FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Frontend administrace e-shopu - objednávky
Student:	Martin Salaj
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem této práce je realizace nového frontendu administrace e-shopu se zaměřením na vyřízení a zpracování objednávky zákazníka. Tato práce volně navazuje na bakalářskou práci Bc. Martina Dvořáka.
Tento frontend by měl komunikovat s nově vznikající odpovídající částí backendu, který je souběžně vytvářen a z části již existuje.

Postupujte v těchto krocích:
1. Analyzujte navržené řešení administrace objednávek od Bc. Martina Dvořáka a současný stav backendu se zaměřením na zpracování objednávek.
2. Na základě analýzy a současného stavu frontendu navrhněte vhodný postup realizace části věnované zpracování objednávek a případně i změn dle zjištěných nedostatků.
3. Konzultujte návrh alespoň se zadavatelem.
4. Implementujte frontend zpracování objednávek, pokuste se jej propojit s nově vznikající odpovídající částí backendu. Velký důraz věnujte použitelnosti.
5. Při realizaci nezapomeňte na důkladné testování, například lze využít aktuální uživatele dané administrace.
6. Zhodnoťte výsledné řešení, navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2023 v Praze.

Bakalářská práce

FRONTEND ADMINISTRACE E-SHOPU - OBJEDNÁVKY

Martin Salaj

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
11. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych vyjádřil svou vděčnost své rodině, přátelům a inženýru Jiřímu Hunkovi za jejich podporu během psaní mé bakalářské práce. Vaše povzbuzení a vedení pro mě byly velkou motivací tento projekt dokončit. Děkuji vám, že jste ve mě věřili a přiměli mě dělat své maximum.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 .....

Abstrakt

Cílem této práce je vytvořit moderní a efektivní webovou aplikaci pro správu objednávek v elektronických obchodech pro administrátory. Zaměřuje se na analýzu návrhu uživatelského rozhraní pro úpravu objednávek, implementaci frontendu a jeho integraci s nově vyvíjeným backendem. Cílem je dodání uceleného a efektivního systému pro správu objednávek v elektronických obchodech, který umožní administrátorům e-shopů zaměřit se na růst svých podniků.

Implementace návrhu frontendu bude založena na moderních technologiích a frameworcích, aby systém byl robustnější a efektivnější. Práce bude také zahrnovat refaktorizaci stávajícího systému na tyto nové technologie. Integrace frontendu s backendem zajistí, že administrátoři e-shopů budou mít k dispozici moderní, efektivní a spolehlivý nástroj pro správu svých objednávek.

Klíčová slova administrace e-shopu, webová aplikace, úprava objednávek, frontend, e-commerce, Vue.js, Vuetify, TypeScript

Abstract

The goal of this thesis is to create a modern and efficient web application for e-shop administrators to manage their orders. The focus will be on analyzing the design for the frontend of the page for editing orders, implementing the frontend based on this design, and integrating it with a newly developed backend. The aim is to deliver a comprehensive and efficient system for managing orders in e-shops, ensuring that e-shop administrators can focus on growing their businesses.

The implementation of the frontend design will be based on modern technologies and frameworks to make the system more robust and efficient. The thesis will also involve refactoring the current system to said new technologies. The integration of the frontend with the backend will ensure that e-shop administrators have access to a modern, efficient, and reliable tool for managing their orders.

Keywords e-shopu administration, web application, editting orders, frontend, e-commerce, Vue.js, Vuetify, TypeScript

Seznam zkratek

MVP	Minimum Viable Product
NTH	Nice To Have
FURPS	Functionality, Usability, Reliability, Performance and Supportability

Úvod

Elektronický obchod (e-commerce) se stal nezbytnou součástí moderního byznysu. S nárůstem popularity online nakupování je pro administrátory e-shopů nezbytné mít k dispozici efektivní nástroje pro účinné řízení svých obchodů. Jedním z klíčových prvků správy e-shopů je řízení objednávek. Tento proces zahrnuje různé úkoly, jako je aktualizace stavů objednávek, sledování zásilek a zpracování plateb. Nicméně současné systémy pro řízení objednávek v e-shopech jsou často zastaralé a postrádají potřebné funkce pro zefektivnění procesu.

Důvody pro realizaci této práce jsou dva. Za prvé, současný systém pro řízení objednávek v e-shopech je zastaralý a neefektivní. Za druhé, s vývojem nového backendu je nutné vyvinout odpovídající frontend, který bude moderní, uživatelsky přívětivý a nabídne vylepšenou funkčnost. S vývojem nové webové aplikace budou mít administrátoři e-shopů k dispozici efektivnější nástroj pro řízení svých objednávek, což jim umožní soustředit se na růst svého byznysu.

Dvořák a Babák započali projekt webové aplikace pro e-shopy, který měl sloužit k efektivnějšímu řízení obchodů. Bohužel, kvůli nedostatku času se jim nepodařilo implementovat systém pro správu objednávek, a proto se moje bakalářská práce zaměřuje právě na tuto oblast. [1][2]

Pro implementaci funkcionality objednávek jsem využil designu, který vytvořil Dvořák. Tento design byl pečlivě navržen s cílem zlepšit uživatelskou zkušenost pro administrátory e-shopu. Obsahuje mnoho funkcí, které jsou důležité pro efektivní správu objednávek. S využitím tohoto designu se budu snažit vytvořit uživatelsky přívětivé rozhraní, které umožní rychlou a efektivní správu objednávek v rámci e-shopu. [1]

Kapitola 1
Cíle práce

Cílem mé práce je třífázový proces. Prvním cílem je analyzovat design frontendu stránky pro úpravu objednávek, který vytvořil Dvořák ve své práci. Tento krok zahrnuje detailní studium návrhu, včetně jeho rozvržení, struktury a použitých prvků. Cílem je získat hlubší porozumění funkcím, které by měl frontend nabízet, a zjistit, jak by se mohl nejlépe integrovat s novým backendem. [1]

Druhým cílem mé práce je implementovat frontendovou část založenou na designu, který Dvořák vytvořil, a který byl analyzován v první fázi. Implementace musí být pečlivě navržena tak, aby zahrnovala všechny funkce, které jsou vyžadovány pro úpravu objednávek. Součástí této fáze je i testování implementace, aby se zajistilo, že je plně funkční a splňuje požadavky. [1]

Posledním cílem mé práce je propojení frontendové části s nově vyvíjeným backendem. Tato fáze zahrnuje navrhování a implementaci API pro komunikaci mezi frontendem a backendem. Cílem je zajistit, aby tyto dvě části byly plně synchronizované a aby bylo možné úspěšně upravovat a zobrazovat objednávky v reálném čase.

Kromě výše zmíněných cílů je důležitým aspektem této práce přepsání stávajícího systému do nových technologií. To bude zahrnovat aktualizaci kódu na moderní programovací jazyky a frameworky, což systém udělá robustnější a efektivnější. Nicméně, tento proces může být náročný a časově náročný a existuje riziko, že může ovlivnit implementaci některých cílů projektu. Proto bude důležité pečlivé plánování a řízení procesu přepsání, aby bylo zajištěno, že implementace návrhu frontendu a integrace s novým backendem nebudou zásadně ovlivněny.

Díky přepsání bude implementace frontendu založena na moderních technologiích a frameworcích. To zajistí, že systém bude škálovatelný, robustní a schopný zvládat složité požadavky na správu objednávek v e-shopech. Implementace bude navržena tak, aby umožňovala budoucí aktualizace a vylepšení, což zajistí, že systém lze přizpůsobit měnícím se požadavkům a technologickým inovacím.

V závěru lze říci, že hlavním cílem této práce je vytvořit moderní a funkční část webové aplikaci pro úpravu objednávek v e-shopech. Tato práce by mohla přinést významné zlepšení v oblasti správy objednávek v e-shopech a pomoci administrátorům e-shopů zefektivnit své procesy.

Kapitola 2
Analýza

V této kapitole nejprve vyberu vhodný model softwarového vývoje a následně provedu analýzu požadavků zpracovaných Martinem Dvořákem. V rámci analýzy provedu změny dle požadavků vedoucího práce, který se vyskytuje v pozici stakeholdera. Nakonec musím zanalyzovat momentální stav webové aplikace a využitých technologií a vybrat vhodný přístup k vývoji komponenty úpravy objednávky.

2.1 Modely softwarového procesu

V životním cyklu softwaru, který by měla bakalářská či diplomová práce dokumentovat, se postupuje v krocích. Nejprve se provede analýza, návrh, implementace, testování a nakonec nasazení softwaru. [3]

Je třeba se zamyslet jestli by bylo produktivní dělat tyto kroky serializovaně, jelikož je možné rozvrhnout práci, tak aby se mohla dělat analýza jedné části této komponenty zároveň s implementací jiné komponenty.

Také je nutné zmínit, že analýza původní webové aplikace, kterou by měla tato nahradit je již provedena v bakalářské práci Dvořáka, stejně tak jako návrh, který je z velké části dodělán a bude pouze upravován dle potřeby. [1]

Nicméně zastávám názor, že v rámci práce, by se měly vyzkoušet všechny kroky. I když analýza v rámci této práce bude spíše analýza již zhotoveného designu a design jako takový se spíše bude týkat úprav částí, se kterými nebudu souhlasit, nebo změn designu dle potřeby. Podíval jsem se tedy na modely softwarového procesu abych vybral ten nejvhodnější.

Teď přiblížím některé metodiky softwarového vývoje a zhodnotím jejich výhody a nevýhody a nakonec vyberu tu, kterou použiji pro tento projekt. Nutno podotknout, že zde nevypíši všechny metodiky, co existují, ale pouze se zaměřím na pár konkrétních, mezi nimiž jsem se rozhodoval.

2.1.1 Vodopád

Ve vodopádu jsou jednotlivé fáze - analýza, design, implementace, testování, nasazení, oddělené a pevně na sebe navazují, to znamená, že nemůžeme začít další fázi dříve než dokončíme tu předešlou. [4]

Výhody:

- jasně stanovený plán
- snadná koordinace práce
- predikovatelnost - deadline, scope

Nevýhody:

- není možně dělat změny návrhu, nebo je to velmi obtížné
- nutnost chápat, požadavky už na začátku
- zpětná vazba od zákazníka až na konci

2.1.2 Iterativní

Vzniká několik verzí systému, které vznikají sekvenčně. Jedná se o sérii vodopádů s menším časovým rozsahem. Každý z vodopádů se zabývá nějakou částí systému. Umožňuje rychlejší reakci na změny, to však může zvětšit rozsah projektu a znepřesnit odhad dodání projektu. [5]

Výhody oproti vodopádu:

- Zákazník má přístup k verzím a může na ně reagovat

Nevýhody:

- Nutno chápat požadavky již na začátku

2.1.3 Agilní

Vychází z iterativního, ale má kratší iterace a jednotlivé verze nemusí být nutně produkční. Ztrácíme tím predikovatelnost dodání, je obtížnější plánovat pro delší časové úseky. Z tohoto důvodu tento model klade vysoké nároky na celý tým. [6]

Výhody oproti iterativnímu:

- verze máme brzy, zákazník může reagovat rychleji

Nevýhody:

- nutné zapojení celého týmu
- silný business vlastník
- cenově náročnější

2.1.3.1 SCRUM

SCRUM je metodika agilního vývoje. Hlavním bodem je, že se skládá ze sprintů, které trvají zhruba 2 až 3 týdny. Každý sprint má svůj sprint goal, který se team snaží udělat. Cílem sprintu je přinést do systému inkrement. Na konci sprintu je review se stakeholderama a project ownerem, kde se zhodnotí práce, dále zde je refinement dalšího sprintu, kde se určí sprint goal na další sprint. Tento postup se opaku dokud se software nezhotoví. SCRUM jako takový má další ceremonie, které tu nebudu zmiňovat jelikož je nehodlám využít, protože nemají smysl, když na práci dělám v podstatě sám. [7]

2.1.4 Analýza návrhového systému

Design systém je kolekce návrhových prvků, které umožňují konzistentní vzhled a chování v aplikacích. Jeho hlavním cílem je ušetřit čas a usnadnit proces návrhu a vývoje. Obsahuje již připravené prvky a komponenty, které lze snadno a opakovaně použít v různých projektech. Design systém také pomáhá zajistit, že aplikace budou mít jednotný a intuitivní vzhled a chování. [8]

2.1.4.1 Material-design

V rámci tohoto projektu byl vybrán Material-design, konkrétně knihovna Vuetify, o které se zmíním samostatně. Material-design jako takový je návrhový systém postavený a podporovaný designéry a vývojáři Google. Díky tomu má rozsáhlou dokumentaci a jeho standardy dodržuje mnoho webových stránek a aplikací. Obsahuje komponenty s moderním vzhledem, které umožní poskládat intuitivní a dobře-vypadající weby a aplikace. [9] [10]

2.1.5 Rozhodnutí

Jelikož na projektu pracuji zejména sám a využívám vedoucího jako stakeholdera a bc. Martina Dvořáka jako konzultanta, tak jsem se rozhodl zvolit agilní přístup. Nechci se úplně zavázat ke konkrétní metodice jako je SCRUM, jelikož s touto metodou přichází spousta ceremonií, které by byly spíše kontraproduktivní (retrospektiva, daily meet, ...). Nicméně stejně bych se chtěl metodou SCRUM inspirovat a dodržovat dvoutýdenní sprinty, kde na konci udělám review s vedoucím a zároveň se domluvím, co bych chtěl udělat další sprint. Zvolil jsem tento postup zejména, kvůli rychlé reakci na změny od stakeholdera. [7]

2.2 Analýza požadavků

Při analýze požadavků se velmi často využívá metoda FURPS: Functionality, Usability, Reliability, Performance and Supportability. Což znamená: funkčnost, použitelnost, spolehlivost, výkon a rozšiřitelnost. Funkčnost je zařízena funkčními požadavky a zbytek požadavky nefunkčními. [11]

Tato metoda vznikla především, aby se na nějaký požadavek nezapomnělo. Zaručí se tím tak maximální spokojenost zákazníka.

