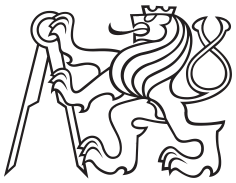
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Návrh vylepšení procesů ve společnosti Country life

Lukáš Havlíček

Vedoucí: Ing. Pavel Náplava, Ph.D
Obor: Softwarové inženýrství a technologie
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Havlíček** Jméno: **Lukáš** Osobní číslo: **499142**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

SW aplikace pro podporu vybraných procesů ve společnosti Country life

Název bakalářské práce anglicky:

SW application for support of selected processes in Country life company

Pokyny pro vypracování:

Vytvořte minimálně první verzi SW aplikace, která pomůže společnosti Country life optimalizovat vybrané procesy. Postupujte následujícím
způsobem:
• Analyzujte aktuální stav procesů ve společnosti Country life a
navrhněte jejich možná zlepšení.
• Zaměřte se na procesy, spojené se skladem a logistikou, které je
možné optimalizovat pomocí IT technologií. Konkrétně podporu dispečerů.
• Pomocí nástrojů SW inženýrství navrhněte a popište aplikaci, která
pomůže dispečerům plánovat objednávky.
• Funkčnost vytvořené aplikace ověřte minimálně formou uživatelských testů.
• Vytvořenou aplikaci vyhodnoťte a navrhněte její další možný rozvoj.

Seznam doporučené literatury:

1. Arlow, J., Neustat, I.: UML 2 a unifikovaný proces vývoje aplikací. Computer Press, ISBN: 978-80-251-1503-9, Praha 2007
2. Dan Ward: React Native Cookbook, Step-by-step recipes for solving common React Native development problems, 2nd Edition, 2019
3. Řepa V., Podnikové procesy - Procesní řízení a modelování, Grada Publishing, 2007

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Náplava, Ph.D. Centrum znalostního managementu FEL

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

Datum zadání bakalářské práce: **30.01.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Pavel Náplava, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. 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

Poděkování

Tímto bych chtěl vyjádřit své poděkování Ing. Pavlu Náplavovi, Ph.D., který mě celou prací provázel, podporoval a poskytoval cenné rady. Také bych chtěl poděkovat firmě Country life za spolupráci, díky které mohla tato bakalářská práce vzniknout. V neposlední řadě chci poděkovat rodině za podporu.

Prohlášení

Prohlašuji, že jsem předloženou bakalářskou práci vypracoval samostatně, a že jsem uvedl všechnu použitou literaturu.

V Praze, 16. května, 2023

Abstrakt

Bakalářská práce se zabývá analýzou, návrhem a implementací mobilní aplikace, která slouží ke zlepšení procesů dispečerů ve velkoskladě firmy Country life. Mobilní aplikace poskytuje přehled objednávek, které se mají rozvážet a umožňuje jejich plánování a řízení, přičemž spolupracuje s firemním systémem ABRA Gen. První část práce se věnuje analýze stavu před zlepšením procesů, kde popisuje stav firmy a její možnosti. Druhá část se zabývá návrhem budoucího stavu a další část implementací samotné aplikace, která je realizována pomocí jazyka React Native. Poslední část se věnuje testování, návrhu dalšího vývoje a shrnutí práce.

Klíčová slova: Vývoj mobilní aplikace, ABRA Gen, React Native, API, Expo CLI

Vedoucí: Ing. Pavel Náplava, Ph.D
České vysoké učení technické v Praze,
Fakulta elektrotechnická,
Technická 2, B2-39d
160 00 Praha 6

Abstract

The bachelor's thesis focuses on the analysis, design and implementation of a mobile application, whose purpose is to improve processes of controllers in warehouse of Country life company. The mobile app provides an overview of customer orders to be delivered and a way to plan and manage those orders, while cooperating with ABRA Gen system. The first part of the thesis examines the current state before process improvement and the company options. The second part deals with the design of the future state, followed by the implementation of the application itself, which is developed using the React Native language. The final part describes testing, proposal of a further development, and summary of the work

Keywords: Mobile app development, ABRA Gen, React Native, API, Expo CLI

Title translation: Design of process improvement in Country life company

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Moderní doba je plná změn a nutí firmy digitalizovat a zavádět technologická řešení pro zlepšování jejich procesů. Firma Country life řeší stejný problém, a sice ten, že některé její procesy, konkrétně procesy týkající se dispečerů a kurýrů v jejím velkoskladě, jsou pomalé a závislé na lidském faktoru. Jelikož tyto procesy stojí firmu poměrně velké množství zdrojů a vznikají při nich chyby, stojí firma o jejich zlepšení pomocí mobilní aplikace.

Cílem této práce je reagovat na tento požadavek prozkoumáním možností firmy Country life a možností systému ABRA Gen, který používá. Dále pomocí nástrojů softwarového inženýrství navrhnout řešení a popsat budoucí stav procesů a aplikace s důrazem na její vizuální návrh ve formě prototypu.

Dalším cílem je tento návrh realizovat implementací v jazyce React Native a poté celou aplikaci ve spolupráci s firmou otestovat, zhodnotit a nasadit.

Mojí motivací pro výběr této práce byla zajímavá příležitost vývoje aplikace pro firmu, která pomůže některým jejím zaměstnancům v každodenní práci. Dále mojí motivací byla příležitost zdokonalit se v několika odvětvích softwarového vývoje, naučit se nové technologie, které se na trhu práce používají a poskytnout čtenářům vhled do toho, jaká úskalí může nést projekt v podobném rozsahu, jako je tento.

Kapitola 2

Stávající stav organizace

2.1 Obecné informace o firmě

Firma Country life (dále budeme používat označení firma) byla založena v roce 1991 Otakarem Jiránkem s cílem zaplnit díru na trhu s biopotravinami a zdravou výživou. Již 31 let se jí daří na tomto poli růst nejen po celém Česku, ale od roku 2017 i na Slovensku. Hlavním předmětem její činnosti je nákup a prodej zboží a vlastních výrobků, přičemž se více se soustředí na B2B trh než na B2C. Za rok 2021 měla firma dle její účetní uzávěrky obrat 461 233 tisíc Kč.[1]

Firma se snaží využít vymoženosti moderní doby a mít sklad digitalizovaný, proto například pro řízení skladu používá technologii WMS, u které se dle výzkumu Straits research očekává v roce 2030 velikost trhu asi 10,26 miliard dolarů a roční růst 15,3 %, z čehož se dá usoudit, že jde o populární technologii.[2]

Firma ke své činnosti využívá i několik softwarových řešení, z nichž pro ni nejdůležitějším řešením je ERP systém ABRA Gen od firmy ABRA Software a.s.[3]

2.2 Rozsah práce

Dříve, než se budeme zaobírat samotným zadáním firmy, je třeba popsat rozsah práce, respektive popsat, u kterých procesů má firma zájem o jejich zlepšení a u kterých nikoliv.

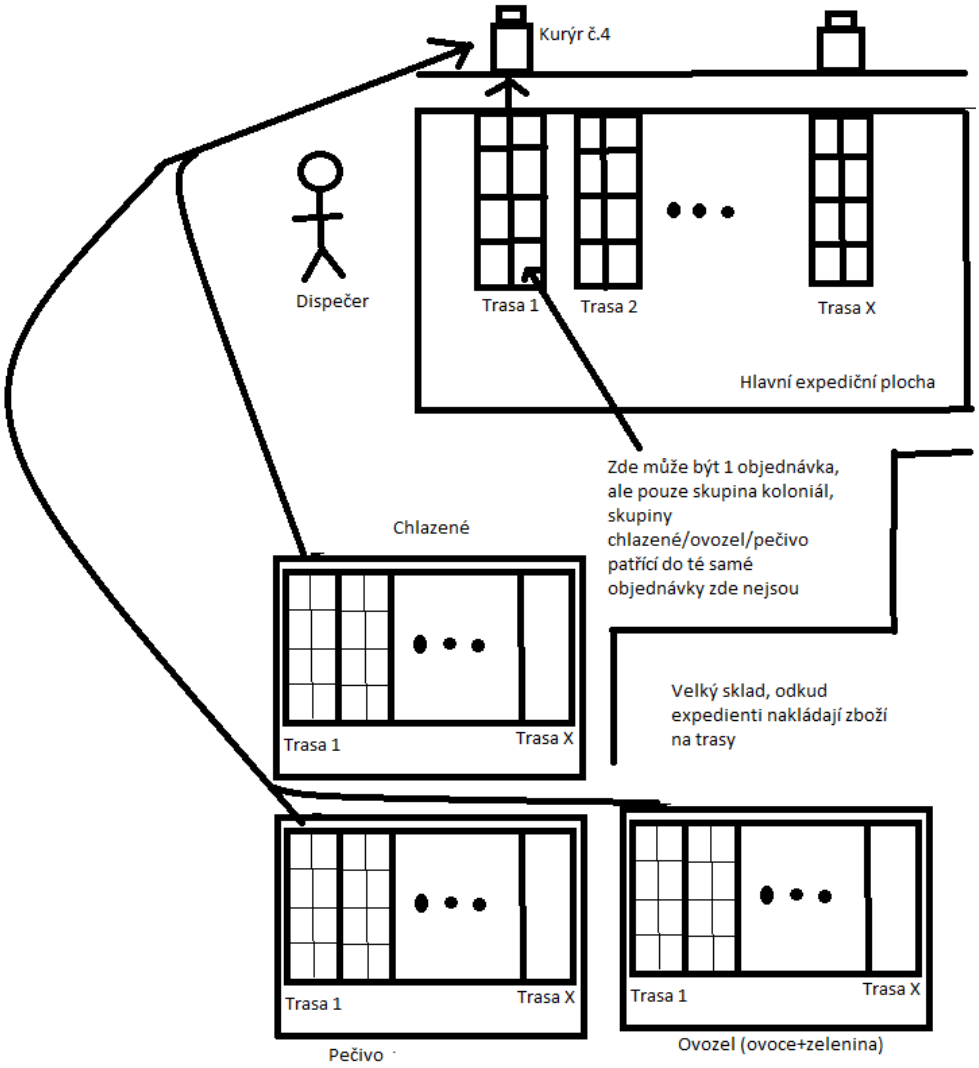
Pole působnosti, kde jsme se s firmou dohodli, že budeme analyzovat, jsou veškeré procesy týkající se dispečerů a kurýrů pracující ve firemním velkoskladě. Z pohledu procesního flow objednávky jde o procesy začínající od fáze, kdy expedient vyexpeduje objednávku až po doručení této objednávky zákazníkovi kurýrem. Předchozí procesy, tedy procesy od přijetí objednávky od zákazníka po vyexpedování objednávky jsou vyřešeny WMS technologií a fungují velmi efektivně, proto nemá smysl se na ně zaměřovat.

Firma již ze začátku přišla s tím, že pro zlepšení těchto procesů týkajících se dispečerů a kurýrů, kde již není použita technologie WMS, chtějí softwarové řešení, proto veškerá následná komunikace k takovémuto typu řešení směřovala.

Firma si předpřipravila i konkrétní funkcionality, které by potenciální aplikace zlepšující tyto procesy měla obsahovat. Než si je ale popíšeme, je třeba pochopit, jak nyní fungují procesy, které budeme zlepšovat.

2.3 Současný stav procesů dispečera a kurýrů na skladě

Pro lepší pochopení procesů, které budou později popsány může sloužit ilustrační obrázek 2.1.



Obrázek 2.1: Ilustrační obrázek fungování dispečera a kurýra na skladě

Pomocí obrázku jsme chtěli zvizualizovat fungování dispečera a kurýra v rámci skladu. Z obrázku je vidět, že kterýkoliv kurýr může přijet ke kterékoliv z tras na hlavní expediční ploše, odkud naloží objednávky do auta. Dispečer

se pak stará o to, aby se správně rozdělily objednávky mezi trasy a pokud je na trase příliš mnoho objednávek, tak je přerozděluje.

