 2022* [online]. Medium, 2020 [cit. 2022-03-03]. Dostupné z: https://medium.com/javarevisited/top-5-frameworks-to-create-cross-platform-android-and-ios-apps-in-2020-d02edf3d01f1.

21. *Going global at Google Pay with Flutter* [online]. Dublin: Google [cit. 2022-01-14]. Dostupné z: https://flutter.dev/showcase/google-pay.

22. *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021* [online]. Statista, 2022 [cit. 2022-03-03]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

23. KUO, Jen-Hao; RUAN, He-Ming; CHAN, Chun-Yi; LEI, Chin-Laung. Investigation of mobile App behaviors, from the aspect of real world mobile backend system. *2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)* [online]. 2017, s. 1–6 [cit. 2022-04-21]. ISBN 978-1-5090-5969-0. Dostupné z DOI: 10.1109/AEECT.2017.8257762.

24. ZALESKI, Mat. *What Is a Mobile App Backend and Does Your Mobile Application Need It?* [Online]. Nomtek, 2021 [cit. 2022-04-21]. Dostupné z: https://www.nomtek.com/blog/mobile-app-backend.

25. SUJAY, Lionel Vailshery. *Most wanted cloud platform among developers, worldwide, as of 2021* [online]. Statista, 2022 [cit. 2022-04-22]. Dostupné z: https://www.statista.com/statistics/793884/worldwide-developer-survey-most-wanted-platform/.

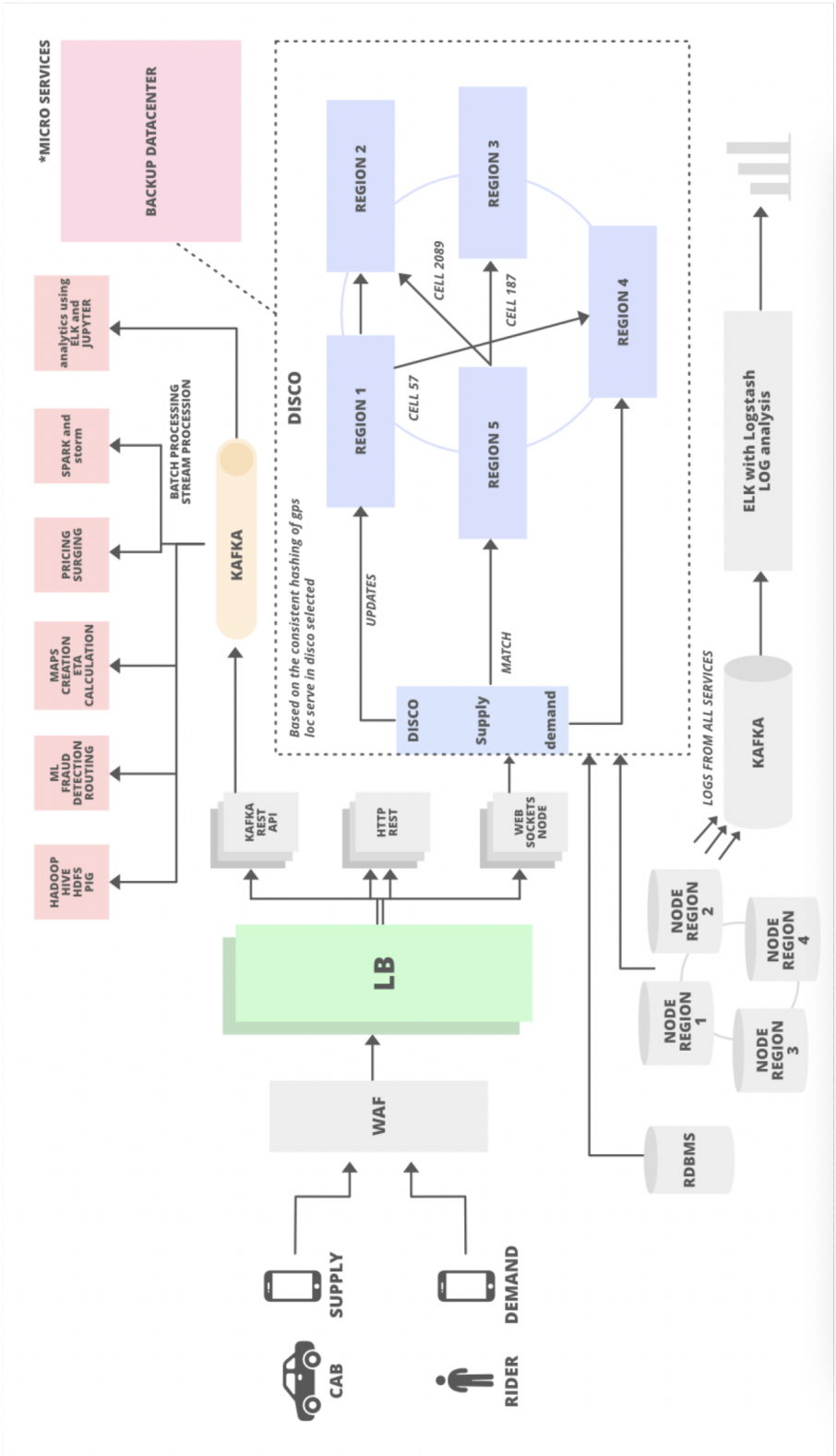
26. *AWS Amplify* [online] [cit. 2022-04-24]. Dostupné z: https://aws.amazon.com/amplify/.
27. *Microsoft Azure* [online] [cit. 2022-04-24]. Dostupné z: https://azure.microsoft.com/cs-cz/.
28. *Learn Firebase fundamentals* [online]. San Francisco: Google [cit. 2022-01-14]. Dostupné z: https://firebase.google.com/docs/guides.
29. *Srovnání platebních bran 2022* [online]. Comgate [cit. 2022-03-04]. Dostupné z: https://www.comgate.cz/blog/srovnani-platebnich-bran.
30. *Responsive Framework* [online]. Pub.dev [cit. 2022-05-06]. Dostupné z: https://pub.flutter-io.cn/packages/responsive_framework.
31. DELANEY, Jeff. *ML Engine Tutorial with Python* [online]. Fireship, 2018 [cit. 2022-04-19]. Dostupné z: https://fireship.io/lessons/serverless-machine-learning-with-python-and-firebase-cloud-functions/.
32. *Datalab documentation* [online]. Google [cit. 2022-04-19]. Dostupné z: https://cloud.google.com/datalab/docs.
33. *Pandas User Guide* [online]. 2022 [cit. 2022-04-19]. Dostupné z: https://pandas.pydata.org/docs/user_guide/index.html.
34. *SciKit* [online]. SciKit [cit. 2022-04-19]. Dostupné z: https://scikit-learn.org/stable/index.html.
35. *Neural Network Models* [online]. SciKit [cit. 2022-04-19]. Dostupné z: https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron.
36. BLACKMON, Marilyn Hughes; POLSON, Peter G.; KITAJIMA, Muneo; LEWIS, Clayton. Cognitive walkthrough for the web. *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02* [online]. 2017 [cit. 2022-04-29]. ISBN 1581134533. Dostupné z DOI: 10.1145/503376.503459.
37. *Building scalable applications with Firestore* [online]. Google [cit. 2022-04-29]. Dostupné z: https://cloud.google.com/architecture/building-scalable-apps-with-cloud-firestore.
38. *Locust* [online] [cit. 2022-05-02]. Dostupné z: https://locust.io.