Další metoda může být například rozdělení požadavků na MVP (Minimum viable product) a NTH (Nice to have). Rozdělíme tím tak požadavky na minimum (MVP), které je potřeba udělat, aby byl produkt zprovoznitelný, když se to zvládne dokončit (NTH). [12]

V rámci NTH požadavků pak nadále můžeme nastavit prioritu, která nám určí pořadí, ve kterém požadavky dodávat. Čím menší číslo priority tím důležitější požadavek je (1-5). [12]

Díky tomu, že jsem si vybral agilní metodiku, tak je možné, že se v průběhu realizace naskytne featura, kterou zadavatel bude chtít mít ve výsledném produktu, pak se běžně postupuje tak, že se zjistí jestli na danou implementaci je budget (v mém případě čas) a následně se prodiskutuje, jestli je to pro zadavatele MVP a nebo NTH s danou prioritou.

2.2.1 Analýza na základě současného návrhu

Nadále se budu věnovat analýze návrhu požadavku "Stránka pro upravení objednávky"vzniklého z práce Dvořáka, ve které již do podrobna zvážil požadavky zadavatele. [1]

2.2.1.1 Analýza stránky pro upravení objednávky

Stránka jako celek bude obsahovat 5 karet. Ačkoli každá z karet má vlastní funkčnost, tak mají schopnost ovlivňovat ostatní. Takže bude třeba, aby mezi nimi byly dobře synchronizovaná data. Zároveň je potřeba, aby všechny karty byly přemístitelné a daly se zvětšovat a zmenšovat dle potřeby. Nejdříve zanalyzuji požadavky stránky jako celku a pak se podívám na každou z karet jednotlivě.

Funkční požadavky

Minimal Viable Product (MVP)

- karta podrobnosti objednávky
- karta historii objednávky
- karta údaje pro doručení objednávky
- karta produkty v objednávce
- karta zásilky objednávky
- možnost přepínání jazyka mezi ENG a CZ
- možnost přesouvání karet
- možnost zmenšování a zvětšování karet

Nice To Have (NTH)

- možnost dostat se na stránku pro upravení dobropisů (prio 1)
- uložení layoutu karet pro další otevření stránky (prio 1)

Nefunkční požadavky

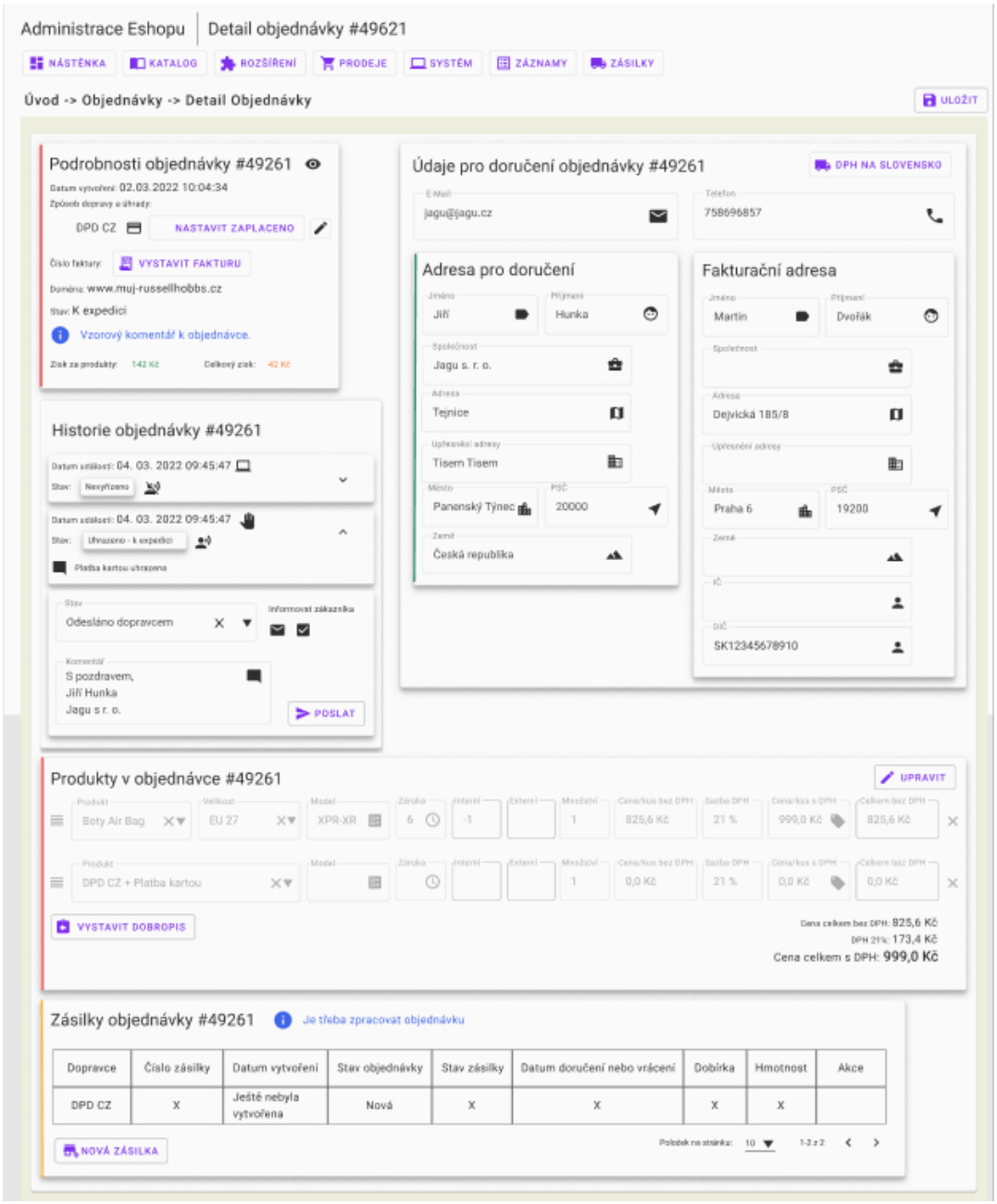
Minimal Viable Product (MVP)

- napojení na backend
- synchronizace dat mezi kartami

Nice To Have (NTH)

- správné fungování pro menší displaye: tablet, mobil (prio 2)

Obrázek 2.1 Návrh stránky Upravit objednávku



2.2.1.2 Analýza karty Podrobnosti objednávky

Karta "Podrobnosti objednávky"slouží zprvu k zobrazení jednotlivých detailů objednávky. Dále by měla umožnit, lehce nastavit způsob doručení a platby. A nakonec umožnit rychle vystavit fakturu.

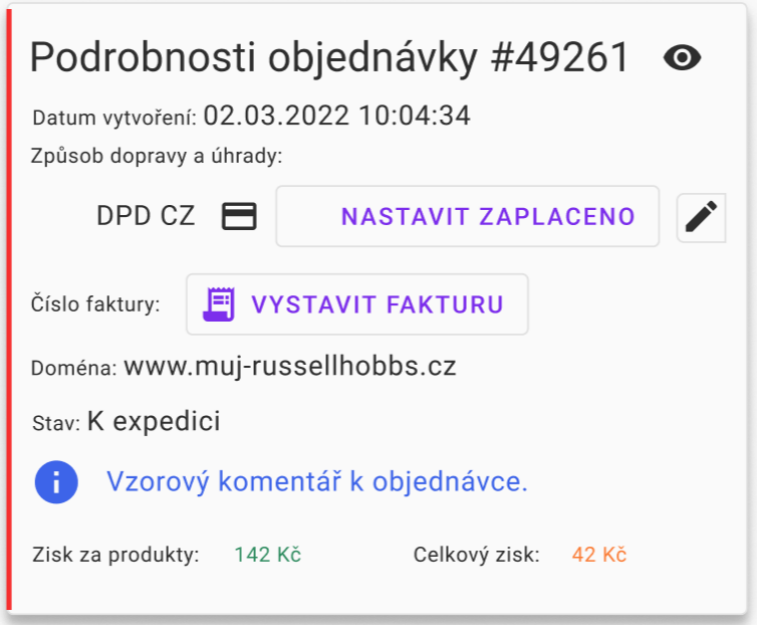
Minimal Viable Product (MVP)

- zobrazení dat: datum vytvoření, způsob dopravy a úhrady, číslo faktury, doména, stav objednávky, komentář, zisk za produkty a celkový zisk

- napojení a synchronizace s backendem

Nice To Have (NTH)
- funkční tlačítko pro nastavení způsobu dopravy a platby (prio 1)
- funkční tlačítko pro vystavení faktury (prio 1)
  - Pokud faktura je vystavená, tak jí zobrazí a dá možnost vytisknout
  - Pokud není vystavená, tak ji stáhne
- linka na levé hraně karty, která zobrazuje stav zaplacení: červená - není zaplaceno, oranžová - není skladem, modrá - je komentář, jinak zelená (prio 2)

Obrázek 2.2 Návrh karty Podrobnosti objednávky



2.2.1.3 Analýza karty Historie objednávky

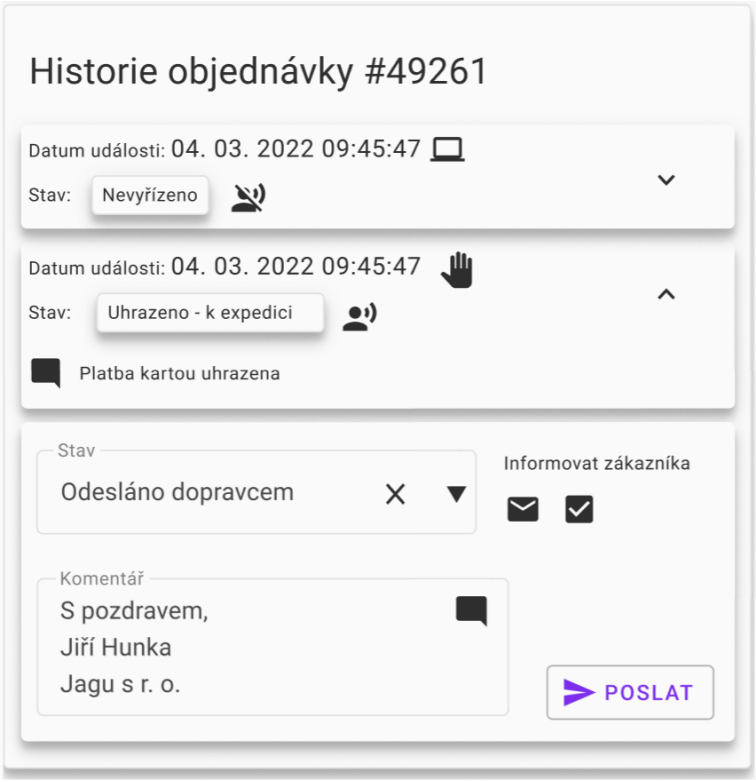
Karta "Historie Objednávky"slouží zejména k historie stavů, kterými objednávka prošla a k možnosti manuálně objednávku přepnout do jiného stavu.

Minimal Viable Product (MVP)
- Seznam karet, které budou zobrazovat historii. Každá karta by měla mít:
  - Datum kdy se do stavu dostala
  - Informaci zda změna stavu byla provedena systémem, nebo správcem. (pomocí ikony počítače a ruky)
  - Stav ve kterém objednávka byla
  - Zda byl zákazník informován (pomocí ikony ne/přeškrtnutého panáka)
  - Vysouvací komentář
- Formulář, který slouží k manuální změně stavu:
  - Výběr stavu do kterého je objednávku možné přesunout (Výběr stavu automaticky vyplní, zbytek formuláře)

- Checkbox zda má být zákazník informován
- Pole kam lze napsat komentář
- Tlačítko, které informace ve formuláři pošle do databáze

- Napojení a synchronizace s backendem

Obrázek 2.3 Návrh karty Historie objednávky



2.2.1.4 Analýza karty Údaje pro doručení objednávky

Karta "Údaje pro doručení objednávky"slouží správci k zobrazení údajů pro doručení a jejich rychlému upravení.