Zároveň lze vidět, že objednávka může obsahovat více skupin produktů, například objednávka může obsahovat ovoce/zeleninu (skupina "ovozel"), chlazené produkty (skupina "chlazené"), "koloniál" (tyto produkty se dávají na hlavní expediční plochu) a pečivo (skupina "pečivo"). Každá část objednávky příslušící některé z těchto skupin patří do stejné trasy.

Momentální procesy na skladě lze tedy rozdělit na workflow dispečera a workflow kurýra.

Dispečer má na starosti předání objednávek z tras kurýrům do auta. Tyto objednávky v případě potřeby přeskládává do jiných tras, pokud by auto nemohlo přijmout všechny objednávky z trasy pro něj připravené. Dispečerů nepracuje více najednou.

Momentální workflow dispečera z pohledu flow objednávky začíná, když expedient vyexpeduje objednávku na některou z tras. Poté pokračuje takto:

1. Dispečer kontroluje, zda je na všech trasách takový počet objednávek (respektive taková celková hmotnost/objem objednávek), který budoucí kurýr dokáže do svého auta přijmout.
2. V případě potřeby přesunu objednávky na jinou trasu tento přesun realizuje a domluví se s kurýrem, že musí doručit objednávku navíc. Na zaznamenání změny trasy v systému pomocí počítače v drtivé většině případů nemá čas, tento proces je příliš dlouhý.
3. Dispečer vytiskne 2 papíry, na kterých je seznam všech objednávek na trase, které řidič musí dovést.
4. Dispečer oba papíry podepíše, jeden předá kurýrovi, druhý uloží do skladu.
5. Dispečer zkontroluje, že na trase žádná objednávka nezbývá a všechny objednávky byly předány řidiči.

Kurýr má na každý den naplánovaný rozvoz a svoz objednávek. Objednávky jsou předem seřazené dle optimální trasy. Má tedy za úkol naložit objednávky z tras do auta a doručit je zákazníkům. Zároveň, má-li naplánovanou svozovou objednávku, musí ji doručit do skladu.

Workflow kurýra vypadá takto:

1. Kurýr si vybere aktivní trasu ke které přijede, aby mohl rozvést objednávky.
2. Kurýr naloží objednávky a přijme papír se seznamem objednávek podepsaný dispečerem.
3. Kurýr rozveze objednávky zákazníkům uvedeným v seznamu, přijme svozové, v případě, že zákazníci/dodavatelé musí doplácet, tak vybere peníze.

4. Kurýr přiveze svozové objednávky do velkoskladu a odevzdá peníze vybrané v hotovosti.

2.3.1 Požadavky firmy

Jak bylo výše zmíněno, firma Country life od začátku počítala s tím, že její momentální procesy popsané výše zlepšíme pomocí softwarového řešení. Firma si i předpřipravila **6 funkčních požadavků**, které by měla aplikace splňovat:

1. Dispečer potřebuje mít přehled o objednávkách a jejich stavech na expediční ploše, aby odváděl práci kvalitněji - velký obrázek, který se bude promítat na stěnu/plátno.
2. Dispečer potřebuje mít možnost rychle načíst objednávku(přes QR kód) a:
    a. Zkontrolovat ji a poslat tuto informaci do systému, aby se snížila nutnost papírování a v ABRA byly korektní aktuální data (management to chce).
    b. Přeřadit ji v systému na jinou trasu, aby byly v systému korektní data a mohlo se tak například získat korektní seznam objednávek určený kurýrovi.
3. Firma potřebuje kurýrovi dát korektní přehledný seznam doručovacích míst společně s dalšími údaji přes mobilní zařízení, aby se omezilo papírování.
4. Dispečer potřebuje být upozorněn na skutečnost, že objednávky na trase se nemohou naložit(přesáhl se limit hmotnosti/objemu auta) a potřebuje nápovědu, kam kterou objednávku přeřadit, aby ušetřil čas on i kurýr a nedocházelo k přetížení vozidla.
5. Firma potřebuje zjišťovat, kdy byli kurýři kde odevzdat svou objednávku, tato data mají pro ni hodnotu (o tomto požadavku se mělo dále diskutovat, není jisté, zda by kurýři vždy po odevzdání objednávky odklikli v ten moment v aplikaci skutečnost, že objednávku dodali).
6. Firma potřebuje rezervační systém ramp (kde budou kurýři vědět, kdy přijede které auto ke které trase, jak dlouho tam bude), aby nedocházelo ke sporům, čekání na rampu a dispečer věděl, kdy, kdo a kam přijede.

Požadavky jsou seřazené dle priority, tedy hodnoty, kterou firma zhodnotila, že jí daný požadavek přinese.

Na základě těchto požadavků jsme se po diskuzi s firmou rozhodli pro řešení ve formě 3 oddělených aplikací

1. Aplikace určená dispečerům, kterou budou používat z mobilního telefonu pro operace s objednávkami a získávání podrobnějších informací.

2. Aplikace určená kurýrům, kterou budou používat z mobilního telefonu pro operace s naloženými objednávkami a získávání dodatečných informací o nich.

3. Aplikace určená dispečerům ve formě velké obrazovky, která se bude promítat projektorem u místa tras s jediným účelem: zobrazit přehledový dashboard objednávek na trasách s dodatečnými užitečnými informacemi.

Firma vznesla také jeden nefunkční požadavek, který je pro ni stěžejní. Jde o požadavek, aby aplikace byla co nejjednodušší a nejintuitivnější. Aplikaci totiž budou používat lidé, kteří mají tendenci se bránit novým změnám.

Nyní jsme si popsali firmu, její současné procesy, které chce zlepšit a její požadavky, které by měly výsledné aplikace zlepšující tyto procesy splňovat. V následující části této kapitoly si detailně popíšeme technické prostředí, konkrétně softwarové ERP řešení ABRA Gen od firmy ABRA software a.s., které firma Country life využívá.

2.4 ABRA Gen

V této části si nejprve uvedeme základní informace o firmě ABRA software a.s. a poté konkrétně popíšeme systém ABRA Gen. Uvedeme si základní technické informace o systému a o databázi, kterou pro svoji činnost využívá a následně se budeme věnovat systémem poskytovanému Web API, kde si ukážeme, jak jsme jej spustili a otestovali jeho funkčnost. Nakonec si popíšeme dotazovací jazyk, pomocí kterého lze s Web API komunikovat.

2.4.1 Základní informace o ABRA Software a.s.

ABRA Software a.s. je technologická akciová společnost působící v IT průmyslu, jejíž majoritní podíl získala v roce 2022 německá investiční skupina Elvaston Capital[4]. Do Česka, Slovenska a Švýcarska dodává ERP informační systémy různých velikostí, které přizpůsobuje klientům na míru. Mezi 2 hlavní produkty, které dodává, patří systém ABRA Gen, sloužící zejména velkým firmám, do kterých spadá i firma. Druhým balíčkem je ABRA Flexi, přizpůsobující se spíše malým firmám.

Smyslem produktů od ABRy je poskytnout jednotné rozsáhlé IT řešení firmám řešící běžné procesy, které lze dělat efektivněji, když jsou digitalizované. Jde například o běžný provoz, účetnictví, komunikaci a mnoho dalšího.[5]

2.4.2 Technické informace

Systém ABRA Gen je založen na třívrstvé architektuře Client/Server a byl vytvořen v integrovaném grafickém vývojovém prostředí firmy Delphi. Je navržen tak, aby zvládl práci většího počtu uživatelů v jednom čase. Systém disponuje databází a rozhraním Web API, které s touto databází může komunikovat skrze jakýkoliv systém podporující http komunikaci. Může tak jít nejen o server, ale i mobilní zařízení, smartwatch, či výrobní linku.

2.4.3 Databáze

Data systém spravuje relační transakční databázi ve variantách pro Oracle, MSSQL nebo pro multiplatformní Firebird. Firma Country life využívá variantu pro Oracle. Základní tabulky jsou v těchto databázových systémech již předpřipravené, ABRA je při zakoupení připravuje na míru, proto zákazníci nemusí tvořit univerzální tabulky typu ”přijatá objednávka”, ”faktura”, což má mimo jiné i tu výhodu, že jsou tyto tabulky již napojené na systém a firma nemusí řešit jejich propojování.

Mimo tyto předpřipravené tabulky je možnost vytvořit vlastní, kterým aby fungovaly s Web API, musí být při jejich vytváření či úpravě v souboru DefRollEditor.exe přiděleno jméno business objektu, pod kterým je lze dotazovat. Nyní si toto Web API popíšeme podrobněji.

2.4.4 Web API

ABRA Web API je rozhraní postavené na REST principech, skrze které je možné komunikovat s databází prostřednictvím http nebo https požadavků. API se dotazuje na business objekty, v databázi znázorňované jako tabulky. Předpřipravené tabulky od ABRy již název business objektu mají.

Struktura http dotazů je ve většině faktorech intuitivní pro uživatele, kteří mají zkušenosti s SQL jazykem. Ovšem, jak bude popsáno níže, jsou zde výjimky, které si vývojář musí nastudovat.

Nejprve je ale nutné nakonfigurovat a spustit Web API, nejlépe v testovacím prostředí. Naštěstí měla firma pro tyto účely verzi ABRy pro testovací účely, která obsahovala i demodata, proto stačilo jen získat přístupové údaje a spustit Web API.

2.4.5 Spuštění Web API pro testovací účely

Pro spuštění Web API pro testovací účely jsme museli provést následující kroky:

- Nainstalovat demoverzi ABRA Gen a zajistit spojení s demodaty.
- Přidat dle existující šablony a upravit konfigurační soubor APIServer.yaml, aby naslouchal na volném portu.
- Skrze Windows Powershell vejít do složky, kde je nainstalována ABRA a spustit skript APIServer.ps1. Pro testovací účely jsme spouštěli s parametrem app, jelikož se při zadání tohoto parametru po ukončení API aplikace vypne a nezatěžuje dále server. Web API jsme tak spustili.

Pro posílání požadavků na toto spuštěné Web API jsme pak museli provést tyto kroky:

- Obstarat si http klienta, s kterým bylo možné posílat http požadavky na Web API (například Postman, Insomnia) a zajistit přístup k tomuto API.

- Zjistit alias spojení, které se v URL http požadavku píše za adresu spojení (například http://localhost:1234/alias spojení/...) - ke zjištění slouží spustitelný soubor DBAdmin.exe.
- Přidat v aplikaci ABRA Gen uživatele s nastaveným flagem "Přihlášení nevizuálního uživatele API".
- V http klientovi nastavit autorizaci na Base64 a vyplnit uživatelské jméno a heslo.
- Nastavit URL ve formátu {protokol}://{adresa spojení}:{port}/{alias spojení}/dotaz.

Takto nastavené prostředí bylo již funkční a stačilo jen otestovat funkčnost Web API pomocí testovacího požadavku.

2.4.6 Testovací požadavek

Pro vyzkoušení poslání požadavku a tím i otestování celé funkčnosti jsme za dotaz (uvedený za aliasem spojení) dosadili "receivedorders". Celý požadavek tak vypadal takto:

```
"http://localhost:1234/aliascl/receivedorders".
```

Tento způsob je uveden v technické dokumentaci ABRy, nicméně jelikož jsme se takto ptali na všechny přijaté objednávky (včetně všech historických), tak požadavek trval příliš dlouho.