Přílohy

A Seznam zkratek

MVP Minimum Viable Product
HCI Human-computer interaction
AR Auto Regression
MA Moving Average
RNN Recurrent Neural Network
BNN Bayes Neural Network
LSTM Long-Short Term Memory
QR Quick Response
RDBMS Relational Database Management System
SQL Structured Query Language
QRF Quantile Random Forest
SMAPE Symmetric mean absolute percentage error
HTML Hypertext Markup Language
CSS Cascading Style Sheets
HTA Hierarchical Task Analysis
API Application Programming Interface
SDK Software Development Kit
URL Uniform Resource Locator
ID Identification
JSON JavaScript Object Notation
ML Machine Learning
BaaS Backend-as-a-Service

B Přiložené obrázky



Obrázek 1: Mikroservisní architektura společnosti Uber [10]

C Instalační příručka

Obsah adresáře `qr-coffee-main`:

- `firebase_functions`: zdrojový kód cloudových Firebse funkcí
- `qr_coffee`: zdrojový kód Flutter aplikace

Prerekvizity:

- Nainstalovaný Flutter 3.0.0 a vyšší.
- Účet Firebase typu Blaze.
- Účet PayU (stačí sandbox).

Postup integrace Firebase s Flutterem:

1. Přihlásit se do konzole Firebase.
2. Založit nový projekt.
3. Propojit tento projekt s aplikací podle návodu v konzoli Firebase.
4. V libovolném IDE (doporučeno VS Code) se proklikat do složky `qr-coffee-main/firebase_functions/functions` a v terminálu spustit příkaz `firebase deploy --only functions` k nahrání cloudových funkcí na server.

Postup integrace PayU s Flutterem:

1. Přihlásit se do konzole PayU.
2. Přidat nový obchod Online payments -> My shops -> Add shop.
3. Rozklinkout nově vytvořený obchod a v záložce POS vyhledat POS ID (šestimístný číselný kód) a Client Secret (32 znaků dlouhý string).
4. Otevřít soubor se zdrojovým kódem `qr-coffee-main/qr_coffee/lib/api/manual_api_order.dart`.
   - Do proměnné `sandboxID` vložit své POS ID.
   - Do proměnné `sandboxSEC` vložit své Client secret.
   - Do proměnné `notifyUrl` vložit https adresu API endpointu cloudové funkce `gatewayNotification` (Firebase konzole -> Functions).

Spuštění aplikace:

1. Spustit emulátor nebo připojit reálné zařízeni k počítači.
2. Pro klasické spuštění aplikace se proklikat do složky `qr-coffee-main/qr_coffee` a v terminálu spustit příkaz `flutter run`.
3. Pro spuštění aplikace v debug módu kliknout na Run -> Start Debugging.