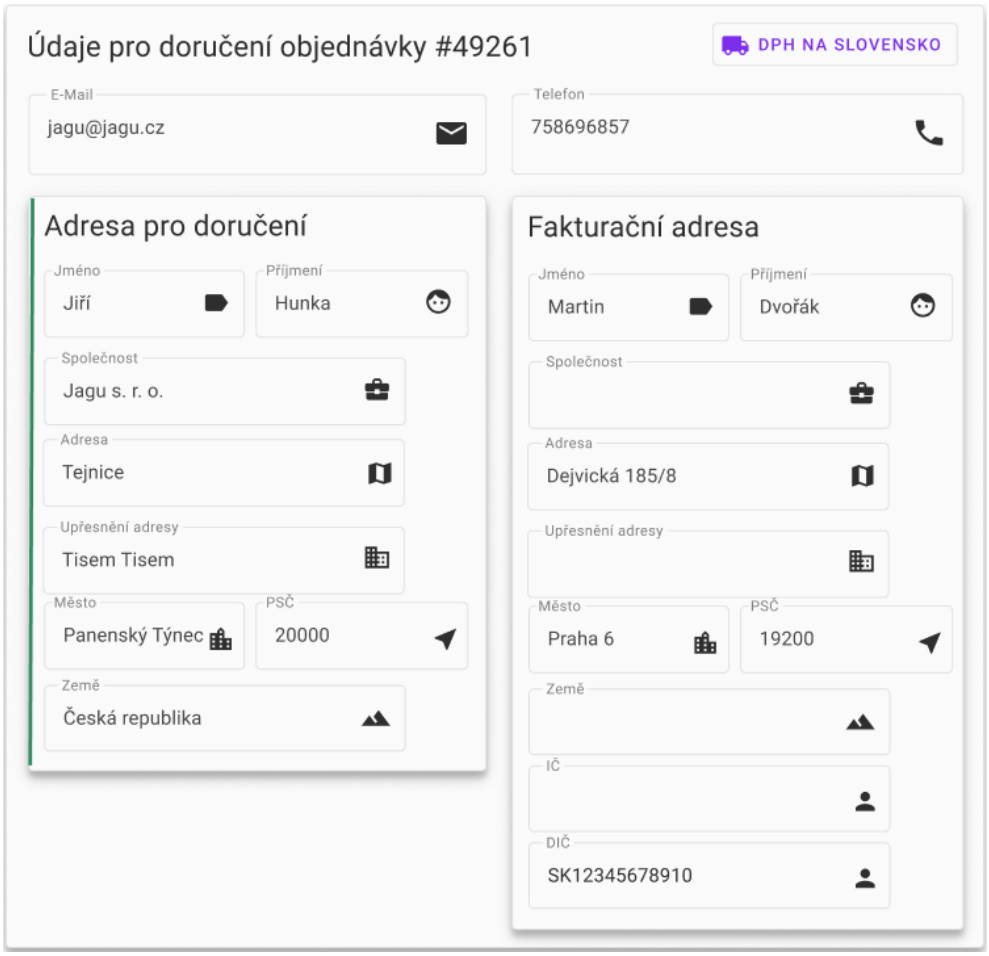
Minimal Viable Product (MVP)

- Formulář, který se dá upravit, s emailem a telefonem
- Karta s Adresou pro doručení
  - tlačítko, které umožní upravit adresu doručení
  - Formulář se jménem, příjmení, společností, adresou a jejím upřesněním, městem, PSČ a zemí.
- Karta s Fakturační adresou
  - Formulář s tím samým co má karta Adresa pro doručení + IČ a DIČ
  - Informace, které budou stejné jako u doručovací adresy, se nezobrazí
  - Všechny údaje je možné volně upravit

- Data se berou z backendu.
- Všechny úpravy musí být synchronizované s backendem.

Nice To Have (NTH)
- Tlačítko pro dph na slovensko (prio 3)
- Na ikonu e-mailu a telefonu lze kliknout a vyvolat napsání e-mailu/volání. (prio 4)

Obrázek 2.4 Návrh karty Údaje pro doručení objednávky



2.2.1.5 Analýza karty Produkty v objednávce

Karta "Produkty v objednávce"má umožnit správci rychle zobrazit jaké produkty se v objednávce vyskytují a informace o celkové ceně. V případě potřeby je možné jednotlivým produkty upravit. Nakonec by měla mít tlačítko pomocí, kterého se dostaneme na stránku, kde se budou upravovat dobropisy.

Minimal Viable Product (MVP)
- Tlačítko, které uzamkne upravování produktů
- Seznam jednotlivých formulářů produktů, které se budou dát libovolně seřazovat pomocí ikony seznamu na levo a mazat pomocí ikony koše napravo. Každý produkt bude mít:

- Ty co lze upravit:
  - Název, který bude mít dropdown menu se všemi produkty. Pří výběru názvu se vyplní zbytek informací.
  - Velikost, pouze v případě, že daný produkt, takovou informaci bude mít. Také bude dropdown menu se všemi velikostmi pro daný produkt
  - Model produktu
  - Záruka produktu
  - Množství produktu v objednávce
  - Cena/kus s DPH
- Ty co nelze upravit.
  - Interní množství
  - Externí množství
  - Cena/kus bez DPH
  - Sazba DPH
  - Cena Celkem bez DPH
- Tlačítko, které přidá nový produkt

Nice To Have (NTH)
- Tlačítko, které správce přesměruje na stránku pro úpravu dobropisů (prio 5)

Obrázek 2.5 Návrh karty Produkty v objednávce

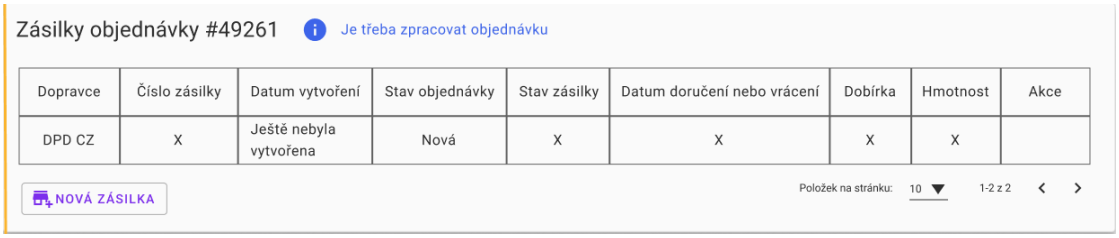
2.2.1.6 Analýza karty Zásilky objednávky

Karta "Zásilky objednávky"má správci ukázat všechny zásilky, na které se objednávka dělí. Dále mu má umožnit přidat novou zásilku.

Minimal Viable Product (MVP)
- DataTable ve kterém každá zásilka má:
  - Jméno dopravce
  - Číslo zásilky
  - Datum vytvoření zásilky
  - Stav objednávky
  - Stav zásilky

  - Datum doručení nebo vrácení
  - Dobírka
  - Hmotnost
  - Akce
- Tlačítko pro vytvoření nové zásilky
- Pruh na straně karty který je zelený pokud všechny objednávky jsou doručeny, oranžový pokud jsou na cestě a červený pokud nebyli doručeny.
- Komentář, který se zobrazí pokud není, žádná zásilka

Obrázek 2.6 Návrh karty Zásilky v objednávce



2.3 Technologie v aplikaci

V této sekci z analyzuji prototyp aplikace z technického hlediska. Představím zde všechny technologie, které byly doposud v projektu využity. Jejich případné nahrazení a další technologie, které případně využiji zmíním v rámci realizace.

2.3.1 Typescript

V rámci Vue component je možné použít buď javascript nebo typescript, jelikož se napříč celou aplikací používá typescript, tak jej využiji i já. Typescript je Javascript s podporou syntaxe pro typy. To přináší své výhody i nevýhody. Programátor je občas nucen hlídat typy, tam kde by v Javascriptu nemusel, ale díky tomu je pak kód pro ostatní čitelnější, protože lze na první pohled vidět, jaký datový typ tam má být použit. [13]

2.3.2 Symfony

Symfony je open-source PHP framework pro vývoj webových aplikací. Jedná se o jeden z nejpopulárnějších a nejrozšířenějších frameworků v PHP komunitě. Symfony usnadňuje vývoj webu poskytnutím mnoho užitečných funkcí a nástrojů, jako například autentizaci, řízení přístupu, zpracování formulářů, cachování, testování a dalších. [14]

Symfony se řídí konceptem "konvencí před konfigurací". Tento přístup usnadňuje spolupráci více lidí v týmu a umožňuje efektivní využití frameworku. Nabízí také rozsáhlou dokumentaci a komunitu, kde vývojáři mohou nalézt odpovědi na své otázky. [14]

2.3.3 Vue.js

Vue.js je open-source framework progresivního JavaScriptu používaný pro tvorbu uživatelských rozhraní a aplikací typu single-page. V posledních letech získal značnou popularitu díky své

jednoduchosti a snadné použitelnosti, což umožňuje vývojářům rychle a efektivně vytvářet složité aplikace. [15]

Díky tomu, že nabízí řadu funkcí, jako jsou například data binding, direktivy pro šablony, výpočetní vlastnosti a komponenty, je mocným nástrojem pro tvorbu interaktivních a responzivních webových aplikací. Je velmi přizpůsobivý a může být snadno integrován do jiných projektů nebo použit v kombinaci s jinými knihovnami a frameworky. V neposlední řadě, je velká výhoda jeho rostoucí a podpůrná komunita, která jej neustále rozvíjí. [15]

2.3.4 Vite

Vite je moderní a lehký nástroj pro sestavení JavaScriptových aplikací. Vite se zaměřuje na poskytování rychlých a efektivních vývojových a sestavovacích procesů pro webové aplikace, aniž by se snižovala funkčnost a flexibilita. [16]

Jednou z hlavních funkcí Vite je jeho schopnost během vývoje poskytovat aplikaci s využitím vestavěného vývojového serveru, který podporuje hot module replacement. To znamená, že jakmile provedete změny ve svém kódu, aplikace se automaticky aktualizuje v prohlížeči bez potřeby manuálního obnovení. Kromě toho Vite využívá nativní funkci ES modulů v moderních prohlížečích pro rychlejší a efektivnější sestavování vašeho kódu během procesu sestavení, což vede k rychlejším časům sestavení a menším výstupním souborům. [16]

2.3.5 Vuetify

Vuetify je knihovna, která využívá Vue.js. Její komponenty odpovídají Material Designu a její autoři tvrdí, že je třeba minimální zkušenost s návrhem, protože obsahuje, vše potřebné k vytvoření skvělé aplikace. [9]

Její komponenty se dají opakovaně používat ve více formách a zároveň umožňují vysokou míru přizpůsobení. Díky tomu je pak možné velmi rychle vytvořit vzhled aplikace, dle našich představ. [9]

Podporuje snadné nastavení barevný témat a vytvoření vlastních aliasů pro ikony, což nám dále usnadňuje použití skrze celou aplikaci. [9]

2.3.6 VueI18n

Jedná se o plugin pro vytváření vícejazyčných aplikací. Umožňuje mít v kódu translation stringy, které nám pak plugin pomocí souboru, kde jsou manuálně napsané překlady, vyhledá a přeloží. [17]

To umožňuje programátorovi, mít stringy na jednom místě a poměrně jednoduše přidat další jazyk. Zároveň také do kódu píše translation stringy a nemusí tak v kódu vyhledávat, když by chtěl upravit text, který se má zobrazit uživateli. [17]

Aplikace má momentálně implementovanou češtinu a angličtinu a v rámci této práce není požadavek na přidání dalšího jazyka. Budu však muset dbát, abych měl implementované oba překlady.

2.3.7 Vue Router

Vue-Router se využívá pro navigaci v aplikaci. Umožňuje vytvářet dynamické i statické cesty, ke kterým lze přiřadit nově napsané komponenty. Ke komponentám se pak uživatel dostane, když přejde na danou cestu. [18]

Pro každou cestu lze nastavit zda má vykonat kód, při vstupu nebo při opuštění cesty a umí pracovat s více módy historie. [18]

2.3.8 VueX

Vuex je knihovna pro správu stavu aplikací v rámci Vue.js. Umožňuje spravovat a sdílet stav na úrovni aplikace centrálním a předvídatelným způsobem. Vuex poskytuje způsob uložení a přístupu k datům globálně, což usnadňuje správu dat napříč komponentami a zajišťuje, že změny v stavu jsou konzistentní napříč celou aplikací. [19]

Jednou z hlavních funkcí Vuexu je jeho schopnost udržovat jediný zdroj pravdy pro stav aplikace. To znamená, že všechny komponenty mohou přistupovat ke stejným datům a jakékoli změny provedené ve stavu jsou automaticky propagovány do všech komponent, které na něj závisí. Vuex také poskytuje sadu nástrojů pro správu složitých datových toků a asynchronních akcí, jako jsou gettery, mutace a akce. [19]

2.3.9 ESlint

Pro správné a konzistentní formátování kódu v týmovém vývoji je velmi důležité používat nástroje pro statickou analýzu. ESLint je vynikajícím nástrojem pro kód napsaný v JavaScriptu nebo TypeScriptu, který umožňuje odhalit potenciální problémy v kódu a provést automatickou opravu některých z nich. Lze ho integrovat do většiny vývojových prostředí nebo použít jako součást pipeline pro kontinuální integraci. [20]

Jednou z největších výhod ESLintu je jeho velká komunita, která vytváří balíčky pravidel a umožňuje nastavit různá pravidla. Lze například nastavit pravidla pro formátování kódu, rozhodnout se, jak se má reagovat na jejich porušení (pouze upozornění, nekompilovatelný kód atd.) Je také možné vytvářet vlastní pravidla dle potřeby. Celkově je použití ESLintu velmi užitečné pro zajištění kvality kódu a snížení času potřebného na ladění a opravování chyb. [20]

2.3.10 Sentry

Sentry je software pro monitorování aplikací a hlášení chyb, který je využíván při vývoji. I když se snažíme minimalizovat vznik chyb, občas se nějaká objeví a Sentry nám pomáhá při jejich identifikaci a opravě. Lze nastavit, v jakých prostředích se mají chyby zaznamenávat. [21]

Pokud dojde k chybě, Sentry nejenom zobrazí její název, ale také poskytne informace o kontextu, zdrojovém kódu, obsahu zásobníku a akcích, které byly provedeny před jejím vznikem. Tyto informace umožňují rychlejší analýzu a opravu chyb. [21]