Proto jsme přidali za parametr dotazu "take=1", který vrátil jen první záznam přijatých objednávek. Dotaz pak vypadal takto:

```
"http://localhost:1234/aliascl/receivedorders?take=1".
```

Tento požadavek proběhl mnohem rychleji(do jednotek sekund). Níže si vysvětlíme, proč takovýto dotaz funguje a jakým způsobem se na Web API dotazovat.[6]

2.4.7 Dotazovací jazyk

Při dotazování na Web API chceme provádět CRUD operace nad tabulkami v databázi. Je to jediný způsob, jak získat a měnit data. Tyto operace ale lze provádět jen nad takovými tabulkami, které jsou zároveň business objekty, tedy kterým bylo při přidání tabulky nebo její editaci přidáno jméno business objektu. Dotazovací jazyk je, jak již bylo zmíněno, zpravidla intuitivní pro uživatele obeznámené s SQL jazykem, ovšem, zmíním několik výjimek.

Dotazovat na data se lze **dvěma základními způsoby**:

- V základní variantě
- V rozšířené variantě

Dotazování v základní variantě

Požadavky v základní variantě se nejčastěji používají pro jednoduché dotazy, kde chceme získat informaci pouze z jedné tabulky. Při používání tohoto typu dotazu píšeme po aliasu spojení lomítko a jméno business objektu (například "/receivedorders"). Toto pravidlo je obdobou příkazu "FROM" v SQL jazyce. Celý takovýto dotaz:

```
"http://localhost:1234/aliascl/receivedorders"
```

by v SQL jazyce by vypadal nějak takto:

```
"SELECT * FROM receivedorders."
```

Dále můžeme přidávat URL-zakódované query parametry, kde máme na výběr z parametrů "select", "where", "expand", "groupby", "orderby", "skip" a "take".

Všechny parametry píšeme za otazník ve formátu "klíč=hodnota" a oddělujeme znakem "&". Mezerník nahrazujeme znakem "+" a rovnítko znakami "eq". Pro příklad, následující SQL dotaz:

```
"SELECT ID, CREATEDATDATE FROM receivedorders WHERE ID='1'"
```

by v dotazovacím jazyce v našem příkladu vypadal následovně:

```
"http://localhost:1234/aliascl/receivedorders?SELECT=ID,
CREATEDATDATE&WHERE=ID+eq+1."
```

Většina parametrů je známá z SQL jazyka a platí pro ně stejná pravidla. Jsou zde ale specifické parametry "expand", "skip" a "take".

Parametr expand slouží k rozvíjení dotazovaných business objektů, chceme-li se ptát na business objekty, které jsou propojené s jinými business objekty. Je to určitá náhrada za "JOIN" v SQL jazyce.

Pro příklad, mezi tabulkami v databázi nalezneme business objekty "receivedorders" a "receivedorders2". Na první pohled to může vývojáře mást, jelikož "receivedorders" jsou přijaté objednávky a "receivedorders2" jsou jednotlivé položky těchto přijatých objednávek, zatímco vývojář si může myslet, že jde o tabulku se stejnými sloupci.

Chceme-li tedy získat všechny položky jedné objednávky ("receivedorder"), musíme při dotazování nad přijatými objednávkami "receivedorders" přidat parametr "expand=Rows", díky kterému dotaz vrátí položky objednávky ("receivedorders2") vztahující se k dané přijaté objednávce ("receivedorder").

Parametr SKIP slouží pouze k přeskočení prvních "x" záznamů. Obdoba v SQL jazyce může být "OFFSET x ROWS".

Parametr TAKE slouží k vybrání pouze určitého množství záznamů. Obdoba v SQL jazyce by byla "FETCH NEXT x ROWS ONLY".

Oba parametry "SKIP" a "TAKE" jsou velmi užitečné při stránkování.

Dotazování v rozšířené variantě

Jsou situace, kdy potřebujeme pracovat s databází složitějším způsobem a dotazování v základní variantě nám nestačí, což je i náš případ. Pro tyto účely slouží rozšířené dotazování, které má určitá specifika. Dotaz je vždy "POST" a v URL adrese končí vždy znaky "/query", například:

`"http://localhost:1234/aliascl/query"`

nebo

`"http://localhost:1234/aliascl/receivedorders/query".`

V těle dotazu se píší parametry dotazu ve formátu "json". Parametry jsou stejné jako pro základní dotazování, jen s několika rozdíly:

- Přibývá parametr "class", což je obdoba pro "FROM" v SQL jazyce.
- Přibývá parametr "name" a "value", kde pomocí "name" se označí existující atribut tabulky a "value" pak v response vrátí výsledek dotazu, který se v tomto parametru specifikuje pod klíčem uvedeným v "name". Lépe se to pochopí na příkladu níže.
- Přibývá parametr "query", který vrátí data, která jsou výsledkem dotazu určeného pomocí parametrů jemu předané. Například "query" s hodnotou

  `{"select":"id", "take":"1"}`

  vrátí jedno ID dotazovaného business objektu.
- Mizí parametr "expand", jelikož lze nahradit parametrem "select".

Příklad takového těla dotazu může být:

```
{
  "class": "receivedorders",
  "select": [
    "createdatdate",
    {
        "name": "firm_id",
        "value": {
        "select": ["Name"],
        }
     }
  ]
  "where": "id='1'"
}
```

V tomto dotazu se ptáme na datum vytvoření přijaté objednávky a jméno firmy, která tuto objednávku vytvořila. Hledáme takový záznam, který má hodnotu "id" 1.

Díky dotazovacímu jazyku jsme schopni posílat přes REST rozhraní i ty nejobtížnější dotazy, díky kterým můžeme získat potřebná data z tabulek.

Dávkové zpracování

Kromě získání dat potřebujeme také umět upravit data. Proto se podíváme na možnosti dávkového zpracování, který prozkoumáme jen do takové míry, abychom věděli, jak jej použít pro naši aplikaci.

Princip dávkového zpracování spočívá v provedení několika http požadavků naráz s tím, že se provedou buď všechny požadavky(operace) nebo žádná. To se nám bude hodit v implementační části pro aktualizaci několika objednávek najednou. Požadavky tohoto typu se posílají na adresu:

```
".../aliascl/batch",
```

kde za 3 tečky dosadíme url a mají tělo dotazu s atributy:

- type - typ operace, například ”update” pro aktualizaci, ”create” pro vytvoření záznamu
- ID - identifikátor operace
- data - obsah operace, které popíši níže

Do obsahu operace se píší tyto atributy:

- class-id - business objekt (například receivedorders odpovídající stejnojmenné tabulce v databázi)
- object-id - při operaci ”update” značí id objektu(řádku v databázi), který chceme aktualizovat
- query - nepovinné, zadává se zde dotaz, který obvykle vrátí atributy změněného business objektu

Jako příklad uvedeme tělo dotazu sloužící ke změně trasy objednávek s ID: ”123ABC” a ”123ABD” na trasu s číslem 5 a vrátíme pro každou objednávku její ID:

```
{
    "items": [
        {
            "type": "update",
            "id": "id0",
            "data": {
                "class_id": "receivedorders",
                "obj_id": "123ABC",
                "object_data": {
                    "trace": "5"
                },
                "query": {
                    "select": [
                        "ID"
                    ]
                }
            }
        },
        {
            "type": "update",
            "id": "id1",
            "data": {
                "class_id": "receivedorders",
                "obj_id": "123ABD",
                "object_data": {
                    "trace": "5"
                },
                "query": {
                    "select": [
                        "ID"
                    ]
                }
            }
        }
    ]
}
```

Tímto jsme prozkoumali veškeré způsoby dotazování, které budeme v následující části bakalářské práce potřebovat. ABRA Gen API poskytuje ještě další možnosti dotazování, ale pro naše účely nám stačí jen tyto.

Požadavky, které posíláme ABRA Gen zpracovává tak, že je překládá do SQL jazyka, tudíž využíváme optimalizačních algoritmů tohoto jazyka, které byly uzpůsobeny tak, aby se požadované odpovědi vracely v co nejkratším čase.[7]

Nyní jsme zanalyzovali současný stav organizace, současné procesy, požadavky firmy a detailně popsali technické prostředí. Tuto analýzu jsme využili

pro návrh budoucího stavu, který podrobně rozebereme v následující kapitole.

Kapitola 3

Návrh budoucího stavu

V této kapitole se budeme věnovat návrhu budoucího stavu aplikace vycházející ze současného stavu a ze schůzek se zadavatelem (lze nalézt v elektronické příloze pod názvem schuzky.pdf). S ním jsme se také domluvili, že **nebudeme implementovat všechny 3 aplikace**, ale **budeme implementovat jen mobilní aplikaci pro dispečery.**

V rámci tvorby návrhu budoucího stavu a následované implementační části **se budeme řídit schválenými wireframy**, které jsou dostupné na tomto odkaze[8].

V kapitole popíšeme použité technologie, způsob distribuce, přínosy aplikace a poté případy užití. Usoudili jsme, že jiné diagramy než use-case diagram nemá smysl tvořit, jelikož je aplikace ve své podstatě velmi jednoduchá. Je to jen frontendová aplikace, která vykresluje a modifikuje data získané z Web API.

3.1 Výběr technologií

Před realizací mobilní aplikace jsme se museli nejprve rozhodnout, jaký typ aplikace chceme vytvořit, a to z následujících typů aplikace:

- Nativní aplikace - například Kotlin/Swift - výhoda této volby je v tom, že nativní aplikace bývají spolehlivější, bezpečnější a rychlejší. Zásadní nevýhoda je však ta, že vývojář musí vytvořit pro každý operační systém samostatnou aplikaci.
- Webová aplikace - například React - výhodou je použitelnost jak pro mobil, tak pro počítač, nevýhodou však, že webová aplikace je v mobilu méně uživatelsky přívětivá než aplikace mobilní.
- Hybridní - například Xamarin/React-native - zásadní výhodou hybridních aplikací je, že se aplikace nemusí vyvíjet zvlášť pro každou platformu. Nevýhodou je, že mívají horší parametry než kdyby se programovaly nativně (například rychlost, spolehlivost).

Z těchto nabízených technologií jsme se rozhodli pro použití hybridní aplikace, jelikož je pro úspěšnost projektu zásadní uživatelská přívětivost pro dispečery, kteří budou aplikaci používat a počet dodaných funkcionalit.

Pro realizaci hybridní aplikace se nám nabízí technologie Flutter, React Native a Xamarin, z nichž jsme se rozhodli pro React Native z důvodu popularity, velkého množství dokumentací a zkušeností s jazykem Javascript.

3.2 Způsob distribuce aplikace

Po dokončení aplikace je třeba výslednou mobilní aplikaci předat koncovým uživatelům, tedy nainstalovat aplikaci dispečerům a kurýrům do jejich mobilních telefonů, které budou používat.

Pro distribuci se nabízí platformy Google store pro Android a App store pro iOS, pomocí kterých lze také distribuovat aplikace specifikovaným uživatelům. Výhodou užití těchto platforem je například optimalizace velikosti aplikace, ale nevýhodou je nutnost procesu kontroly aplikace, zdlouhavost distribuce a tyto platformy navíc nejsou zdarma.

Kromě této možnosti se také nabízí možnost pro Android (u iOS je proces složitější), a sice distribuce .apk souboru, který se dá po stažení do mobilního zařízení přímo nainstalovat a spustit. Díky Expo CLI by také šlo aplikaci kdykoliv vzdáleně aktualizovat (v případě opravování chyb, dodávání nových funkcionalit). Jedinou nevýhodou je to, že by tento soubor zabíral velké množství místa v paměti.

Jelikož lze všem budoucím uživatelům zajistit operační systém Android a druhá možnost z výše zmíněných je méně pracná, tak jsme si vybrali tuto možnost, tedy distribuci .apk souboru e-mailem.

3.3 Přínosy navržené aplikace

V rámci projektu je dobré znát přínosy navrženého řešení nejen pro účely zdůvodnění smyslu projektu stakeholderům, ale i proto, abychom při tvorbě use casů mysleli na jejich realizaci a po implementační části si zhodnotili, že jsme je dokázali naplnit. Podle diskuzí s ředitelem IT jsme došli k tomu, že výsledné řešení bude mít tyto přínosy:

3.3.1 Přínosy pro business

Hlavní přínos pro business bude ušetření času dispečerů. Po zavedení řešení dispečeři nebudou muset komunikovat s kurýry změny tras. Zároveň budou mít lepší přehled o trasách, což bude vést nejen ke zjednodušení práce - tudíž k nižší chybovosti, ale řešení pak pomůže například odhalit některé nesrovnalosti (například objednávka má být na jiné trase) s předstihem. Firma bude po nasazení také lépe připravena řešit procesy plně elektronicky a ne pouze papírovou formou.

3.3.2 Přínosy pro IT

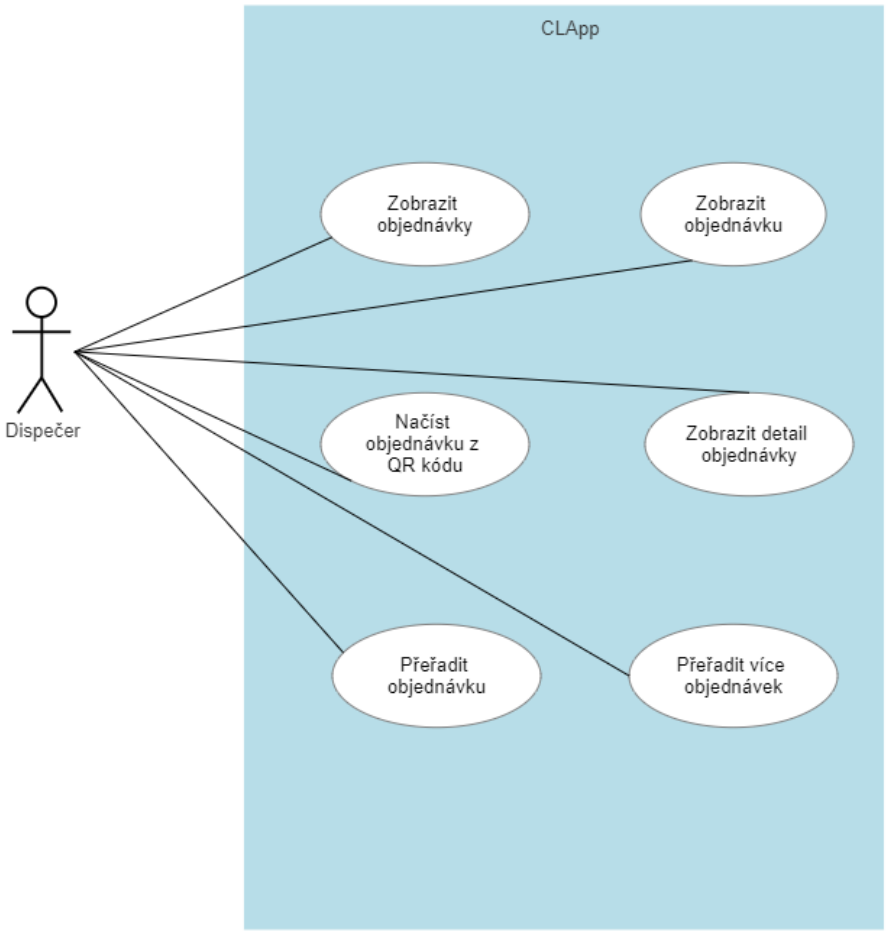
Přínosy pro IT budou spočívat zejména v tom, že v systému budou korektní data, které bude možné použít na další rozvoj. Dále se dispečeři naučí používat

při práci aplikaci, do které bude následně možné doimplementovat další funkcionality, které jim budou moci usnadnit práci.

3.4 Případy užití

V rámci této sekce si ukážeme všechny případy užití aplikace, ze kterých následně vybereme ty nejklíčovější a ty popíšeme podrobně. Tento podrobný popis nám poslouží pro následovanou implementační část.

3.4.1 Use case diagram



Obrázek 3.1: Případy užití

Na obrázku 3.1 vidíme případy užití, z nichž jsme se rozhodli podrobněji popsat případy ”zobrazit objednávky”, ”zobrazit objednávku” a ”přeřadit objednávku”, jelikož jsou klíčové a implementace ostatních případů užití se od těchto příliš neliší.

Ke každému z těchto případů užití ukážeme i wireframe/wireframy, podle kterých se bude daný případ užití implementovat. Tyto wireframy jsou již schválené zadavatelem. Jejich předchozí verze a způsoby, jakými se ladily do těchto finálních podob jsou popsány v elektronické příloze pod názvem ”schuzky.pdf”.

3.4.2 Zobrazit objednávky

Wireframe k tomuto use case lze vidět na obrázku 3.2.

Obrázek 3.2: Wireframe – seznam objednávek

Scénář:

0. Vstup - aplikace se spustila nebo uživatel změnil den, na který chce zobrazit objednávky.

1. Aplikace zavolá http request na Web API s tělem dotazu:

```
{
  "class": "receivedorders",
  "select": [
    "trace_id", //číslo trasy
    "state_id", //id stavu - například připravuje se, připravená atd.
    "ORDNUMBER",
    { "name": "FirmOffice_ID",
        "value": {
        "select": [
            "Name" //název provozovny (firm_name)
            { "name": "ADDRESS_ID",
            "value": {
            "select": ["City"] //město provozovny
            }}]
        }
    },
    {
        "name": "DOCQUEUE_ID",
        "value": {
            "select": ["Code"]
        }
    },
    {
        "name": "PERIOD_ID",
        "value": {
            "select": ["Code"]
        }
    },
    { "name": "Rows",
      "value": {
      "query": {
          "select": ["DELIVERYDATE$DATE"],
          "take" : 1
      }
      }
    }
  ],

  "where": "exists(Rows WHERE DELIVERYDATE$DATE=timestamp'2023-06-31')",
}.
```

   - V dotazu namísto data (2023-06-31) se dosadí aktuální den. Během čekání na response pak musí uživatel vědět, že toto čekání probíhá (například hláška "probíhá načítání objednávek" nebo točící se spinner).

- Jestliže se nevrátí pozitivní response (not response.ok), vypíše se chybová hláška, jinak se response namapuje na objekt dle datového modelu níže do objektu ”Order” s výjimkou, že nemapujeme atributy: groceries, chilled, fruitVeg, pastry.

2. Objednávky se vykreslí dle wireframu v barvách, kde zelená=”state.id=20”, žlutá=”state.id=19”, zbytek bílá (tyto hodnoty se mohou kdykoliv změnit). Po klepnutí na selectbox se uživateli zobrazí trasy, ze kterých si bude moci vybrat pouze aktivní trasu (aktivní trasa je právě taková, pro kterou existuje objednávka s tou danou trasou a datem doručení na právě zvolený den). Objednávka, která nemá přiřazenou trasu (trace=”0” OR trace=””) je brána jako objednávka ”bez přiřazení”.
3. Uživatel zvolí v selectboxu trasu (”bez přiřazení”, ”trasa 1”, ”trasa 2”, ...).
4. Aplikace vykreslí jen objednávky příslušející zvolené trase.

3.4.3 Zobrazit objednávku

Wireframe k tomuto use case lze vidět na obrázku 3.3.

Obrázek 3.3: Wireframe – přehled objednávek

0. Uživatel klepl na objednávku ze seznamu objednávek.

1. Z parametrů obrazovky získáme identifikátor objednávky, pomocí kterého se zeptáme na data pomocí následujícího POST requestu:

```
{
    "class": "receivedorders",
    "select": [
    "trace_id", //číslo trasy
    "state_id", //id stavu - například připravuje se, připravená atd.
    "ORDNUMBER",
    "isPastry",
    { "name": "FirmOffice_ID",
        "value": {
        "select": [
            "Name" //název provozovny (firm_name)
            { "name": "ADDRESS_ID",
            "value": {
            "select": ["City"] //město provozovny
            }}]
        }
    },
    {
        "name": "DOCQUEUE_ID",
        "value": {
            "select": ["Code"]
        }
    },
    {
        "name": "PERIOD_ID",
        "value": {
            "select": ["Code"]
        }
    },
    {
        "name": "Rows",
        "value": {
            "query": {
                "select": [
                    "DELIVERYDATE$DATE",
                    "ROWTYPE",
                    "QUANTITY",
                    {
                        "name": "STORECARD_ID",
                        "value": {
                            "select": ["Code"]
                        }
                    }
                ]
```

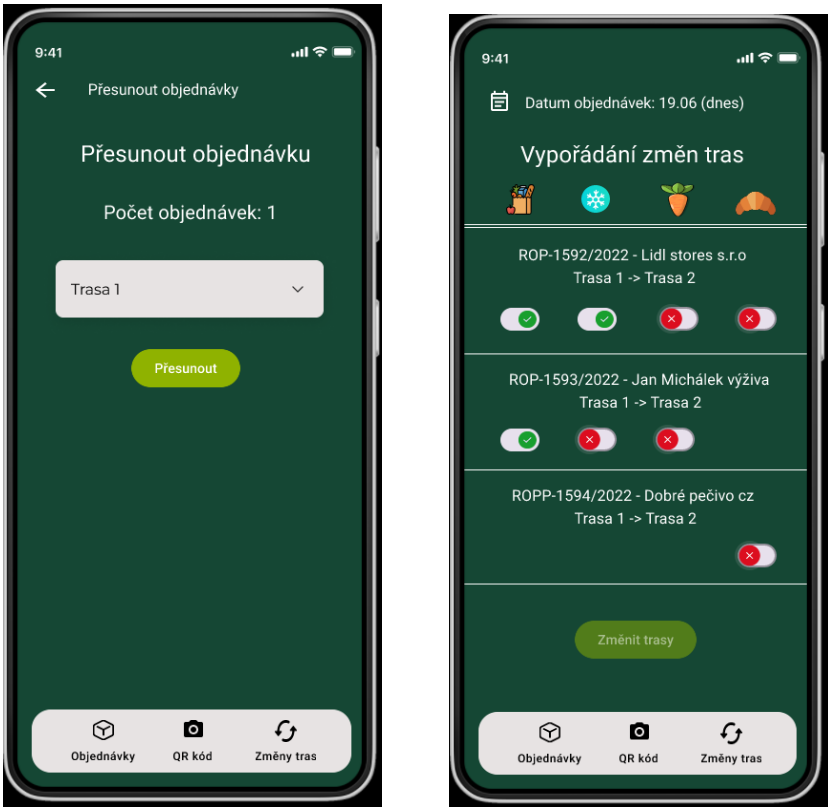
```
                }
            }
        }
        ],

    "where": "ID eq '123456ABC'"

    }
```

Místo '123456ABC' patří skutečné ID objednávky, u které chceme vidět přehled.

2. Získané údaje z response namapujeme dle datového modelu níže do objektu "Order" a zobrazíme data dle wireframu.

3.4.4 Přeřadit objednávku

Wireframy k tomuto scénáři lze vidět na obrázku 3.4



(a) : Změna trasy **(b) :** Vypořádání změny tras

Obrázek 3.4: Změnit trasu

Scénář:

0. Uživatel klepl na tlačítko ”přeřadit objednávku”.
1. Zobrazí se obrazovka (a) změna trasy.
2. Uživatel vybere trasu, na kterou chce objednávku přeřadit a zvolí ”Přesunout”.
3. Zobrazí se obrazovka (b) Vypořádání změny tras. Tato obrazovka zobrazí 4 ikonky reprezentující ”koloniál”, ”chlazené”, ”ovozel” a ”pečivo”. Pod ní se zobrazí všechny objednávky, u kterých se uživatel rozhodl změnit trasu, a to ve formátu: Název objednávky, Název provozovny, z které trasy do které je objednávka přesouvána a switch buttony reprezentující přesunutou skupinu produktů.
4. Uživatel zaklikne všechny switch buttony (a fyzicky přesune všechny části všech objednávek).
5. Rozsvítí se tlačítko ”Změnit trasy”, aby uživatel věděl, že na nic nezapomněl a všechny trasy přesunul.
6. Uživatel změní trasy.
7. Systém pošle POST request na adresu ”.../aliascl/batch” s tělem dotazu dle. příkladu zde
8. Zobrazí se hláška, zda se podařilo přesunout objednávky či nikoliv.