2.4 Použité podpůrné nástroje pro vývoj

V rámci vývoje softwaru se používá množství nástrojů, které se snaží programátorům usnadnit práci. Jelikož se Programátoři vždy snaží zautomazitovat, vše co jen lze, aby se mohli soustředit na psaní implementace, tak jsou krásným příkladem pipeliny na GitLabu sloužící k automatické kontrole. [22]

Další částí podpůrných nástrojů mohou být například systémy správy verzí, jako je GitLab. Přináší programátorům možnost pracovat na jednom projektu paralelně a mít společnou online pracovní složku. [23]

2.4.1 Gitlab

GitLab je systém pro správu verzí projektů, který umožňuje spolupráci na kódu, řídit verzování a nasazování softwaru a provádět automatické testování. Nabízí řadu nástrojů pro zlepšení efektivity a organizace vývojového procesu, včetně integrovaného správce úkolů, sledování času a nákladů, a mnoho dalších. GitLab umožňuje týmům efektivně spolupracovat a koordinovat svou práci, a to vše na jednom místě. [23]

2.4.2 GitLab CI

Automatizace procesu vývoje softwaru je pro vývojáře velmi prospěšná. GitLab CI je nástroj, který automatizuje úkoly jako sestavení, testování a nasazení aplikace. Je užitečný při identifikaci chyb v raných fázích vývoje a zajišťuje, že kód nasazený na produkci odpovídá standardům vývojového týmu. Nastavení pro GitLab CI se provádí pomocí souboru s názvem .gitlab-ci.yml, kde jsou definovány různé etapy s jednotlivými úkoly. Úkoly mohou zahrnovat statickou kontrolu kódu a úkoly nasazení, přičemž každý úkol je plánován podle konkrétních kritérií. [22]

2.4.3 Code review

Jedním z nejdůležitějších procesů při vývoji, kteréhokoli softwaru, je code review. Princip tohoto procesu spočívá v tom, že se jiný člen týmu podívá na kód a zkontroluje, zda odpovídá standardům týmu, ještě předtím, než se dostane do testování. [24]

Tento proces pomáhá odhalit možné chyby v logice, neefektivitu kódu, dodržování stanovených coding guidelines. [24]

V rámci této práce dělám na stránce sám, ale roli seniora / jiného člena týmu, bude mít Dvořák, který má v této práci roli konzultanta. [1]

2.5 Analýza backendu

Implementace backendu vznikla v předmětu BI-SP1 a BI-SP2 v době kdy Dvořák pracoval na své práci. Systém má název Porcupine a slouží k interakci s databází. Pro frontend to znamená, že backend nám odhalí endpointy, pomocí kterých můžeme jednoduše interagovat s databází rovnou z frontendu. To nám umožňuje zobrazovat informace, aktualizovat informace, nebo informace přidávat a mazat. [1]

Konkrétně u objednávek byla na začátku pouze provizorní implementace backendu. Kterou se pak rozhodli vedoucí projektu změnit. Tato změna už však nebyla provedena na straně frontendu, jelikož vývoj backendu byl zpožděn.

V tomto projektu začnu rovnou pracovat s novou implementací endpointů objednávek, abych tak mohl napojit celý frontend na aktuální verzi backendu. Více o procesu změny OrderAPI napíši v realizaci.

V neposlední řadě je důležité zmínit, že backend se stále vyvíjí v BI-SP1 a BI-SP2 akademického roku 22/23, což znamená, že budu muset dbát na jejich změny, aby frontend správně fungoval, ale zároveň budu mít možnost si říct o endpoointy, které jsou potřeba, ale momentálně neexistují. Více o spolupráci s týmy z BI-SP1 a BI-SP2 taktéž v realizaci.

2.6 Závěr

V rámci této kapitoly jsem rozhodl, že v projektu budu pracovat agilně s prvky metodiky SCRUM. Analyzoval jsem návrh z práce Dvořáka a vypsal jednotlivé požadavky jež jsem pak rozdělil na funkční, nefunkční, kterým jsem nadále přiřadil prioritu od MVP po NTH (prio 1-5). Nakonec jsem prošel všechny využité technologie, ke který se budu odkazovat v realizaci. [1]

Tato analýza mi umožnila mít v celku jasno, jak postupovat, s čím začít jako první a co si nechat nakonec, nebo v případě nedostatku času vyškrtnout z finální verze.

Jelikož, tato analýza vznikla na začátku vývoje, tak je zcela možné, že se vyskytnou další požadavky. K jejich případné analýze přistoupím při jejich vzniku.

Kapitola 3
Realizace

Zanalyzované řešení je nyní třeba implementovat, v rámci realizace budou zmíněny i změny návrhu a jejich odůvodnění. Díky nátuře agilního vývoje, je přirozené, že se požadavky mohou změnit.

Nadále zmíním spolupráci s týtem SP-1, kteří současně pracují na vývoji back endu a jiných částech frontendu.

Export změn kódu z gitu se nachází v příloze ve složce *source* a celý kód se nachází na firemním gitlabu Jagu. [25]

3.1 Změny v technologiích

V této sekci se podíváme na nové technologie, které přibily, nebo nahradili technologie, které se už v aplikaci vyskytovali. Dále, zmíním proces refactoringu, ze starých aplikací na nové, jelikož se jedná o nemalou část práce, která podstatně zvětšila scope implementace.

3.1.1 Vue 3

Rozhodl jsem se předělat projekt do Vue 3, jelikož již vyšla knihovna Vuetify 3, na kterou se ve vývoji aplikace při bakalářské práci Martina Dvořáka čekalo. Vue 3 má oproti Vue 2 výhodu, že podporuje composition API i option API, tudíž není nutné psát nové komponenty v option API jako ve Vue 2. Composition API se skládá ze 3 API: Reactivity API, Lifecycle Hooks a Dependency Injection. Cílem composition API je příblížit se ke psaní kódu v javascpritu. [15]

3.1.1.1 Reactivity API

Reactivity API přináší metody ref(), reactive(), computed stav a pozorovatele (watchers). Zmíním zejména ref() a reactive(), jelikož zbytek v aplikaci není využit. [15]

ref()

Metoda ref() vytvoří reaktivní objekt Ref s atributem value. Atribut value, stejně tak jako jeho atributy, se dá volně nastavovat pomocí operátoru -". Zároveň je reaktivní - všechny operace čtení jsou stopovatelné a operace psaní spustí spojené efekty. Metoda je definována následovně: [15]

```
1  function ref<T>(value: T): Ref<UnwrapRef<T>>
2
```

```
interface Ref<T> {
  value: T
}
```

Výpis kódu 3.1 Ref typ

Díky tomu je pak možné psát kód následovně:

```
const count = ref(0)
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

Výpis kódu 3.2 Použití ref()

reactive()

Metoda reactive() vrátí reaktivní proxy původního objektu. Tato konverze je hluboká: ovlivní všechny vnořené atributy, zároveň rozbalí všechny vnitřní ref() objekty. Používá se pak takto: [15]

```
const count = ref(1)
const obj = reactive({ count })

// ref bude rozbalena
console.log(obj.count === count.value) // true

// aktualizuje `obj.count`
count.value++
console.log(count.value) // 2
console.log(obj.count) // 2

// take aktualizuje `count` ref
obj.count++
console.log(obj.count) // 3
console.log(count.value) // 3
```

Výpis kódu 3.3 Použití reactive()

3.1.1.2 Lifecycle Hooks

Lifycycle Hooks přináší do Compositon API metody, které reagují na životní stav komponenty. Zda byla komponenta aktualizována, nasazena, sezane a spoustu dalších. Já zde vypíši pouze ty, které použiji v aplikaci. [15]

onBeforeUnmount()

Tento hook se zavolá těsně předtím, než je komponenta sesazena. Dá se tak použít například ke flush dat čekajících na periodické volání do databáze, abychom zapříčinili jejich ztrátám. [15]

```
const data = ref({name: Martin, surname: Salaj})

onBeforeUnmount(() => {
  API.put('/clients', data);
})
```

Výpis kódu 3.4 Použití metody onBeforeUnmount()

V v tomto příkladě máme zaručeno v případě přepnutí na jinou stránku a sesazení aktivní komponenty, že se nám do naší databáze putne objekt v posledním stavu ve kterém se vyskytl a zamezíme tak případné ztrátě dat.

3.1.1.3 Dependency Injection

Dependency injection umožňuje pomocí metod provide() a inject() ijekci dat z předchůdce do potomka. Je to podobně propům, které se používali ve Vue 2 a stále se používají ve Vue 3. Hlavní výhoda Dependency injection oproti propům je, že není nutné data posílat vždy na přímého potomka, ale můžeme je pomocí inject() dostat do libovolného potomka. [15]

provide()

Metoda provide potřebuje 2 argumenty: klíč, pomocí kterého se pak hodnota injectne z potomka, a samotnou hodnotu, kterou chceme potomkům předat. Hodnota může být jak staticka tak reaktivní.

```
<script setup>
import { ref, provide } from 'vue'
import { fooSymbol } from './injectionSymbols'

// predame statickou hodnotu
provide('foo', 'bar')

// predame reaktivni hodnotu
const count = ref(0)
provide('count', count)
</script>
```

Výpis kódu 3.5 Použití provide()

inject()

Metoda inject má jeden argument a to sice klíč. Klíč vytvořená metodou provide() se pak postupně pokusí vyhledat v předcích dokud na něj nenarazí. Když klíč najde tak vezme jeho hodnotu, kterou pak budeme moct dál používat. Pokud je v předcích komponenty více stejných klíčů, pak vybere ten, který nalezne dřív. Pokud nenalezne klíč pak vrátí undefined. [15]

```
<script setup>
import { inject } from 'vue'
import { fooSymbol } from './injectionSymbols'

// injectnuti staticke hodnoty
const foo = inject('foo')

// injectnuti reaktivni hodnoty
const count = inject('count')

// injectnuti se zakladni hodnotou
const bar = inject('foo', 'default value')

// injectnuti se zakladni hodnotou vytvorenou tovarnou
const baz = inject('foo', () => new Map())
</script>
```

Výpis kódu 3.6 Použití inject()

3.1.2 Refactor na Vue 3

Původní aplikace byla napsána v technologii Vue 2 zejména kvůli knihovně Vuetify, která nebyla plně dokončena pro použití ve Vue 3, a na které je aplikace postavena. Jelikož na začátku tohoto roku vyšla verze 3.1.x pro Vuetify, která už v sobě má veškeré featury, které potřebujeme v aplikaci tak jsem se rozhodl migrovat na Vue 3. Pro plnou migraci bylo třeba postupně převézt všechny komponenty na Vue 3 syntax.

3.1.2.1 Vue Compatibility build

Vue Compatibility build (dále jen Compat build) slouží k migraci z Vue 2 na Vue 3, proto jsem se ho také rozhodl využít. Jeho hlavní účel je mít možnost postupně jednotilvým komponentám říkat, zda se mají kompilovat jako Vue 2, nebo Vue 3. Pak je možné postupně a relativně bezpečně migrovat aplikaci. Ve finále jsem zjistil, že pro tento projekt tento postup nebyl možný a to právě kvůli knihovně Vuetify, při jejíž aktualizaci na verzi pro Vue 3 jsem zjistil, že je nekompatibilní s Compat buildem. Musel jsem tak Compat build dát pryč a rovnou aktualizovat verzi Vue.

3.1.2.2 Proces migrace bez Compat buildu

Jelikož jsem musel aktualizovat verzi Vue, tak jsem měl celý projekt ve staré syntaxi. To znamenalo, že mi projekt nešel vykompilovat a tak jsem musel iterativně aktualizovat jednotlivé komponenty z Vue 2 na Vue 3 dokud jsem se nedostal do situace, kdy bylo možné spustit aplikaci. Jeden z největších rozdílu mezi verzemi Vue je ten, že nově se používají metody createApp(), createRouter(), atd., které přijímají trochu jiné parametry než ve staré verzi.

Migrace routeru

Migrace routeru byla relativně jednoduchá, ale repetetivní. Bylo třeba změnit importy, aby odpovídali Vue 3 a zároveň vue-router změnil typ jednotlivých cest, takže bylo třeba pro všechny routery změnit typy cest na RouteRecordRaw a následně vytvořit router pomocí metody createRouter(history: createWebHistory(), routes: RouteRecordRaw[]).

```
1          .
2          .
3          .
4  const routes: Array<RouteConfig>
       =[
5    ...homepageRoutes,
6    ...manufacturerRoutes,
7    ...productRoutes,
8    ...categoryRoutes,
9    ...reviewRoutes,
10   ...orderRoutes,
11   ...customerGroupsRoutes,
12   ...customersRoutes,
13   ...userRoutes,
14   ...userRoleRoutes,
15   notFoundRoute
16 ];
17
18 const Router = new VueRouter({
19   mode: 'history',
20   routes
21 });
22         .
23         .
24         .
25 export{Router}
```

Výpis kódu 3.7 Vue 2 router

```
1          .
2          .
3          .
4  const routes: RouteRecordRaw[]
       =[
5    ...homepageRoutes,
6    ...manufacturerRoutes,
7    ...productRoutes,
8    ...categoryRoutes,
9    ...reviewRoutes,
10   ...orderRoutes,
11   ...customerGroupsRoutes,
12   ...customersRoutes,
13   ...userRoutes,
14   ...userRoleRoutes,
15   notFoundRoute
16 ];
17
18 const Router = createRouter({
19   history: createWebHistory(),
20   routes: routes
21 });
22         .
23         .
24         .
25 export{Router}
```

Výpis kódu 3.8 Vue 3 router

Router pak použijeme v main.ts, kde ho přidáme do aplikace.

Migrace Vuex storu

Migrace Vuex storu byla asi nejjednodušší, stačilo změnit inicializaci storu použitím metody createStore(...).