Tímto máme popsanou celou kapitolu návrhu budoucího stavu, která nám poslouží pro implementační část. Vybrali jsme technologii React Native a rozhodli jsme, že výslednou aplikaci budeme distribuovat jen pro Android přes e-mail jako .apk soubor (přičemž stále budeme moci uživatelům vzdáleně aktualizovat aplikaci). Dále jsme popsali přínosy navržené aplikace a případy užití, ze kterých jsme vybrali 3 nejklíčovější a ty popsali detailně. Zbylé případy užití jsou principem velmi podobné těm vybraným, proto jsme je nepopisovali.

Nyní máme veškeré podklady k tomu, abychom mohli navázat implementační částí.

Kapitola 4

Implementační část

V této kapitole se budeme věnovat implementaci aplikace vycházející z předchozí kapitoly. Z důvodu rozsáhlosti projektu nemá smysl popisovat veškerý kód, proto se zaměříme pouze na důležité části, na kterých naše aplikace staví.

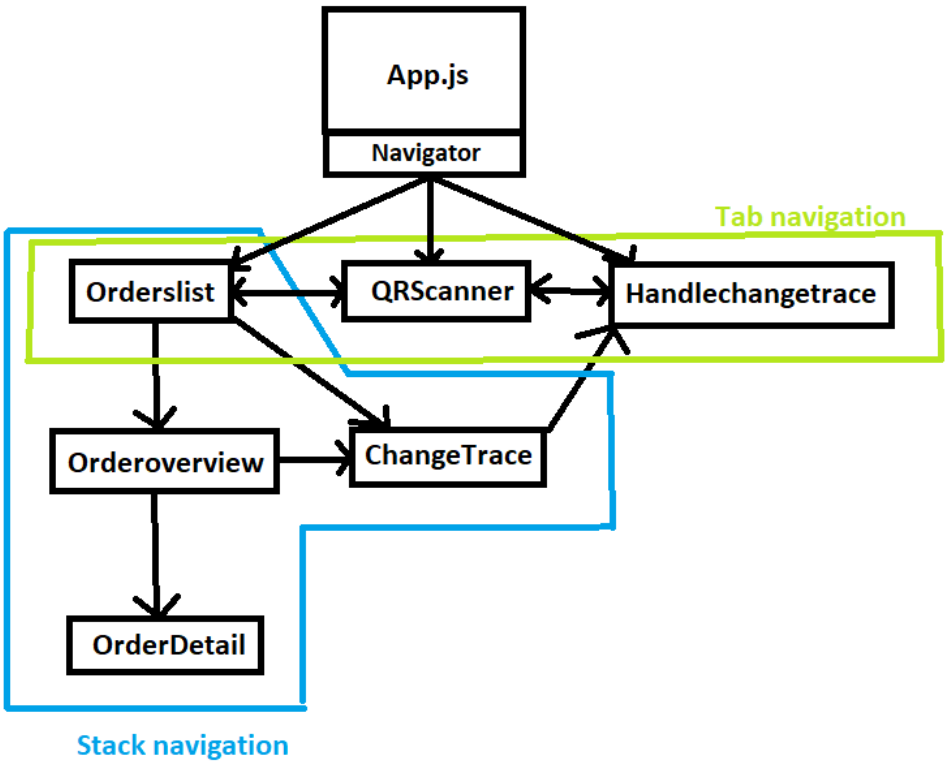
Nejprve si popíšeme architekturu aplikace, v níž se podíváme na strukturu React obrazovek, datový model a způsoby vývoje, pro které jsme se rozhodli.

Dále si popíšeme použité knihovny, které jsme v rámci implementace použili a poté postup implementace, kde popíšeme základní části, na kterých naše aplikace stojí.

4.1 Architektura aplikace

Pro popis frontendové architektury naší aplikace nemá smysl použít žádný z běžně používaných diagramů (komponentový diagram, sekvenční). Důvodem je jednoduchost aplikace - naše aplikace pouze zobrazuje data systému ABRA a upravuje je, přičemž pracuje se dvěma základními objekty, které budou dále v této kapitole popsané. Co již ale smysl má, je diagram aplikace z pohledu React komponent.

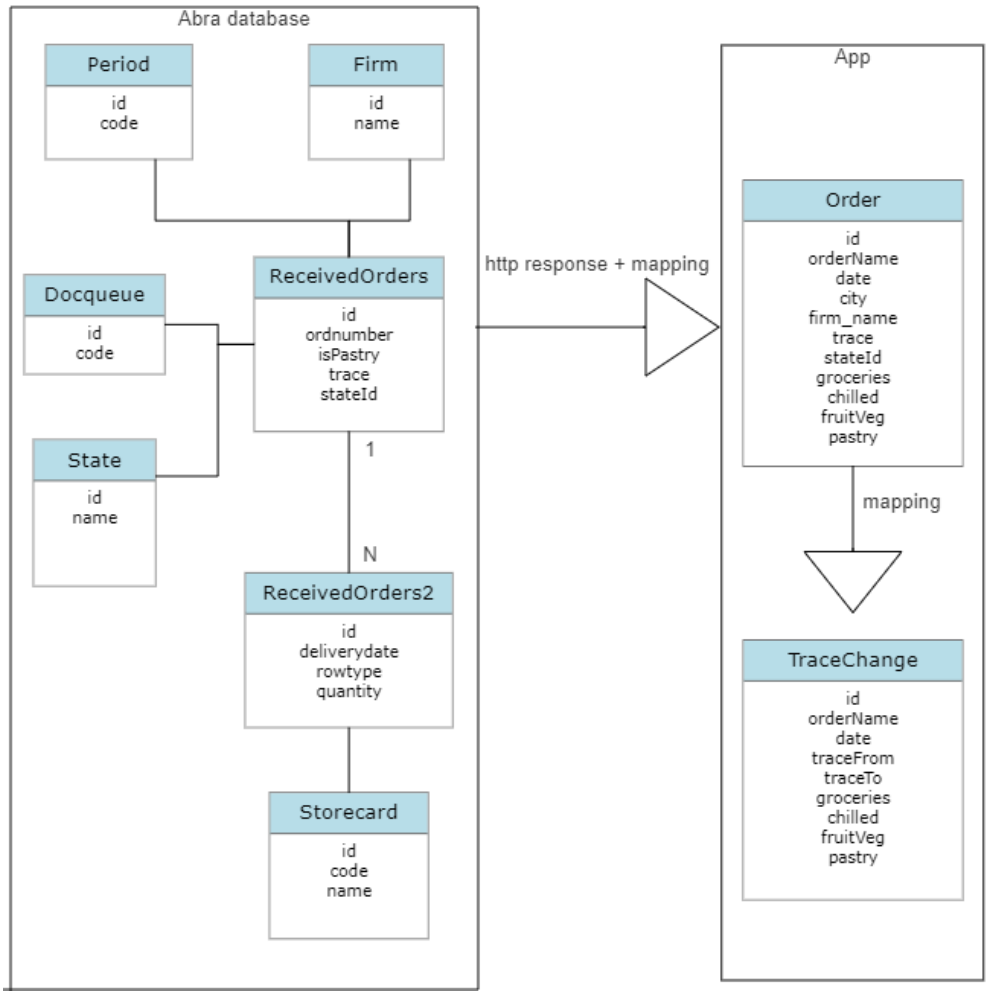
4.1.1 Struktura aplikace



Obrázek 4.1: Architektura obrazovek

Diagram na obrázku 4.1 popisuje architekturu React aplikace. Aplikace je složena z 6 různých obrazovek, mezi kterými navigátor přepíná obrazovky formou ”tab navigation” a ”stack navigation”. Pomocí ”tab navigation” přepínáme mezi přehledem objednávek, QR skenerem a vypořádáním objednávek (bottom tab navigator, který je vidět na všech obrazovkách, viz. wireframy[8]) a pomocí ”stack navigation” přepínáme mezi všemi ostatními obrazovkami.

4.1.2 Datový model



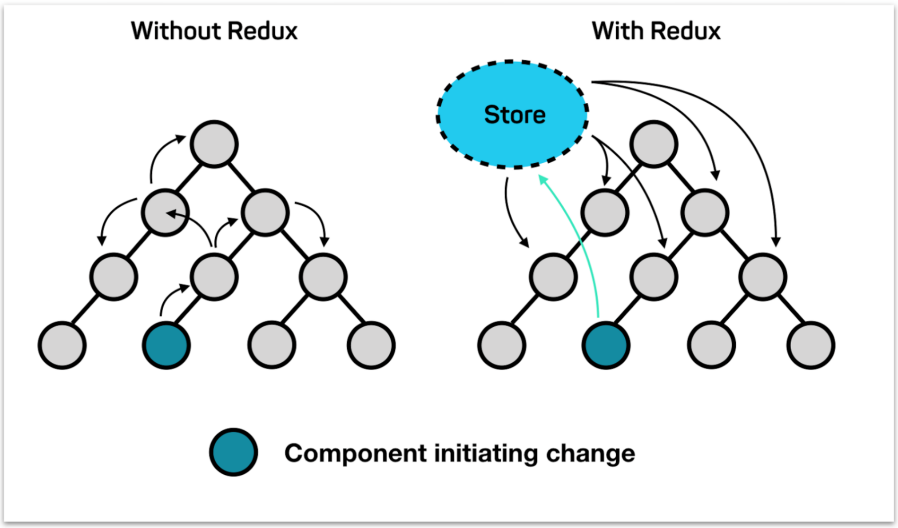
Obrázek 4.2: Datový model

Tento zjednodušený datový model popisuje databázové tabulky, se kterými pracujeme na straně Abry (levá část) a objekty, se kterými pracujeme v rámci aplikace (pravá část). Z datového hlediska se v naší aplikaci při získávání dat z ABRy děje to, že se aplikace doptává na hodnoty z tabulek ABRy a mapuje je do objektu "Order". Některé z atributů objektu "Order" stačí jen správně namapovat (id, trace, firm-name, ...) a jiné se musí z tabulek vydolovat podmínkami (orderName, groceries, chilled, ...). Objekt "TraceChange" reprezentuje změnu objednávky z jedné trasy do druhé a plní se hodnotami z "Order" a hodnotami zadanými uživatelem.

Nyní bychom se podívali konkrétněji, podle kterých podmínek se některé atributy objektu "Order" plní.

- **orderName** - Tento atribut slouží jako identifikátor objednávky pro zaměstnance a je použit například v případu užití zobrazit objednávky. Naplňuje se ve formátu: ”DOCQUEUE.Code-ORDNUMBER/PERIOD.Code”. Příklad může být ”ROP-12345/2023”.
- **groceries** - Tento atribut značí, zda objednávka obsahuje ”koloniál”. Je naplněn hodnotou true, pokud existuje taková položka objednávky, pro kterou platí: ro2.rowtype=8 and ro2.quantity > 0 and ro2.Storecard.code. substring(0,3)!=”180” and ro2.Storecard.code.substring(0,3)!=”185”.
- **chilled** - Atribut značící, zda objednávka obsahuje ”chlazené”. Naplněn hodnotou true, pokud existuje taková položka objednávky, pro kterou platí: ro2.rowtype=8 and ro2.quantity > 0 and ro2.Storecard.code. substring(0,1)!=”5”.
- **fruitVeg** - Atribut značící, zda objednávka obsahuje ”ovozel”. Naplněn hodnotou true, pokud existuje taková položka objednávky, pro kterou platí: ro2.rowtype=8 and ro2.quantity > 0 and ro2.Storecard.code. substring(0,3)=”180” or ro2.Storecard.code.substring(0,3)=”185”.
- **pastry** - Atribut značící, zda objednávka obsahuje ”pečivo”. Naplněn hodnotou receivedOrder.isPastry.

4.1.3 Kontextový přístup k implementaci



Obrázek 4.3: Kontextový přístup

Na obrázku 4.3 jsou zobrazeny 2 přístupy k řízení dat v aplikaci, a to přístup kontextový (bez Reduxu) a přístup s Reduxem. Obrázek chceme využít pro popsání rozdílů mezi těmito dvěma přístupy a důvodů, proč jsme se v rámci vývoje rozhodli právě pro přístup kontextový.

Výhody použití přístupu s Reduxem spočívají zejména v odseparování logiky pro řízení stavů a pro to, aby se při použití stejných dat na více místech změny nemusely propisovat vnořováním a vynořováním z komponent, jako je to zobrazeno na stromové struktuře vpravo. Tento přístup má ale nevýhodu, že je složitější a vyžaduje velké množství kódu jen za účelem toho, aby návrhový vzor fungoval. Jelikož je aplikace poměrně jednoduchá a neměla by se v ní stát situace, kde by se dle stromu vlevo propagovala změna dat přes větší hloubku než 2, přiklonili jsme se ke kontextovému přístupu (bez Reduxu).

4.1.4 React Native klient

Při vývoji v React Native existují 2 základní způsoby, jakými vyvíjet, a to buď za pomoci React Native klienta nebo Expo klienta.

React Native CLI

React Native CLI je oficiální nástroj pro vývoj React Native aplikací od původní firmy Facebook (nynější Meta). Oproti vývoje s Expo klientem je tento přístup flexibilnější, vývojář je nucen řídit a konfigurovat tam, kde by se při přístupu s Expo CLI konfigurovalo automaticky a může do kódu přidávat nativní moduly. Navíc, aplikace vyvinuté tímto způsobem zabírají méně místa v úložišti.

Expo CLI

Expo CLI je nástroj postavený na React Native CLI, který poskytuje určitou abstrakci nad zmíněným přístupem. Vývoj s Expo CLI je pro vývojáře jednodušší, velké množství konfigurací řeší Expo CLI za něj. Navíc je tomuto klientovi vytvořena aplikace Expo go, díky které může uživatel v reálném čase vidět změny, které vyvíjí a nepotřebuje po každé změně znovu spouštět emulátor. Vývojář tak může provádět základní testy funkčnosti již při vývoji.

Oba tyto přístupy mají své výhody a nevýhody, pro jednoduchost vývoje jsme si však vybrali Expo CLI, který je zároveň populárnější a doporučovanější pro začátečníky.[10]

4.2 Použité knihovny

V následující části si popíšeme knihovny, které jsme v implementační části použili.

4.2.1 Navigace

První knihovnou, kterou si popíšeme a která zároveň tvoří základ naší aplikace, je navigace. Jak již bylo popsáno v popisu struktury React aplikace, naše aplikace obsahuje 2 typy navigace, ”tab navigation” a ”stack navigation”.

(a) : Tab navigation **(b) :** Stack navigation

Obrázek 4.4: Použité navigace

”Tab navigation” funguje na jednoduchém principu, že při klepnutí (na ikonku) se zobrazí daná obrazovka. V naší aplikaci je do této ”tab navigation” vnořena ”stack navigation”.

”Stack navigation” si oproti ”tab navigation” ukládá obrazovky, které si uživatel zobrazil, do zásobníku, aby se v případě, že se chce vrátit na původní obrazovku jednoduše vrátil a to typicky klepnutím na šipku zpět v horním menu aplikace nebo v dolním menu mobilního zařízení.

Naše ”tab navigation” implementace vypadá takto:

```
export default function TabNavigator() {
    const Tab = createBottomTabNavigator();

    return (
        <Tab.Navigator>
            <Tab.Screen name="OrderslistTab"
            component={StackNavigator} options={{
                tabBarLabel: 'Objednavky',
                tabBarIcon: ({color, size}) => (
                    <Octicons name="package"
                    color={color} size={size}/>
                ),
            }}/>
            <Tab.Screen name="QrScanner"
            component={QrScanner} options={{
                tabBarLabel: 'QR skener',
                tabBarIcon: ({color, size}) => (
                    <MaterialIcons name="qr-code-scanner"
                    color={color} size={size}/>
                ),
            }}/>
            <Tab.Screen name="HandleChangeTrace"
            component={HandleChangeTrace} options={{
                tabBarLabel: 'Zmeny tras',
                tabBarIcon: ({color, size}) => (
                    <MaterialIcons name="autorenew"
                    color={color} size={size}/>
                ),
            }}/>
        </Tab.Navigator>
    }
```

V kódu vidíme ”tab navigator” složený ze 3 obrazovek s popiskem: ”Objednávky”, ”QR skener”, ”Změny tras”. Každá z těchto 3 obrazovek (Tab.Screen) obsahuje:

- name - popisek, kterým se odkazujeme na obrazovku, chceme-li se na ni navigovat z jiné obrazovky.
- component - samotná obrazovka (funkce, která vrátí komponentu, která se vykreslí na novou obrazovku).
- tabBarLabel - popisek zobrazený v dolním navigačním menu.
- tabBarIcon - ikonka (obrázek) zobrazená v dolním navigačním menu.

V první z těchto obrazovek je za ”component” dosazena komponenta StackNavigator. Pomocí ní docílíme toho, abychom mohli v ”tab navigation” mít ”stack navigation”. Tato komponenta vypadá takto:

```
export default function StackNavigator() {
const Stack = createStackNavigator();

return (
<Stack.Navigator>
<Stack.Screen name="Orderslist"
              component={Orderslist}
              options={{headerShown: false}}/>
<Stack.Screen name="OrderOverview"
              component={OrderOverview}
              options={{title: 'Prehled objednavky'}}/>
<Stack.Screen name="ChangeTrace"
              component={ChangeTrace}
              options={{title: 'Presun objednavek'}}/>
<Stack.Screen name="OrderDetail"
              component={OrderDetail}
              options={{title: 'Detail objednavky'}}/>
</Stack.Navigator>
);
}
```

V komponentě vidíme 4 obrazovky, jednu hlavní (Orderslist), z které se uživatel naviguje do ostatních obrazovek (OrderOverview, ChangeTrace, OrderDetail). Z těchto obrazovek se uživatel vždy může vrátit do předchozích zvolených obrazovek až do hlavní obrazovky ”Orderslist”.

Aby celá navigace fungovala, tak již stačí jen zaobalit TabNavigator v NavigationContainer takto:

```
<NavigationContainer>
  <TabNavigator />
</NavigationContainer>
```

4.2.2 Lokální databáze

Dále bychom si popsali knihovnu, která nám poslouží k implementaci lokální databáze. Knihovna se jmenuje Asyncstorage a budeme ji využívat k ukládání změn tras. K její instalaci použijeme následující příkaz:

```
npx expo install @react-native-async-storage/async-storage
```

Tato knihovna slouží jako key-value lokální databáze a má velmi jednoduchou implementaci. Data, která uložíme, jsou dostupná po zavření aplikace, ale po smazání aplikačních dat (lze provést například v nastavení telefonu) jsou nedostupná. Tento typ databáze se nehodí pro práci s velkým množstvím dat, jelikož má tato databáze limit (v nižších desítkách MB). Nás ale toto omezení nelimituje, protože pracujeme s malým množstvím dat. Příklad funkce, která uloží hodnotu "value" pod klíčem "key" do databáze vypadá takto:

```
const storeValue = async (value) => {
  try {
    await AsyncStorage.setItem('key', value)
  } catch (e) {
    // handle error
  }
}
```

V naší implementaci při získávání změn tras jsme implementovali tuto funkci:

```
export const getChangesFromStorage = async () => {
    try {
        const changes = await AsyncStorage.getItem("Changes");
        if (changes !== null) {
            return changes;
        }
    } catch (e) {
        console.error("get from storage error: " + e);
        alert("Nepodarilo se ziskat zmeny tras")
    }
    return [];
}
```

Tato funkce jednoduše získá změny tras, které byly uloženy pod klíčem "Changes".

4.2.3 QR skener

Pro implementaci QR skeneru jsme použili knihovnu expo-barcode-scanner. Tuto knihovnu lze nainstalovat následujícím příkazem:

```
npx expo install expo-barcode-scanner
```

Pomocí této knihovny chceme docílit toho, aby se uživateli zapnula kamera a při natočení na QR kód se načetla daná objednávka. Pro zapnutí kamery je třeba požádat o přístup ke kameře například tímto způsobem:

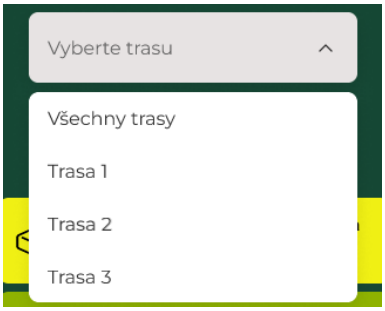
```
const [hasPermission, setHasPermission] = useState(null);

useEffect(() => {
    return navigation.addListener('focus', () => {
        setHasPermission(false);
        (async () => {
            const {status} = await BarCodeScanner
                                .requestPermissionsAsync();
            setHasPermission(status === "granted");
        })();
    });
}, [navigation]);
```

Následná funkcionalita reakce na QR kód je obsažená v této komponentě:

```
<BarCodeScanner
    onBarCodeScanned={handleBarCodeScanned}
/>,
```

kde ”handleBarCodeScanned” je funkce, která se zavolá pokaždé, kdy kamera detekuje QR kód.[11]

4.2.4 Selectlist



Obrázek 4.5: Select list

Pro účely implementace selectlistu výše(použitého v případu užití ”zobrazit objednávky” a v případech užití ”změnit objednávku” a ”změnit objednávky”) jsme se rozhodli použít knihovnu react-native-dropdown-select-list. Tuto knihovnu jsme nainstalovali příkazem:

```
npm install react-native-dropdown-select-list.
```

Její použití bylo velmi jednoduché:

```
<SelectList
    data={selectChoices}
    setSelected={setSelected}
    search={false}
    defaultOption={{key: '1', value: 'Trasa 1'}}
/>
```

Za ”data” jsme dosadili pole objektů s atributy ”key”, ”value”, ”disabled”, které všechny prvky z pole vykreslí pod názvem s hodnotou získanou z ”value”. Za setSelected jsme dosadili funkci, která se provede po každém výběru některé z možností, není-li to stejná možnost, která je právě vybraná. Parametr search je v podstatě vizuálně funkcionální záležitost, kterou chceme mít nastavenou na ”false”. DefaultOption určuje výchozí hodnotu.[12]

4.2.5 Datepicker

Obrázek 4.6: Date picker

Pro implementaci výběru dne zobrazeného výše jsme použili knihovnu @react-native-community/datetimepicker. Její implementace je opět velmi jednoduchá:

```
<DateTimePicker
    value={currentDate}
    onChange={onChange}
/>
```

Za hodnotu ”value” jsme dosadili čas, který se následně zobrazí jako označený a za hodnotu ”onchange” funkci, která se po změně data v rámci této komponenty provede.[13]

4.3 Postup implementace

Nyní, když máme popsanou architekturu a použité knihovny, můžeme začít popisovat samotný postup implementace. Z důvodu přílišného rozsahu nebudeme popisovat přímo napsaný kód, ale pouze důležité funkce a komponenty přímo poskytované frameworkem React Native, knihovnou React nebo jazykem Javascript, na kterých naše aplikace stojí.