```
1          .
2          .
3          .
4  const Store = new Vuex.Store<
       State>({
5    ...
6  });
7          .
8          .
9          .
10 export {Store, ...};
```

Výpis kódu 3.9 Vue 2 vuex store

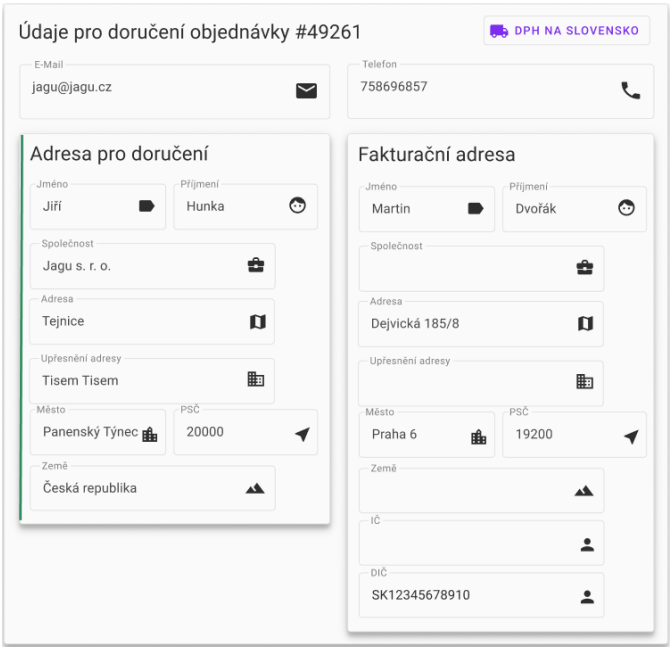
```
1          .
2          .
3          .
4  const Store = new createStore({
5    ...
6  });
7          .
8          .
9          .
10 export {Store, ...};
```

Výpis kódu 3.10 Vue 3 vuex store

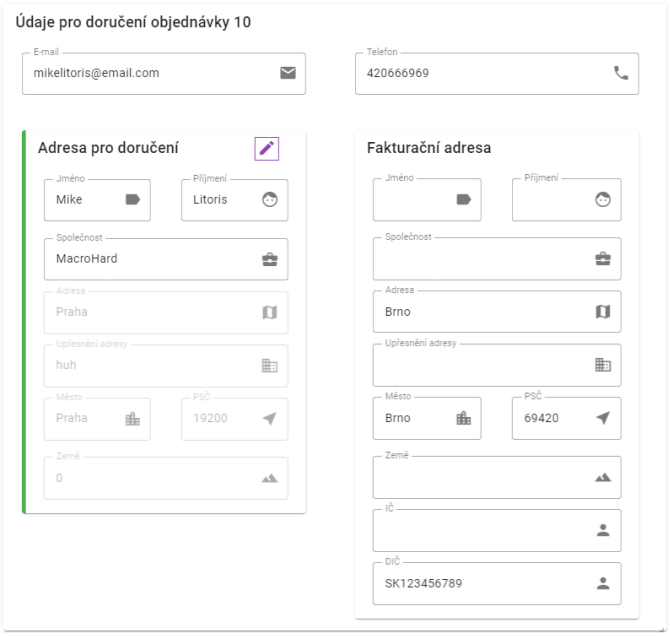
Migrace Vuetify

Změna z Vuetify 2 na Vuetify 3 bohužel nebyla tak jednoduchá jak jsem čekal. Nejen, že bylo potřeba změnit inicializaci, ale v rámci Vuetify knihovny bylo provedeno dost vnitřních změn. Některým komponentám odebrali eventy, změnili fungování v-model, atd. Jednou z největších změn byla změna stylu:

Obrázek 3.1 Karta ve Vuetify 2



Obrázek 3.2 Karta ve Vuetify 3



Na první pohled je vidět, že nové karty nejsou tak výrazné, co však vidět není, je změna v jejich automatickém paddingu a marginu. Nastavení, které jsem měl nastavené pro Vuetify 2 bylo prakticky nepoužitelné. Musel jsem tak všechny komponenty projít a upravit padding a margin u všech Vuetify komponent.

3.1.2.3 Změna v main.ts

V neposlední řadě bylo třeba změnit prakticky celý main.ts. Jelikož se ve verzi Vue 3 změnilo jak se nahlíží na instanci Vue, tak bylo potřeba všechny věci, které jsem již zmínil nějak do naší aplikace inicializovat. Hlavní rozdíl oproti Vue 2 je v tom, že v nové verzi aplikaci inicializuji pomocí createApp(App) a pak všechny pluginy "importuji"pomocí .use(plugin), kdežto ve staré verzi jsme měli objekt Vue a vytvořili jsme jeho novou instanci pomocí new Vue(pluginy + render funkce).$mount('#app');.

```
1  import Vue from 'vue';
2          .
3          .
4          .
5
6  Sentry.init(app);
7
8  // eslint-disable-next-line
9  Vue.component('x-toolbar',
       XToolbar);
10 // eslint-disable-next-line
11 Vue.component('draggable',
       draggable);
12 Vue.use(snackbarPlugin);
13
14 export const eventBus = new
       Vue();
15
16 new Vue({
17   i18n: I18n,
18   store: Store,
19   router: Router,
20   vuetify: Vuetify,
21   render: h => h(App)
22 }).$mount('#app');
```

Výpis kódu 3.11 Vue 2 main.ts

```
1  import {createApp} from 'vue';
2          .
3          .
4          .
5  const app = createApp(App);
6
7  Sentry.init(app);
8
9  app
10   .use(I18n)
11   .use(Router)
12   .use(Store)
13   .use(Vuetify)
14   .use(VueGridLayout);
15
16 // eslint-disable-next-line
17 app.component('x-toolbar',
       XToolbar)
18 // eslint-disable-next-line
19 app.component('draggable',
       draggable)
20 app.component(snackbarPlugin);
21
22 app.mount('#app');
```

Výpis kódu 3.12 Vue 3 main.ts

Na první pohled si lze povšimnout, že Vue 3 je v tomto ohledu přehlednější a přináší způsob jak přidávat pluginy lehce a škálovatelně. Když se v budoucnu bude chtít přidat nový plugin, tak ho pouze stačí vytvořit jako předešlé pluginy a přidat do aplikace pomocí .use().

3.2 Změna OrderAPI

OrderAPI je souboru, který slouží k vytvoření method na volání endpointů backendu. Dvořák jej naimplementoval pro stránku Objednávky, která slouží k zobrazení všech objednávek. Tato implementace však již neodpovídá novému návrhu. Následuje ukázka změny. [1]

```
interface Order {
  id: number,
  seen: boolean,
  invoiceNumber: number,
  domain: string,
  slovak: boolean,
  customerId: number,
  comment: string,
  state: string,
  transport: string,
  paymentType: string,
  paid: boolean,
  createdAt: Date,
  wholePrice: number,
  profit: number,
  cameFrom: string,
  agreement: boolean,
  allStocked: boolean,
  action: string[],
  order: number
}
```

Výpis kódu 3.13 Starý návrh

```
interface Order {
  id: number,
  viewed: boolean,
  invoiceId: string,
  paymentMethodId: number,
  shippingMethodId: number,
  paidOrder: boolean,
  dateAdded: string,
  domain: string,
  orderStatusId: number,
  profit: number,
  inStock: boolean,
  currency: string,
  comment: string,
  reason: string,
  contactInformation: {
    telephone: string,
    email: string,
    shippingAddress:
        ShippingAddress,
    paymentAddress: PaymentAddress
  },
  history: [
    History
  ],
  products: [
    OrderProduct
  ]
}

interface OrderStatus {
  id: number,
  name: string,
  comment: string,
  informCustomer: boolean
}

interface History {
  dateAdded: string,
  orderStatusId: number,
  notify: boolean,
  comment: string,
  createdBySystem: boolean
}

interface OrderProduct {
  id: number,
  productId: number,
  name: string,
  size: string,
  model: string,
  warranty: number,
  taxClassId: number,
  price: number,
  total: number,
  quantity: number,
  internal: number,
  external: number,
  ordinalNumber: number
}
```

Výpis kódu 3.14 Nový návrh

Jak lze vidět, tak stará implementace byla poměrně přímočará. Sloužila svou práci skvěle pro rozsah předešlého požadavku, ale pro požadavky této práce v ní chybělo spoustu věcí, které byly při vývoji velmi důležité. Krom změn jmen některých atributů jich bylo i mnoho přidáno.

Jak jsem již zmínil, tak celá stránka je sestavena z 5 karet. V nové implementaci je teď přidána například historie, která slouží pro kartu "Historie objednávky id". Dále je tam přidáno pole products, které je nezbytné v kartě "Produkty v objednávce id"a nakonec bych zmínil přidání adres, bez kterých by karta "Údaje pro doručení objednávky id"nemohla existovat.

3.2.1 Spolupráce s BI-SP2 na backendu

Změny v OrderAPI, které jsem doposud zmínil, byly určené pouze k aktualizaci frontendu na současný stav backendu. To však není vše, co bylo třeba změnit. Při vývoji se naskytlo více situací, kdy mi současný backend nestačil, abych mohl splnit požadavek.

Tak jsem se obrátil na týmy BI-SP2, které vyvíjí a aktualizují backend, aby mi vytvořili endpointy, se kterými bych mohl dál pracovat. Jelikož jsem vyvíjel většinu času nad mockem, tak jsem začal řešit stav backendu pozdě. Endpointy byly zcela rozbité a nedodělané a Nikita Golmgren je musel na základě mích požadavků projít a opravit. Toto selhání v komunikaci s backend týmem, mě stálo poměrně dost času, který jsem pak postrádal pro opravu chyb z testování s uživateli.

Ve finále se povedlo zprovoznit alespoň do přijatelného stavu většinu endpointů potřebných k fungování aplikace. Nicméně, nepodařilo se dostat endpoint pro upravování produktů v objednávce do stavu, ve kterém by byl použitelný.

3.2.1.1 Endpoint pro karu Historie objednávky

V této kartě je požadavek, že by administrátor měl mít možnost manuálně změnit stav objednávky. To udělá pomocí formuláře jehož prvním polem je autocomplete se stavy. Tyto stavy se však někde musí vzít. Kdyby bylo možné dát objednávku do jakéhokoli stavu, tak stačí vzít z databáze všechny stavy. Tak to však není, objednávka má v každém stavu, pouze pár jiných stavů do kterých může přejít.

Z tohoto důvodu jsem požádal tým BI-SP1, aby mi vytvořili nový endpoint, který vrátí pole stavů, do kterých v daný okamžik může objednávka přejít. Volám ho pak pomocí následují funkce:

```
1 getAvailableOrderStatuses: function (orderId: number): Promise<Response<
    OrderStatus[]>> {
2   return this.API.get([this.DOMAIN, orderId, 'available-statuses']);
3 },
```

Výpis kódu 3.15 Endpoint pro možné stavy

3.2.1.2 Endpoint pro karu Zásilky objednávky

Dalším endpointem, o kterém jsem v průběhu vývoje zjistil, že není naimplementován backendem, byl endpoint, který mi z databáze získá všechny zásilky objednávky. V původním požadavku bylo navrženo, že by se zásilky objednávky měli dát upravovat a mazat. Nicméně v současné době, nám interní systém nenabízí takovouto manipulaci se zásilkami, takže tato komponenta bude muset prozatím fungovat jen jako výpis všech zásilek. Více o této změně napíši v samostatné sekci věnované změnám oproti původnímu návrhu.

3.3 Implementace jednotlivých částí

V této sekci se budu věnovat především implementaci jednotlivých karet a míře splnění jejich požadavků oproti původnímu návrhu. Zároveň zmíním změny, co vznikli po konzultacích s vedoucím nebo byli ne/přímím důsledkem implementace frontendu a současného backendu. Dále se budu věnovat změnám, které jsem se rozhodl udělat z vlastního uvážení.

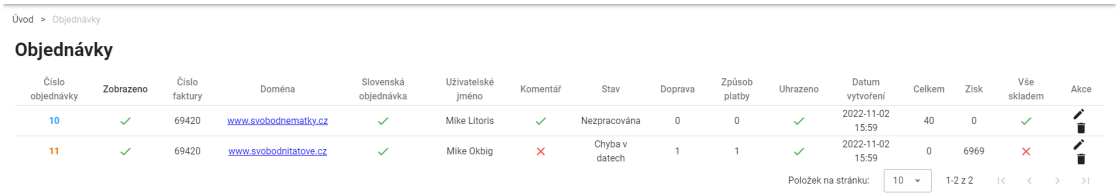
Nebudu již do podrobna zmiňovat asi největší změnu a to sice přechod z Vuetify 2 na Vuetify 3, který měl asi největší dopad na finální vzhled stránky, jelikož jsem jej již zmínil v sekci 3.1.2.2.

3.3.1 Implementace stránky Objednávky

Tato stránka byla již naimplementována Dvořákem, ale kvůli refaktoru na Vue 3, ji bylo třeba celou předělat. Jsou zachovány stejné sloupce, ale byla dodělána vizuální stránka návrhu. Všechny pravdivostní hodnoty jsou reprezentovány zelenou fajfkou a červeným křížkem. Id objednávky má stejnou barvu jako pruh v Podrobnostech objednávky. A doména opravdu odkazuje na stránku. [1]

Tuto stránku jsem dodělal, aby se má práce jako celek dala pořádně prezentovat. Je na ní třeba dodělat dost věcí. Například v původním backendu šlo objednávky mazat, ale momentálně takový endpoint neexistuje. Takže tato implementace je spíše prototyp, který sice odpovídá návrhu, ale není zcela funkční. Já tuto stránku použiji především na dostání se k stránce Úpravy objednávky.

Obrázek 3.3 Implementace stránky Objednávky

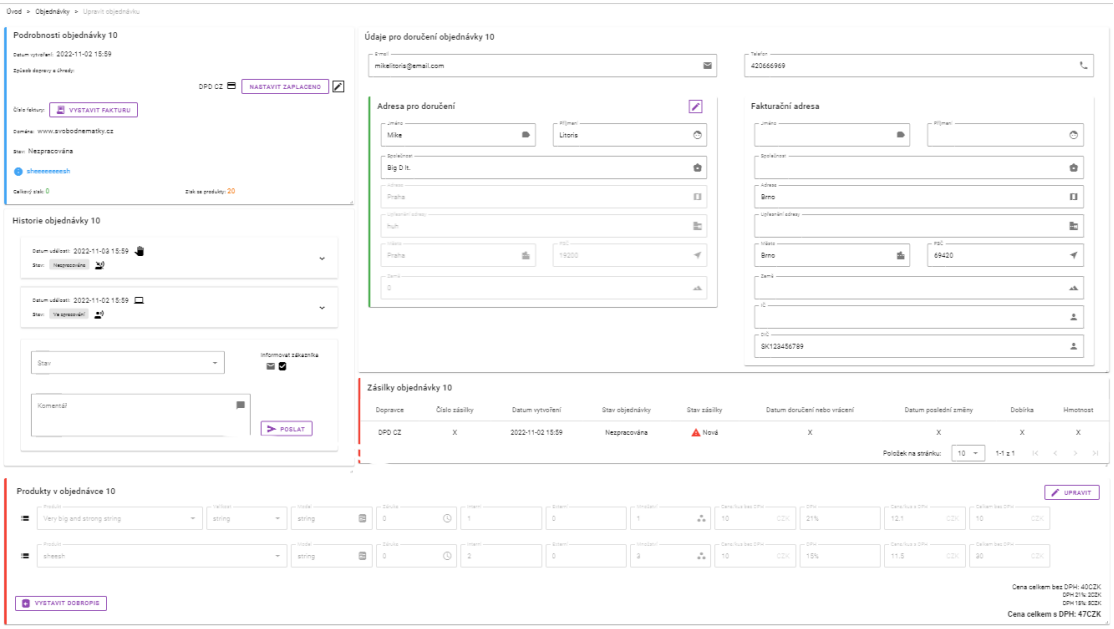


3.3.2 Implementace stránky Upravit objednávku

Tato stránka je jeden velký grid-layout, kde je zobrazeno 5 hlavních karet. Většina požadavků z 2.2.1.1 je splněna. Karty lze přemístit a škálovat. Polohy a velikosti jednotlivých karet se ukládají do lokální storage browseru, takže lze v každé objednávce upravit rozložení dle potřeby.

V rámci implementace této komponenty je řešeno napojení na backend. Tato stránka je parentem ostatních 5 karet, z každé karty, kde se upravují data, je posílán event, který je zde zpracován. Díky tomu, že je komunikace s backendem řešena na jednom místě, tak je jednoduché reagovat na reakce backendu.

Obrázek 3.4 Implementace stránky Upravit objednávku



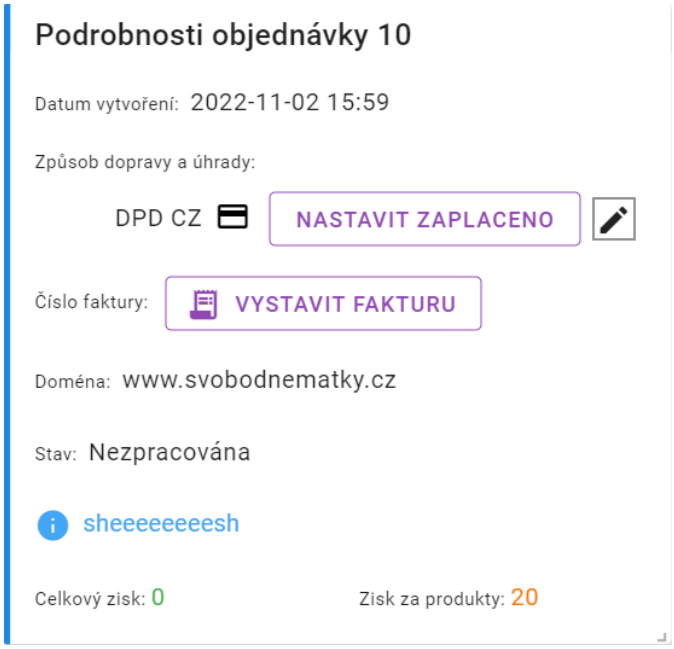
3.3.3 Implementace karty Podrobnosti objednávky

V této kartě bylo třeba splnit požadavky z 2.2.1.2: zobrazit všechny důležité informace objednávky. Také má umožnit administrátorovi nastavit způsob platby a dovážky a možnost vystavení a zobrazení faktury pomocí tlačítka. Nakonec je po levé straně barevný pruh, který má bravu dle návrhu.

Tlačítko nastavit zaplaceno nebylo zhotoveno jelikož jeho chování je úzce spjaté s dialogem z 3.3.3.1, který se nestihl dokončit. Tlačítko pro upravení metody platby a dovozu je taktéž neimplementováno ze stejného důvodu.

Je třeba dbát na fakt, že ne v každém stavu lze měnit objednávku, proto tlačítka posílají eventy do parenta, kde se tyto situace řeší.

Obrázek 3.5 Implementace karty podrobností objednávky

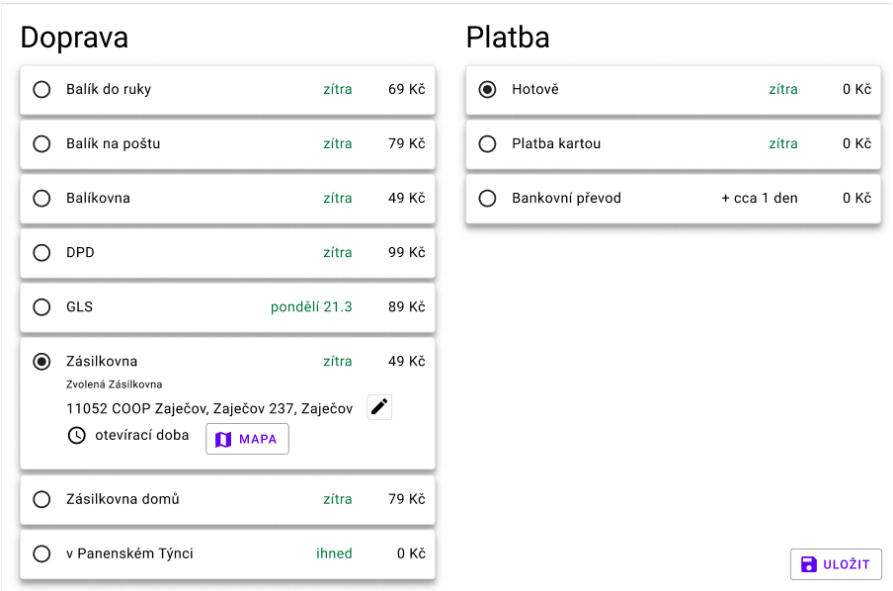


3.3.3.1 Spolupráce s týmem BI-SP2

Tým BI-SP2 má za úkol vytvořit dialog, kde bude možné vybrat způsob platby a dopravy. Na základě rychlosti jejich práce ho zakomponuji do karty, nebo kartu připravím tak, aby do ní byl lehce přidatelný.

Jelikož se nakonec nepovedlo dialog dokončit včas, tak nebyl přidán do řešení mé práce. Avšak připravil jsem implementaci tak, aby šel poměrně jednoduše přidat. Zároveň jsem se snažil netriviální části kódu okomentovat, aby bylo jasné jak komponenty upravit.

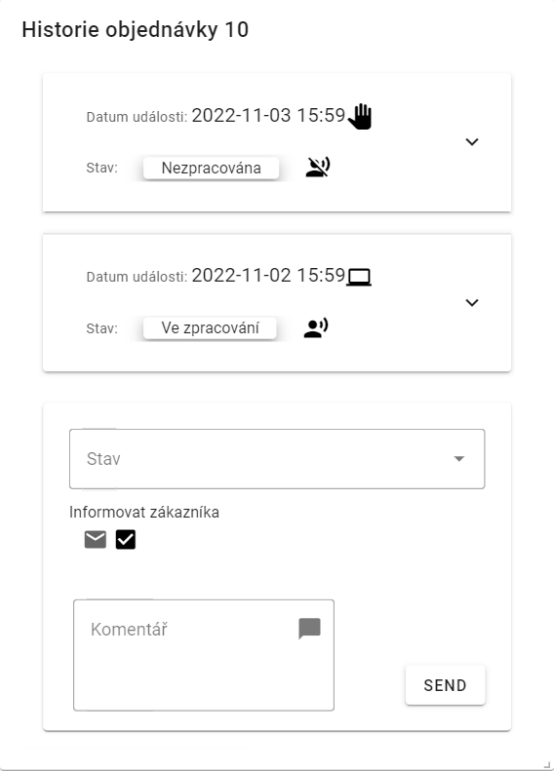
Obrázek 3.6 Dialog pro výběr možnosti platby a dovážky



3.3.4 Implementace karty Historie objednávky

V této kartě se podařilo dodržet všechny požadavky ze sekce 2.2.1.3. Karta správně ukazuje historii objednávky a dává administrátorovi možnost vyplnit formulář, bylo však potřeba dbát na fakt, že ne v každém stavu lze objednávku upravit. Proto je z této komponenty pouze odeslán event do parenta, kde se tato situace bude řešit.

Obrázek 3.7 Implementace karty Historie objednávky



3.3.5 Implementace karty Údaje pro doručení objednávky

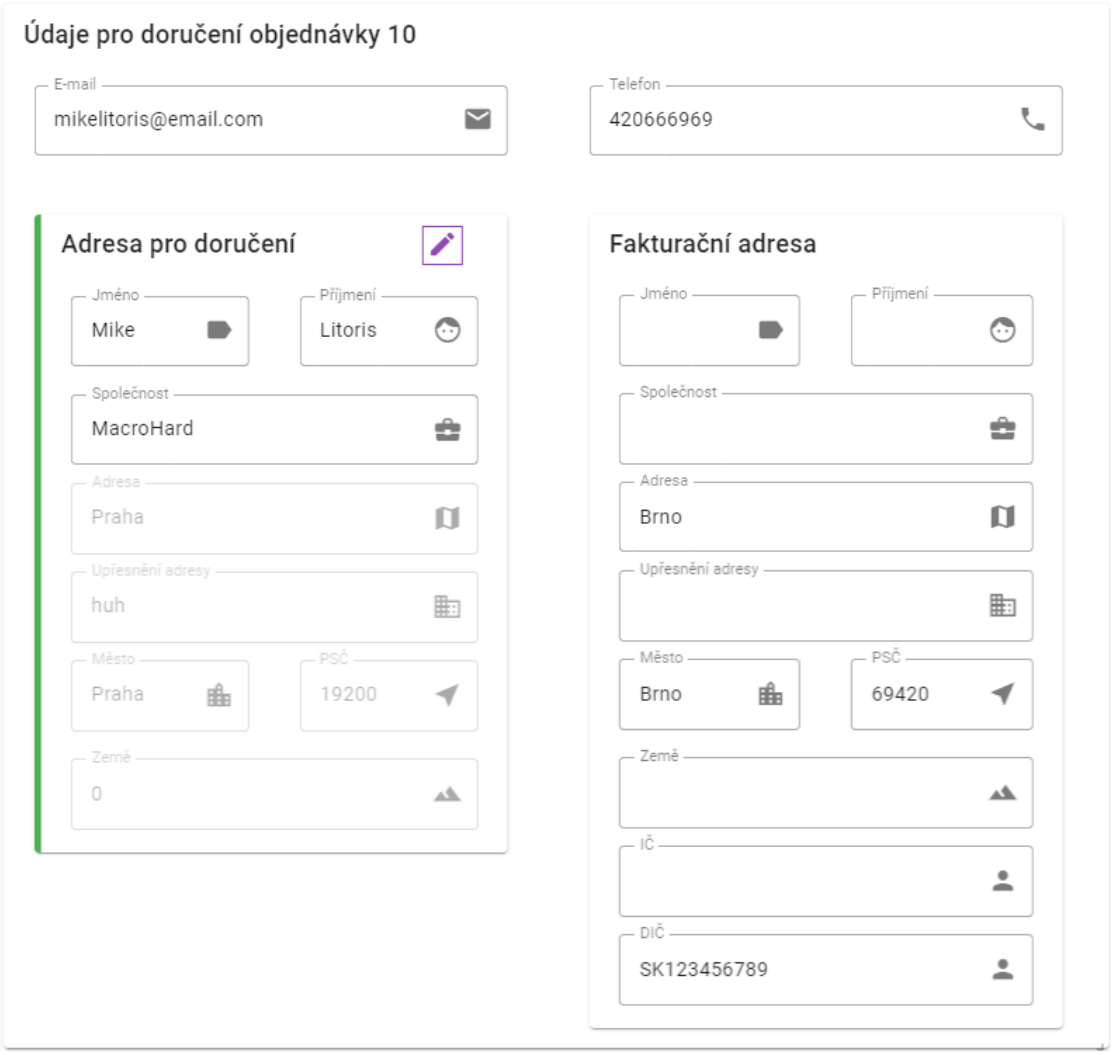
V se podařilo splnit většinu požadavků z 2.2.1.4. Lze zobrazit a upravovat kontaktní údaje. Opět byla třeba dbát na fakt, že údaje lze upravit jen v určitých stavech a tak všechny úpravy posílají event do parenta, kde se to řeší. Barevný pruh na straně karty Adresy pro doručení vizualizuje, zda je objednávka doručitelná. A tlačítko s tužkou umožňuje editovat Adresu pro doručení.

Změny oproti návrhu

Přidání tlačítka, které má za účel zapnout a vypnout možnost upravit doručovací adresu. V případě, že se objednávka má doručit domů, tak je možné adresu upravit přímo, v opačném případě se otevře dialog z 3.3.3.1. Tato funkcionalita je momentálně nefunkční, protože dialog nebyl dokončen včas, a tak jsem pouze připravil místo, kam se v komponentě přidá.

Nice to have (NTH) featury této karty, nebyly implementovány, protože refaktor na Vue 3 zabral více času, než bylo plánováno a ostatní karty měly NTH featury s větší prioritou.

Obrázek 3.8 Implementace karty Údaje pro doručení objednávky



3.3.6 Implementace karty Produkty v objednávce

Toto byla asi nejtěžší karta, co se týče implementace. Dle požadavků z 2.2.1.5, bylo třeba naimplementovat zobrazení produktů, možnost je upravit, přidat a smazat a nakonec tlačítko, které odkáže na novou stránku, kde se budou řešit dobropisy. Opět je třeba zmínit, že přidat produkty, lze jen v některých stavech objednávky a tak tato karta posílá event do parenta, který to dále řeší.

V rámci této práce, byly splněny všechny skoro věechny požadavky na tuto komponentu. Chybí jen možnost vybrat z produktů způsob doručení a platby, tento cíl jsem nestihl udělat, protože refaktor na vue 3 zabral více času než jsem čekal a tato funkcionalita byla komplikovanější než jsem předpokládal. Také je třeba podotknout, že tlačítko odkazující na dobropisy odkazuje pouze na novou stránku, co čeká na budoucí implementaci.

Změny oproti návrhu

Přidávání produktů bylo změněno. Jedná se o vizualní změnu. Je přidána ikona "+", která přidá prázdný formulář, který se pak dále dá vyplnit. V původním návrhu byl rovnou prázdný formulář.

Tuto změnu, jsem provedl, protože mi to přišlo z návrhové hlediska jako lepší řešení, které je zároveň více intuitivní pro uživatele.

Obrázek 3.9 Implementace karty Produkty v objednávce

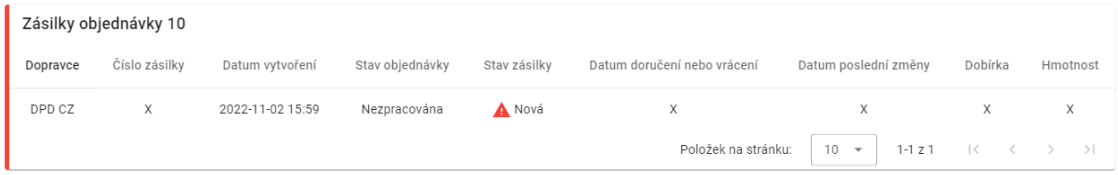
3.3.7 Implementace karty Zásilky v objednávce

Této kartě chybí endpoint v backendu a ani není v plánu ho stihnout naimplementovat. Proto se jedná pouze o prototyp, který je plně připraven na integraci s backendem.

Změny oproti návrhu

V návrhu je tlačítko, které má umožnit přidat novou zásilku, to však momentálně systém neumožňuje, takže toto tlačítko nebude implemontované. Dále se přidá sloupec s posledním datem změny.

Obrázek 3.10 Implementace karty Zásilky v objednávce



3.3.8 Napojení na backend

Napojení na backend bylo velmi obtížné, hlavně kvůli tomu, že jsem se spoléhal na to, že budou hotové endpointy a správně podle dokumentace a začal jsem s jejich testováním poměrně pozdě. Hned při prvním pokusu jsem zjistil, že většina endpointů na masteru backend projektu v gitu neexistuje, a tak jsem si musel vyhledat větev, kde měly být naimplementovány. Po nasazení větve do masteru jsem zjistil, že endpointy sice vytvořeny byly, ale neodpovídali schématu v dokumentaci, to mi rozbilo celý frontend.

Kontaktoval jsem svého vedoucího, abych mu dal update a ten mi doporučil Nikitu Golmgrena, který má bakalářskou práci ve které se věnuje backendu pro vytváření a spravování objednávek. Jelikož endpointy, které potřebuji, spadají pod správu objednávek, tak mi byl ochotný s tém pomoci.

Společně jsme prošli dokumentaci a já mu dal všechny požadavky, které jsem potřeboval. Iterativně jsme byli schopní dostat backend i frontend do stavu, kde lze provést skoro všechny

potřebné požadavky. Nestihl se doladit endpoint pro upravení produktů, kde jsme nebyli schopní přijít, zda je chyba na frontendu nebo backendu. Časem bychom chybu určitě našli, ale bohužel už nám nezbyl čas.

3.3.9 Implementace tmavého režimu

Programátoři dost často pracují dlouho do noci. Já tomu nejsem výjimkou a jelikož jsem strávil hodně času prací spíše po nocích, tak jsem se rozhodl ulevit svým očím a naimplementovat tmavý režim.