Použití těchto funkcí a komponent popíšeme ve 3 logických částech:

- HTTP dotazování - kde popíšeme použití funkce ”fetch”.
- Práce s daty - kde nás budou zajímat ”React hooks”, konkrétně ”useEffect” a ”useState”.
- Zobrazení dat - kde budeme řešit vykreslovací komponenty: ”View”, ”Flatlist”, ”CustomText”.

4.3.1 HTTP dotazování

První důležitá část naší aplikace je HTTP dotazování. K dotazování jsme použili javascriptovou metodu ”fetch” ve všech případech tímto způsobem:

```
export const fetchAPI = (url, body) => {
    return new Promise((resolve, reject) => {
        fetch(url, {
            method: 'POST',
            headers: {
                'Authorization': 'Basic ' + encodedCredentials,
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(body),
            timeout: 30000,
        })
            .then(res => {
                if (!res.ok) {
                    throw new Error('Adresa v siti
                    nebyla nalezena');
                }
                return res.json();
            })
            .then(
                (result) => {
                    resolve(result);
                },
                (error) => {
                    reject(error);
                }
            );
```

```
    });
}
```

V ukázce vidíme funkci s parametry url (url adresa na kterou se dotazujeme) a body (tělo dotazu). Funkce vrací ”Promise”, což je objekt reprezentující hodnotu, která v době vytvoření nemusí být známá a díky němuž můžeme jednoduše počkat, než se provede dotaz (to může trvat několik sekund) a až poté provést zamýšlenou akci, respektive po zavolání funkce resolve nebo reject.

V Promise se nachází samotná ”fetch” funkce sloužící k poslání http požadavku, v našem případě vždy s metodou ”POST”.

V hlavičce je třeba poslat autorizaci, tedy string ve formátu ”username” + ”:” + ”heslo”, který je zakódovaný kódováním base64. Zde stojí za zmínku, že kódování samo o sobě nezajistí zabezpečení, chceme-li zabezpečit komunikaci, musíme použít jiné metody, například zajistit komunikaci protokolem https.

Poté, co přijde odpověď, zkontrolujeme, zda je odpověď v pořádku (status code je ve formátu 2xx). Pokud je v pořádku, funkce vrátí odpověď a zavolá funkci ”resolve”, čímž prohlásí ”Promise” za ”fulfilled” a provede se případná .then() funkce. V opačném případě funkce vyhodí error, prohlásí ”Promise” za ”rejected” a provede se případná .catch() funkce.

4.3.2 Řízení vykreslování dat

Další důležitá část, kterou stojí za to popsat, je řízení vykreslování dat. Po každém http dotazu a při některých uživatelských interakcích dochází k ukládání a vykreslování dat v reálném čase. Pro tyto účely jsme v rámci naší aplikace použili funkce useState a useEffect.

- useState - tato funkce složená ze 2 hodnot reprezentující současný stav a funkce na změnu stavu slouží k uchování stavu proměnné. Má tu dobrou vlastnost, že při každé aktualizaci stavu dojde k překreslení komponent, kde se vyskytuje hodnota tohoto stavu.
- useEffect - tato funkce má 2 parametry, a sice funkci a závislost, dle které se daná funkce zavolá, typicky jméno proměnné, která pokud se změní, zavolá se funkce.

Tyto ”React hooks” jsme použili například tímto způsobem:

```
const [isLoading, setIsLoading] = useState(false);
const [response, setResponse] = useState([]);
const [error, setError] = useState(null);
const [date, setDate] = useState(new Date());
useEffect(() => {
    setIsLoading(true);
    fetchOrders(date)
        .then(
            (result) => {
                setIsLoading(false);
                setError(null);
                setResponse(result);
            },
            (error) => {
                setIsLoading(false);
                setError(error);
            }
        );
}, [date])
```

Jako ukázku jsme vybrali funkci, která zajišťuje, aby si při změně data objednávek (děje se i při iniciálním načtení obrazovky) aplikace dotáhla objednávky pomocí http requestu (fetchOrders) a umožnila její zobrazení (setResponse). V ukázce se funkce v prvním argumentu useEffectu zavolá kdykoliv dojde k zavolání funkce ”setDate”. Funkce tedy nastaví ”isLoading” na ”true”, díky čemuž může aplikace zobrazit uživateli indikátor, že dochází k načítání.

Dále zavolá http request ”fetchOrders” a po dokončení se nastaví ”isLoading” na false, aby uživatel znovu věděl, že aplikace již nenačítá.

Obdobný princip používání useEffect a useState jsme použili na několika místech v naší aplikaci.

4.3.3 Vykreslovací prvky

Poslední část, která stojí v postupu implementace za pozornost, jsou vykreslovací prvky, neboli komponenty, které jsou reálně viditelné v aplikaci. Tyto prvky, zejména první 2 z nich, jsou v aplikaci využívány velmi hojně. Jde o následující prvky:

- View - určitý ekvivalent ”div” tagu v html - jde o komponentu, která slouží k zaobalení a pozicování dalších komponent v rámci obrazovky.
- CustomText - komponenta, kterou jsme si vytvořili jako náhradu za ”Text”, abychom si mohli uzpůsobit veškerý text dle našich potřeb. Komponenta slouží k vykreslení textu.
- Flatlist - Komponenta sloužící k vykreslení pole prvků (například seznamu objednávek).

Jednoduchý příklad může vypadat takto:

```
<View style={styles.list}>
    <FlatList
        data={data}
        renderItem={({item}) => (
            <View>
                <CustomText>item.name</CustomText>
            </View>
        )}
    />
</View>
const styles = StyleSheet.create({
    list: {
        marginTop: 30,
    },
})
```

Příklad výše (který není použit v aplikaci, ale podobá se vykreslení seznamu objednávek) ukazuje jednoduché použití všech výše uvedených komponent. Příklad obsahuje ”View” komponentu, která je pozicována pomocí stylů definovaných v ”styles.list” tak, aby byla shora odsazena o 30 dp (density-independent pixels).

Dále příklad obsahuje komponentu ”FlatList”. Komponenta obsahuje následující props:

- data - zde se dosadí pole objektů, které se mají vykreslit.
- renderItem - zde se dosadí funkce, která určí, jakým způsobem se každý objekt z pole ”data” vykreslí.

Funkce dosazená za ”renderItem” způsobí, že pro každý prvek z pole ”data” se vykreslí text s názvem odpovídající atributu ”name” daného prvku.

Nyní máme popsanou celou implementační část. Popsali jsme si architekturu, použité knihovny a postup implementace, kde jsme si ukázali stavební kameny, na kterých naše aplikace stojí. Celou implementaci je možné si prohlédnout v elektronické příloze pod názvem CLApp.zip.

Kapitola 5
Uživatelské testování

V následující části popíšeme, jakým způsobem jsme naši aplikaci testovali. Základní testy funkčnosti (že aplikace nespadne; že funguje, jak má) jsme prováděli v reálném čase při vývoji díky Expo CLI. Poté testování probíhalo ve 2 fázích, kde v první fázi se testoval vzhled a uživatelská přívětivost a v další fázi se testovala veškerá funkčnost. Z časových důvodů se nestihla poslední zamýšlená fáze, a sice testování funkčnosti v produkčním prostředí. Ta bude realizována později v termínu, který jsme si se zadavatelem domluvili.

5.1 První fáze

V první fázi testování jsme se sešli s UI designerem s dlouholetými zkušenostmi, který nám dal zpětnou vazbu ke vzhledu a uživatelské přívětivosti aplikace.

Schůze jsem se účastnil já a designer a ze schůze vzešlo 5 podnětů, které pro úspěšnost projektu byly méně stěžejní. Všechny podněty byly zapracovány do aplikace. Zde je kompletní výčet podnětů:

- Nadpisy napříč aplikací by měly být více odsazené.
- V přehledu objednávky - tlačítka by měla být na stejném místě.
- V přehledu objednávky - skupiny produktů (”koloniál”, ”chlazené”, ”pečivo”) by měly mít vedle sebe ikonku.
- Změna trasy - pro lepší uživatelskou přívětivost třeba změnit design, aby se na levé straně obrazovky ukazovalo odkud se přeřazuje a na pravé straně kam se přeřazuje objednávka/objednávky.
- ”Stack navigator” - horní levé menu by mělo ve vnořené obrazovce ukazovat, kam se chce uživatel vrátit (případně text ”zpět”) a ne nadpis obrazovky, kde se uživatel nachází.

5.2 Druhá fáze

Ve druhé fázi testování se již testovalo, zda je aplikace funkční a zda obsahuje všechna data, která dispečer potřebuje. Testování jsem se účastnil já, 3 dispečeři a ředitel velkoobchodu a testovali jsme 3 průchody aplikací, které zároveň

prochází všemi stavy, které v pozitivním případě mohou v aplikaci nastat (pozitivní případy znamenají takové případy, kdy uživatel nemá problém s připojením k internetu a se spojením s ABRA Gen). Tato fáze probíhala v testovacím prostředí ABRy.

Tyto průchody jsou vypsány zde:

- Průchod z přehledu objednávek do detailu konkrétní objednávky.
- Průchod z přehledu objednávek do provedení přeřazení trasy.
- Průchod z přehledu objednávek do provedení přeřazení více tras naráz.

Z testování vzešly 3 náměty, které pro úspěšnost projektu byly důležité a 1 námět, který byl méně důležitý. Všechny náměty jsme zapracovali do aplikace a jejich výčet v pořadí 3 důležité a 1 méně důležitý je zde:

- Přehled objednávek neukazuje správný název provozovny a adresu.
- Přehled objednávky neukazuje správně, zda objednávka obsahuje ”chlazené”.
- Přehled objednávky nevypisuje důležité údaje, konkrétně přesnou adresu provozovny a její jméno, datum doručení, váhu objednávky.
- Celou obrazovku ”detail objednávky” není třeba v aplikaci mít (stačí informace v přehledu objednávky).

5.3 Shrnutí

Během testování vzešlo 9 námětů. které byly všechny zapracovány. Některé z nich mohly být díky Expo CLI zapracovány hned poté, co byly vzneseny, což firma velmi ocenila.

Během testování se výrazně změnil vzhled obrazovek Orderoverview (zobrazit objednávku), ChangeTrace (přeřadit objednávku/více objednávek) a úplně zanikla obrazovka OrderDetail (zobrazit detail objednávky), viz. struktura aplikace a use case diagram. Výsledná aplikace se tak poměrně výrazně liší od původního schváleného prototypu, nicméně účel prototypu byl nastínit, jak by aplikace mohla zhruba vypadat, ne jak přesně bude vypadat, což prototyp splnil.

Pro plné otestování jsme se zadavatelem naplánovali schůzku, na které by měla proběhnout třetí fáze testování. V rámci této fáze bude aplikace otestována v produkčním prostředí.

Kapitola 6
Návrh dalšího vývoje

Nyní, když máme naši aplikaci otestovanou, nastal vhodný čas pro návrh dalšího možného vývoje. Vzhledem k tomu, že původní konzultovaný návrh řešení procesů zahrnoval 3 aplikace a až poté došlo k domluvě, že se bude implementovat jen jedna aplikace, otvírá se nám více možností směřování budoucího vývoje.