Inspiroval jsem se různými tmavými režimy jiných aplikací jako je PHP storm atd. A řídil jsem se barvami material designu.

Na vrchním toolbaru jsem chtěl zachovat oranžovou barvu, která je hlavní vizuálních prvek stránky ve světlém režimu, takže jsem tu barvu použil pro jeho text a ikony. Dále jsem vzal ostatní barvy a za pomocí tabulky material design barev jsem je ztmavil, aby tolik nevyčnívali oproti zbytku.

Výsledek, se kterým jsem přišel je prototyp, který lze poměrně jednoduše upravit díky Vuetify. Nicméně, mi velmi pomohl po nocích udržet pozornost a soustředit se na práci.

Všechny obrázky karet ve tmavém režimu naleznete v mediální příloze.

Kapitola 4
Testování

V této kapitole se budu věnovat testování provedené práce. Zmíním, co to jsou mock data a testy, která jsem provedl. Testů softwaru je spoustu a já se rozhodl vybrat ty, co dělá vývojář sám pomocí debugovacího režimu ve vývojovém prostředí, nebo v mém případě, debugovací konzoli v prohlížeči. A v druhé řadě testy s uživateli, které se zaměřují spíše na správnost řešení požadavků.

4.1 Mockovací data

Při vývoji jakéhokoli systému je velmi užitečné mít pro testování tak zvané mockovací data. Pomocí nich nepotřebujeme žádnou databázi řešit napojení na databázi a můžeme testovat funkcionalitu kódu zcela lokálně. Mockovací data se obvykle ukládají přímo někam do adresáře projektu.

V našem projektu máme data rozdělena na dvě části. Tak zvané mockovací handlery, které simulují komunikaci s databází, lze si u nich nastavit, co mají vracet a jak se mají chovat při různých situacích a samotná mockovací data. Když jsem začal s touto prací, tak bylo celé mockování uděláno poměrně nešikovně. Všechno se nacházelo ve stejném souboru, což bylo velmi nečitelné.

Navrhl jsem proto novou strukturu pro celé mockování. Handlery pro každý modul budou mít svůj soubor, stejně tak jako mockovací data pro každý modul. Postup jak toho docílit jsem popsal přímo v komentářích v kódu.

Já sám jsem pak byl nespokojený s tím jak byla má mockovací data navrženy. Kdykoli bych například přidal novou proměnou do nějakého objektu, tak bych musel projít každý mock toho objektu zvlášť a upravit jej. Proto jsem si navrhl generátor na data, který tuto práci udělá za mě. Má to výhodu, že úpravu objektů je poměrně jednoduchá, ale zase na druhou stranu, budou data každé načtení jiná. To by se dalo spravit více způsoby, buď zaručit, aby náhodný seed byl při každém načtení stejný,bylo by třeba více testování, jelikož nevím zcela jistě, jak vue.js řeší náhodnou generaci, a nebo při prvním načtení s mockem uložit data do vuex storu, to už však nechám budoucím vývojářům, budou-li takovou funkcionalitu potřebovat.

V neposlední řadě bych chtěl zmínit, že momentální mock neumí data aktualizovat, veškeré žádosti typu PUT a POST pouze sledují, zda se provedli a nebo ne. Správně chování PUT a POST lze doimplementovat, ale jednalo by se o netriviální snahu, takže je třeba zvážit, jestli je to opravdu třeba.

4.2 Testy vývojáře

V průběhu projektu jsem používal zejména vestavěnou konzoli ve webovém prohlížeči, kam jsem si vypisoval data v objektech, co používám. To mi umožnilo, podívat se jestli se implementovaná

logika chová podle očekávání a nebo ne. V konzoli se zároveň ukazují chybové hlášky z komunikace s backendem, kde lze vidět zda má žádost o data byla úspěšná a nebo ne.

Nadále jsem používal soubor, kam si backend loguje chybové hlášky. Ten byl užitečný zejména pro zjištění, kde se vyskytuje chyba při komunikaci s backendem.

4.2.1 Testování intergrace s backendem

Jedná se testování vývojáře. Jak jsem již zmiňoval, tak při prvním napojení backend uplně nefungoval. Tady jsem právě využil výše zmíněné postupy, abych přišel na to, kde se vyskytuje chyba. Zprvu jsem se zaměřil na mé požadavky. Zkontroloval jsem správnost jejich názvů a korektnost jejich návratových hodnot. Po usouzení, že jejich názvy mám v pořádku jsem zjišťoval, jestli vůbec proběhly. Pokud mi backend vrátil kód 404, pak jsem věděl, že takový endpoint na backendu neexistuje. Ostatní kódy mohly znamenat více věcí, takže bylo potřeba zkontrolovat data, co backend posílá a data co frontend očekává.

Když nastala situace, že data vzájemně neseděli, pak jsem se obrátil na backend tým, protože frontend byl připraven podle navržené dokumentace. V případě, že backend nebyl schopný data upravit, pak jsem musel upravit implementaci na frontendu. Některé věci byly napsané, tak aby fungovali s backendem, který je teď nasazený s plným vědomím faktu, že se v blízké budoucnosti plánuje na backendu změna. Všechny takové části jsem se snažil pečlivě okomentovat, aby další vývojáři věděli, co bude třeba změnit.

Ve finále testování intergrace s backendem vedlo na částečný úspěch. Povedlo se odhalit spoustu chyb jak v backendu, tak ve frontendu. Většina z nich se povedla opravit, ale již nezbyl čas na opravu úpravy produktů. Na to jsem vytvořil nový úkol pro budoucí BI-SP týmy.

4.3 Testování s uživateli

Zde vypíši proces testování s jednotlivými uživateli. Zmíním jejich zpětnou vazbu a řešení nalezených chyb. Ať už se bude jednat o jejich opravu, a nebo vytvoření nového úkolu pro budoucí BI-SP tým.

4.3.1 Jiří Hunka - vedoucí práce, hlavní Stakeholder

Jako prvního uživatele jsem si vybral svého vedoucího Jiřího Hunku, hlavně protože je v rámci tohoto projektu hlavní stakeholder. Zároveň jsem si vyzkoušel jak ve skutečnosti takové testování probíhá. Tuto schůzi jsem ještě nenahrával, protože to pro mě byl zkušební pokus. Z testování vzešlo poměrně dost zpětné vazby.

Hunka měl problém se stránkou, kde se zobrazuje seznam objednávek, konkrétně s tlačítkem pro úpravu. Jeho vize byla taková, že by mělo jít kliknout na celý řádek. Zároveň by se řádek měl zvýraznit. Pokusil tuto chybu opravit, ale povedlo se mi přidat pouze kliknutí na řádek, nikoli jeho zvýraznění. Vytvořil jsem tak úkol do budoucna pro BI-SP tým.

Další problém s tlačítkem pro vystavení faktury. To mělo vždy nápis vystavit fakturu, i když faktura ani neexistovala. Upravil jsem jej tak, aby nápis tlačítka správně odpovídal existenci faktury. Dále měl problém s tisknutím faktury, kde se automaticky nezobrazovalo klasické okno na tisknutí ve webovém prohlížeči, to jsem taktéž opravil.

Nadále, se mu nelíbilo, že při změně stavu v kartě historie objednávky lze při odeslání stále klikat na formulář a nebylo mi jasné, zda se něco děje. Upravil jsem formulář tak, aby při odeslání nového stavu zešedl a nešlo na něj klikat dokud se stav nedostane do systému.

Pak odhalil, že není jasně vidět, jaký je způsob doručení a platby. Navrhl, že by to mělo být zobrazeno přímo v kartě Údaje pro doručení. Konkrétní řešení zatím není, vytvořil jsem pro to úkol pro budoucí BI-SP tým. V kontextu s adresami, také přišel na to, že momentální frontend, řeší kontrolu, zda je vůbec možné upravovat objednávku velmi reaktivně, tj. čeká až mu backend

vrátí žádost, která nešla provést. Chtěl by, aby to bylo více aktivní a administrátor vůbec neměl možnost upravovat, když se objednávky vyskytne v takovém stavu. Taktéž jsem na to vytvořil úkol, který bude nejspíš potřebovat spolupráci s backendem.

Ačkoli testování odhalilo poměrně nemalé množství nedostatků, tak by se dalo říct, že právě díky tomu bylo úspěšné.

4.3.2 Libor Kudrna

Oproti testování s vedoucím, jsem si připravil jasně dané testy, které jsem chtěl postupně projít. Jako první jsem vybral změnu údajů u adresy pro doručení. Tady se uživateli zdálo, že změna jména je intuitivní, avšak měl problém s tím, že nebylo zcela jasné, co se děje na pozadí. Navrhli jsme spolu například notifikace na spodní levé straně okna, podobně jako fungují notifikace systému windows.

Dále měl problém s tím, že vyplnění adresy není chytré. Představoval by si, že napíše začátek adresy a to mu dá možnost vybrat si konkrétní adresu, přičemž její výběr předvyplní poštovní směrovací číslo a město. V rámci testu změny adresy se ptal na limitace ohledně zobrazených dat, konkrétně na zemi, která momentálně zobrazuje pouze id země. Tato limitace vznikla z mého omylu, kde jsem si myslel, že na backendu není endpoint pro získání seznamu zemí, avšak po testování jsem tento problém prošetřil a našel daný endpoint. Vytvořil jsem tedy úkol, ve kterém se tato chyba opraví.

Hned v dalším testu uživatel narazil na problém s objednávkami, chyběl mu tam filter na vyhledávání. Filtry nebyly součástí zadání, nicméně jedná se o velmi velkou změnu kvality užívání, na které již pracuje BI-SP2 tým. Zpět k zadání testu, chtěl jsem po uživateli, aby změnil stav objednávky. Tam mu nebylo hned jasné, v jakém stavu se objednávka vyskytuje a zároveň mu vadilo, že stavy v automatickém doplňování nejsou seřazeny logicky. První problém je otázka na návrh designu, ke které je potřeba další diskuse s Hunkou a Dvořákem, proto jsem založil úkol, ve kterém se schůzka uskuteční. Druhý problém částečně vyřeší endpoint z backendu, který momentálně nefunguje na 100 %, jakmile se opraví.