6.1 Dashboard pro dispečery

První možností je implementace dashboardu pro dispečery. Tento dashboard se konzultoval v prvních schůzkách s firmou a vzešel z toho wireframe, který lze najít na tomto odkaze.[8]

Návrh je velmi jednoduchý, musel by se prokonzultovat s firmou (přičemž by se dohodlo, která data by byla pro firmu ještě třeba vizualizovat) a šlo by o zobrazení přehledu pohybu objednávek na velké plátno, což by pomohlo dispečerům získat ještě lepší přehled pohybu objednávek.

Toto řešení by mohlo být realizováno jako webová aplikace v jazyce React, která by podobně jako naše řešení pro dispečery komunikovala se systémem ABRA Gen pomocí REST rozhraní a zobrazovala získaná data.

6.2 Aplikace pro kurýry

Další možností je vytvoření verze aplikace pro kurýry, a to buď v rámci aplikace původně určené dispečerům, nebo jako separátní aplikaci. Kurýrovi by mohla pomoci zkvalitnit práci aplikace obsahující přehled, který by byl podobný, jako má dispečer, akorát s přesnějšími informacemi o adresách doručení a s dodatečnými informacemi (například o jaké objednávky se jedná, jestli klient platí hotově atd.).

6.3 Rozvoj stávající aplikace

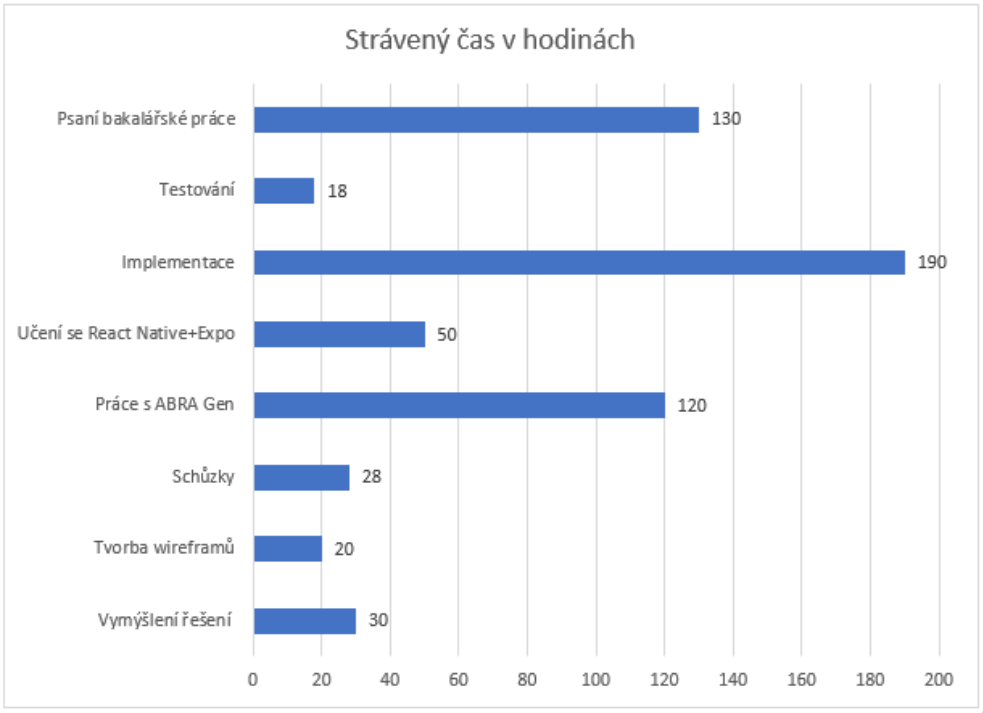
Další zřejmá možnost pro další vývoj je rozvoj aplikace pro dispečery. V rámci finální schůzky s dispečery padlo několik nápadů na funkcionality,

například přidat možnost ”odložit” objednávku, aby byla viditelná kterýkoliv den. Kromě toho by šlo i nabízet přesuny tras v reálném čase podle celkové hmotnosti objednávek na trase. Pro všechny tyto návrhy na rozvoj, včetně aplikace pro kurýry a dashboardu, budou stěžejní konzultace s firmou.

Kapitola 7

Časové rozložení práce

V této kapitole se podíváme na časové rozložení práce, tedy kolik času nám zabraly jednotlivé aktivity, které jsme konali v rámci bakalářské práce.



Obrázek 7.1: Strávený čas na bakalářské práci

Na obrázku 7.1 je vidět graf znázorňující časovou náročnost jednotlivých aktivit v rámci bakalářské práce. Uvedená čísla jsou v hodinách a jsou pouze orientační, nicméně reálné hodnoty se od nich příliš neliší.

Aktivita ”vymýšlení řešení” zahrnuje pouze přemýšlení nad řešením, tedy přemýšlení, jak by mohly fungovat budoucí procesy s použitím aplikace a rozplánování aplikace.

Aktivita ”práce s ABRA Gen” zahrnuje nastudování tohoto systému, zprovoznění Web API pro testovací účely a implementaci v aplikaci.

Aktivita ”učení se React Native+Expo” zahrnovala učení se technologie

React Native a učení se technologie Expo CLI. Aktivita zabrala poměrně velké množství času zejména z toho důvodu, že nám dlouho trval proces zjišťování, jak funguje aktualizace aplikace vzdáleně bez nutnosti opětovné distribuce a instalace.

Největší množství času nám trvala implementace, v rámci které jsme také testovali funkčnost v reálném čase pomocí Expo CLI.

Graf na obrázku 7.1 může dát čtenáři hrubou představu o tom, kolik času mohou jednotlivé aktivity v rámci podobného projektu při neznalosti technologií zabrat. Chtěl bych tímto také ukázat, že integrace s firemním systémem (například ABRA Gen) může při neznalosti tohoto systému trvat dlouho (v našem případě 120 hodin). V takové situaci může být výhodnější si zaplatit externí firmu (nebo přímo poskytovatele ABRA), která tuto integraci provede.

Celkový čas strávený na projektu bez započítání psaní bakalářské práce, byl 456 hodin, což může dát čtenáři přibližnou představu o náročnosti projektu podobného rozsahu při neznalosti technologií.

Kapitola 8

Závěr

Cílem naší práce bylo zlepšit ve firmě Country life procesy dispečerů a kurýrů pracujících ve velkoskladě a to pomocí nástrojů softwarového inženýrství. V průběhu analytické části projektu jsme však zjistili, že realizace 3 separátních aplikací, tedy 2 aplikací pro dispečera a 1 pro kurýra, by byla vzhledem ke složitosti technologií ABRA Gen a React Native velmi časově náročná a že má firma zájem zejména o mobilní aplikaci pro dispečery. Proto jsme se po konzultacích se zadavatelem rozhodli realizovat jen tuto jednu aplikaci pro dispečery. Pro případnou budoucí realizaci zbývajících 2 aplikací máme základní podklady ve formě prototypů.

Nejprve se nám podařilo prozkoumat možnosti Web API poskytované technologií ABRA Gen a poté navrhnout prototyp, který firma schválila. Následně se nám podařilo realizovat aplikaci implementací v jazyce React Native pomocí technologie Expo CLI, která je dnes velmi populární a poskytuje mimo jiné testování funkčnosti v reálném čase, což velmi ocenila firma při testování, jelikož některé náměty jsme do aplikace zakomponovali ihned po sdělení těchto námětů.

Po implementační části jsme aplikaci otestovali jak vizuálně za pomoci UI designera s dlouholetou zkušeností, tak za pomoci koncových uživatelů, ředitele IT a ředitele velkoobchodu, který zároveň nejvíce ovlivnil, zda se projekt uskuteční či nikoliv. Testovací část byla provedena v testovacím prostředí ABRy, ale ještě nebyla provedena v produkčním prostředí. Z tohoto důvodu nemůžeme prohlásit projekt za úspěšně uzavřený. K tomuto stavu má ale velmi dobře nakročeno, jelikož je testování v produkčním prostředí naplánováno na konkrétní termín.

Všechny cíle práce tedy byly naplněny, s výjimkou dokončení testování a nasazení, což proběhne až po odevzdání této práce.

Po otestování aplikace v produkčním prostředí plánuje firma nasadit tuto aplikaci do provozu ihned, jakmile to bude možné. Proto se dá očekávat, že v práci na mobilní aplikaci (případně na mobilní aplikaci pro kurýry nebo webové aplikaci pro dispečery) budu pokračovat. Práce mi tedy přinesla nejen rozhled v nových technologií a zkušenost, ale i potenciálního partnera, se kterým možná budu moci spolupracovat na dalších projektech.

Literatura

[1] Official Country life financial statement for 2021 - https://or.justice.cz/ias/ui/vypis-sl-detail?dokument=73218442&subjektId=116361&spis=100851 (visited: 19.12.2022)

[2] Straits research about market trend of WMS technology - https://straitsresearch.com/report/warehouse-management-system-market (visited: 8.5.2023)

[3] Country life website - https://www.countrylife.cz/ (visited: 19.12.2022)

[4] elvaston bought abra news - https://www.lupa.cz/aktuality/nemci-za-stovky-milionu-ovladli-ceskou-spolecnost-abra-software (visited: 8.5.2023)

[5] ABRA website - https://www.abra.eu (visited: 8.5.2023)

[6] ABRA Gen technical documentation - https://help.abra.eu/cs/22.2/G3/Content/Part50_TechDoc/tech_doc.htm (visited: 16.04.2023)

[7] ABRA Gen API documentation - https://apisandbox.abra.eu/ (visited: 16.04.2023)

[8] Final wireframes in Figma - https://www.figma.com/file/chHlWZ1y1rzZDlmMZZe0XA/countrylife?node id=51495

[9] With Redux or without Redux picture - `https://blogs.kowthasaketh.com/create-own-react-redux-using-context-api-in-react` (visited: 21.04.2023)

[10] React Native CLI vs Expo CLI website- https://levelup.gitconnected.com/react-native-cli-vs-expo-cli-which-one-do-i-choose-bdf02ea457bf (visited: 26.04.2023)

[11] Expo documentation - https://docs.expo.dev (28.04.2023)

[12] Dropdown selectlist library - https://www.npmjs.com/package/react-native-dropdown-select-list (visited: 1.3.2023)

[13] Date picker library - https://www.npmjs.com/package/@react-native-community/datetimepicker (visited: 3.3.2023)

Příloha A

Struktura přiložených souborů

```
Obrazky ..... Obrázky uvedené v práci.
CLApp.zip ..... Zdrojový kód aplikace.
thesisSource.zip..... Zdrojové kódy textu ve formátu LaTeX.
thesis.pdf..... Text práce ve formátu PDF.
schuzky.pdf..... Záznam schůzí ve formátu PDF.
```

Příloha B

Seznam použitých zkratek

Zkratka	Význam
WMS	Warehouse Management System
API	Application Programming Interface
ERP	Enterprise resource planning
REST	Representational State Transfer
UI	User Interface
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
CLI	Command-line Interface
HTML	HyperText Markup Language
SQL	Structured Query Language
NPM	Node Package Manager
NPX	Node Package Execute
CRUD	Create Read Update Delete
URL	Uniform Resource Locator
MB	Megabyte
APK	Android Package Kit
DP	Density-independent pixels

Příloha C

Slovník

demodata data, pomocí kterých lze testovat funkčnost. 8

expedient pracovník vyskládávající objednávky ze skladu na expediční plochu. 3, 5

koloniál skupina produktů zahrnující všechny produkty kromě skupiny ovozel (ovoce nebo zelenina), chlazené (produkty, které jsou skladovány v chlazeném prostředí) a pečivo. 5, 23, 28, 39

ovozel skupina produktů zahrnující pouze ovoce a zeleninu. 5, 23, 28

svozové objednávky objednávky, kde na straně objednávajícího je firma. 6

trasy očíslované vykládací plochy, odkud kurýři nakládají objednávky do svých vozidel. 5, 20, 23