Při druhém testu se také velmi věnoval jednomu z hlavních požadavků na přesouvání jednotlivých karet. Vadilo mu, že v tom neměl dostatek volnosti. Toto je chyba vývoje, kde jsem nastavil, nějaké minimální velikosti pro každou kartu a nebral jsem přitom v potaz velikost okna. Bude tedy potřeba tuto logiku předělat, v rámci úkolu, který jsem vytvořil, bude třeba i předělat zvětšování a zmenšování karet, které teď sice funguje, ale potřebuje doladit. Další problém našel s tlačítkem poslat v kartě Historie objednávek, kde mu přišlo, že "poslat"je zavádějící a navrhl "uložit"jako řešení. V neposlední řadě velmi ocenil tmavý režim a vzhled jednotlivých komponent.

Stejně jako u prvního testu by se mu líbilo, kdyby byl více upozorněn na změnu, co provedl. Ideální by bylo stejné řešení s notifikacemi, které jsem již zmínil, proto přichází v úvahu zamyslet se nad novou komponentou, která by se dala použít na více místech v celém projektu. Na závěr tohoto testu zmínil, že by se mu líbilo někde na stránce mít tlačítko "Zpět na objednávky".

Ve třetím testu jsem chtěl otestovat vystavení a zobrazení faktury. Tady měl problém, že po vystavení nebylo zcela zřejmé, zda je faktura již vystavena. Navrhl rozdělit tlačítko na 2 části, kde jedna fakturu pouze vystaví a druhá i rovnou zobrazí. Také navrhl, že by tlačítko mělo změnit barvu na zelenou, aby indikovalo úspěšné vystavení faktury. Vytvořil jsem pro to úkol.

Při otevření faktury zmínil, že by se mu líbilo mít tlačítko pro zavření v levém horním rohu a tlačítko pro tisknutí nechat dole. Také jsem pro to vytvořil nový úkol.

V posledním testu jsem chtěl otestovat základní filtraci na stránce s objednávkami, tento test prošel bez nějakých problémů.

Po dokončení testů jsem dal prostor na připomínky a dotazy. Se stránkou byl vcelku spokojen, ale přesto měl pár návrhů a výtek. Líbilo by se mu, kdyby při napsání IČ byly vyplněny údaje s IČem spojené automaticky. Zároveň by chtěl, aby tam byla kontrola na DIČ.

Měl taky obavy ohledně implementace produktů v objednávce. Kde zmínil, že model může mít také dopad na ostatní údaje, a tak by měl být automaticky doplňován. Tento bod je třeba

přednést před Hunku a domluvit se s ním jak dál pokračovat.

Pak se už jen ptal na změny způsobu doručení. Vysvětlil jsem mu jaký je pro ně plán, a kdo na tom momentálně pracuje. Následně se vrátil k adresám a zmínil, že některé doručovací služby neberou číslo jiné národnosti než národnosti uvedené v doručovací adrese. Shrnul bych všechny připomínky na adresu pod jeden úkol, kde bude potřeba si sednout s Hunkou a řádně probrat všechny limitace na každé pole. Vytvořil jsem pro to úkol.

V závěru bych řekl, že tento test byl velmi úspěšný hlavně díky zpětné vazbě, která tento projekt posune dál a také díky spokojenosti uživatele zároveň chci zmínit, že celé testování je nahrané a video můžete nalézt v mediální příloze.

4.3.3 Analýza výsledků z testování

Zde vypíši tabulku s úkoly, které vznikli z testování. Zhodnotím v nich jejich závažnost, obtížnost a zda jsem je do odevzdání stihl opravit.

úkol	závažnost	obtížnost	hotovo
Přechod na úpravu objednávek pomocí celého řádku	nízká	vysoká	ne
Jiný text tlačítka pro zobrazení faktury	střední	nízká	ano
Tisk faktury není automatický	nízká	nízká	ano
Zašednutí formuláře pro změnu stavu při uložení stavu	nízká	nízká	ano
Aktivní kontrola dat na frontendu	vysoká	vysoká	ne
Zobrazení způsobu platby a doručení u adres	nízká	střední	ne
Notifikace při ne/úspěchu změny dat	střední	vysoká	ne
Chytré doplňování adres	střední	vysoká	ne
Zobrazení názvu země namísto jejího id	nízká	nízká	ne
Filtry na tabulku s objednávky	střední	střední	ne
Schůze o nejasnosti v návrhu karty historie objednávky	nízká	nízká	ne
Logické seřazení stavů v automatickém doplnění formuláře	nízká	střední	ne
Doladění přesouvání a změn velikostí karet	vysoká	vysoká	ne
Změna názvu tlačítka poslat v kartě historie objednávek	nízká	nízká	ano
Přidat tlačítko "zpět na objednávky"	nízká	nízká	ne
Přesunout tlačítko "X"u faktury do horního pravého rohu	nízká	nízká	ne
Předělat tlačítko pro vystavění faktury	střední	střední	ne
Limitace na data v adresách	vysoká	střední	ne

Kapitola 5
Závěr

V této práci jsme úspěšně analyzoval návrh uživatelského rozhraní Dvořáka pro úpravu objednávek, diskutoval jeho možnou implementaci, implementoval jej ne základě moderních technologií a frameworků, a následně jo integroval s funkčními částmi nově vznikajícího backendu.

Také jsem se zaměřil na jiné části projektu jako jsou mocky, kde jsem navrhl novou strukturu jak je psát a využívat. Ne malou součástí implementace byla také aktualizace kódu na moderní programovací jazyky a frameworky, což zajistilo možnost dalšího rozšíření a vylepšení systému.

V rámci testování byl systém podroben testům s uživateli, aby bylo ověřeno, zda je systém snadno použitelný a splňuje potřeby administrátorů elektronických obchodů. Ačkoli uživatelé měli připomínky, tak testy byly vcelku úspěšné a řekli, že systém má potenciál být velmi intuitivní a jednoduchý na použití při zakomponování jimi navržených změn. Všechny navržené změny jsem sepsal jako úkoly pro další studenty BI-SP a jiné, které by toto téma zajímalo.

V budoucnu je možné tento systém dále vylepšovat a rozšiřovat, aby se lépe přizpůsobil administrátorům. Zároveň je také potřeba zaměřit se na připomínky z testování, které se nestihly opravit v rámci této práce.

Celkově lze tedy říci, že cíle této práce byly úspěšně splněny a výsledkem je velmi pevný základ moderního a efektivního nástroje pro správu objednávek v elektronických obchodech. Systém se snaží umožnit administrátorům elektronických obchodů snadno spravovat své objednávky a současně se zaměřovat na růst svého podnikání.

Bibliografie

1. DVOŘÁK, Martin. *Frontend administrace e-shopu*. 2022. Dipl. pr. České vysoké učení technické v Praze, Fakulta Informačních technologií. vedoucí Ing. Jiří Hunka.
2. BABÁK, Jan. *Frontend administrace e-shopu*. 2022. Dipl. pr. České vysoké učení technické v Praze, Fakulta Informačních technologií. vedoucí Ing. Libor Kudrna.
3. POINT, TUTORIALS. *SDLC - Overview* [online]. 2023. Dostupné také z: `https://www.tutorialspoint.com/sdlc/sdlc_overview.htm`.
4. POINT, TUTORIALS. *Waterfall model* [online]. 2023. Dostupné také z: `https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm`.
5. POINT, TUTORIALS. *Iterative model* [online]. 2023. Dostupné také z: `https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm`.
6. POINT, TUTORIALS. *Agile model* [online]. 2023. Dostupné také z: `https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm`.
7. SCRUMGUIDES.ORG. *Scrum Guides* [online]. 2023. Dostupné také z: `https://scrumguides.org/`.
8. FANGUY, Will. *A comprehensive guide to design systems* [online]. 2019. Dostupné také z: `https://www.invisionapp.com/inside-design/guide-to-design-systems/`.
9. VUETIFY. *A Material Design Framework for Vue.js* [online]. 2023. Dostupné také z: `https://vuetifyjs.com/en/`.
10. GOOGLE. *Material Design* [online]. 2023. Dostupné také z: `https://m3.material.io/`.
11. NIKOLAY, GEKHT. *Create Better Backlog and Engage the Development Team with FURPS.* [Online]. 2020. Dostupné také z: `https://gehtsoftusa.com/blog/create-better-backlog-and-engage-the-development-team-with-furps/`.
12. DISTRICT, MaRS Discovery. *Product development: Minimum viable product (MVP) approach* [online]. 2023. Dostupné také z: `https://learn.marsdd.com/article/product-development-minimum-viable-product-mvp-approach/`.
13. MICROSOFT. *JavaScript With Syntax For Types* [online]. 2023. Dostupné také z: `https://www.typescriptlang.org/`.
14. SAS, Symfony. *Symfony* [online]. 2023. Dostupné také z: `https://symfony.com/doc/current/index.html`.
15. YOU, Evan. *The Progressive JavaScript Framework* [online]. 2023. Dostupné také z: `https://vuejs.org/`.
16. YOU, Evan. *Vite, Next Generation Frontend Tooling* [online]. 2023. Dostupné také z: `https://vitejs.dev/`.

17. KAWAGUCHI, KAZUYA. *Vue I18n Internationalization plugin for Vue.js* [online]. 2023. Dostupné také z: `https://vue-i18n.intlify.dev/`.
18. YOU Evan; MOROTE, Eduardo San Martin. *The official router for Vue.js* [online]. 2023. Dostupné také z: `https://router.vuejs.org/`.
19. YOU, Evan. *What is Vuex?* [Online]. 2023. Dostupné také z: `https://vuex.vuejs.org/`.
20. FOUNDATION, OPENJS. *Pluggable JavaScript linter* [online]. 2023. Dostupné také z: `https://eslint.org/`.
21. SOFTWARE, FUNCTIONAL. *Application Monitoring and Error Tracking Software* [online]. 2023. Dostupné také z: `https://sentry.io/welcome/`.
22. GITLAB. *GitLab CI/CD* [online]. 2023. Dostupné také z: `https://docs.gitlab.com/ee/ci/`.
23. GITLAB. *GitLab* [online]. 2023. Dostupné také z: `https://docs.gitlab.com/ee/`.
24. DAN RADIGAN, Atlassian. *Why code reviews matter (and actually save time!)* [Online]. 2023. Dostupné také z: `https://www.atlassian.com/agile/software-development/code-reviews`.
25. SALAJ, Martin. *Added view for updating orders* [online]. 2023. Dostupné také z: `https://gitlab.jagu.cz/eshops/administrace-fe/-/merge_requests/41`.

Obsah přiloženého média

- `recordings` ..... adresář s videi
- `images` ..... adresář s obrázky
- `source` ..... adresář se zdrojovým provedených změn
- `text` ..... text práce
  - `source` ..... adresář zdrojového kódu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF