F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Tvorba e-shopu pro obchod outdoorového oblečení

Jan Kleveta
Softwarové inženýrství a technologie

Květen 2025
Vedoucí práce: Ing. Daniel Groschup

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kleveta** Jméno: **Jan** Osobní číslo: **503204**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Enterprise systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Tvorba e-shopu pro obchod outdorového oblečení

Název bakalářské práce anglicky:

Creation of an e-shop for an outdoor clothing store

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Daniel Groschup katedra počítačů FEL

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

Datum zadání bakalářské práce: **14.02.2025** Termín odevzdání bakalářské práce: **23.05.2025**

Platnost zadání bakalářské práce: **20.09.2026**

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis proděkana(ky) z pověření 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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kleveta** Jméno: **Jan** Osobní číslo: **503204**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Enterprise systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Tvorba e-shopu pro obchod outdorového oblečení

Název bakalářské práce anglicky:

Creation of an e-shop for an outdoor clothing store

Pokyny pro vypracování:

1) Zmapujte problematiku online prodeje outdoorového vybavení a
proveďte sběr požadavků pro e-shop.
2) Prozkoumejte možné využití existujících e-commerce řešení a
porovnejte je s implementací vlastního řešení. Zaměřte se na finanční stránku,
udržitelnost a rozšiřitelnost. Na základě průzkumu definujte klíčové
požadavky, které musí eshop splňovat. Minimálně by se mělo jednat o:
- Procházení produktů
- Správu/administraci produktů
- Správa a vytváření objednávek
3) Navrhněte řešení, které odpovídá definovaným požadavkům a definujte MVP (minimal viable product),
které je jádrem takového systému.
4) Implementuje navržené MVP.
5) Proveďte uživatelské testování výsledného systému.

Seznam doporučené literatury:

Fowler M.: Patterns of Enterprise Application Architecture, 2002.
Walls C.: Spring Boot in Action, 2015.
Miroslav Hučka a kol.: Modely podnikových procesů, 2017.
Hana Klčová, Petr Sodomka: Informační systémy v podnikové praxi, 2010.

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

PROHLÁŠENÍ

Já, níže podepsaný

Příjmení, jméno studenta:	Kleveta Jan
Osobní číslo:	503204
Název programu:	Softwarové inženýrství a technologie

prohlašuji, že jsem bakalářskou práci s názvem

Tvorba e-shopu pro obchod outdorového oblečení

vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu.

Prohlašuji, že jsem v průběhu příprav a psaní závěrečné práce použil nástroje umělé inteligence. Vygenerovaný obsah jsem ověřil. Stvrzuji, že jsem si vědom, že za obsah závěrečné práce plně zodpovídám.

V Praze dne 23.05.2025

Jan Kleveta
.....
podpis studenta

Poděkování / Prohlášení

Rád bych poděkoval rodině, všem blízkým a kamarádům, kteří mě během studia podporovali a pomáhali mi.

Dále bych chtěl vyjádřit poděkování všem vyučujícím ČVUT FEL, kteří mi během studia s lidským přístupem předali mnoho cenných znalostí. Především děkuji vedoucímu této bakalářské práce, Ing. Danielu Groschupovi, za pomoc s její realizací.

Prohlášení o samostatném vypracování a o použití nástroje umělé inteligence potvrzené a vygenerované v systému KOS je v této práci v souladu s pravidly vloženo na samostatném listu.

Abstrakt / Abstract

Tato bakalářská práce se zabývá návrhem a implementací e-shopu specializovaného na prodej outdoorového oblečení. Práce se zaměřuje na analýzu trhu s outdoorovým vybavením, sběr požadavků na systém a porovnání dostupných e-commerce řešení se zaměřením na jejich finanční náročnost, rozšiřitelnost a udržitelnost. Na základě analýzy byla zvolena vlastní implementace jako nejvhodnější varianta. Bylo navrženo jádro systému, které zahrnuje využívání uživatelských účtů, správu produktů a tvorbu a správu objednávek. Implementace využívá mikroservisní architekturu, technologie Java, Spring Boot, PostgreSQL a Next.js. Funkčnost systému byla ověřena uživatelským testováním.

Klíčová slova: e-shop, outdoor, Java, Spring Boot, REST API, PostgreSQL, Next.js, API gateway, mikroslužba

This bachelor's thesis focuses on the design and implementation of an e-shop specialized in selling outdoor clothing. The work centers on analyzing the outdoor equipment market, gathering system requirements, and comparing available e-commerce solutions with an emphasis on their financial demands, scalability, and sustainability. Based on the analysis, a custom implementation was selected as the most suitable option. The core of the system was designed to include user account management, product administration, and order creation and management. The implementation uses a microservice architecture and technologies such as Java, Spring Boot, PostgreSQL, and Next.js. The functionality of the system was verified through user testing.

Keywords: e-shop, outdoor, Java, Spring Boot, REST API, PostgreSQL, Next.js, API gateway, microservice

Title translation: Creation of an e-shop for an outdoor clothing store

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

1.1 Předmluva a motivace

V posledních letech jsme svědky postupné změny spotřebitelského chování, která se projevuje přesunem obchodních aktivit do online prostředí. Tento vývoj výrazně akcelerovala pandemie COVID-19, jež do světa e-commerce přivedla i nové věkové skupiny. Tento trend je patrný i v oblasti prodeje outdoorového vybavení, kde zákazníci stále častěji upřednostňují pohodlí a flexibilitu nakupování prostřednictvím internetových obchodů před návštěvou kamenných prodejen, což vyžaduje přizpůsobení obchodní strategie prodejců. Prostředí e-commerce je však vysoce konkurenční a vytváří nemalé nároky na kvalitu prodejního kanálu, přičemž možností realizace je mnoho a zmapování problematiky je klíčové pro budoucí efektivní provoz e-shopu.

1.2 Cíle

Cílem této práce je zmapovat problematiku online prodeje outdoorového vybavení a provést sběr požadavků pro optimální e-shop. Analyzovat možné využití existujících řešení a porovnat je s vlastní implementací. Následně vytvořit návrh a implementaci jádra e-shopu pro prodej outdoorového oblečení, který bude mít formu MVP a měl by minimálně umožnit procházení, správu a administraci produktů, dále správu a vytváření objednávek.

1.2.1 Struktura práce

Kapitola 2 představuje úvod do problematiky online prodeje outdoorového vybavení, zaměřuje se na finanční stav trhu, vliv způsobu online prezentace na chování zákazníka, postavení výrobců na trhu a vznik nových tržních segmentů otvírajících obchodní příležitosti.

V kapitole 3 definuji cíle projektu a provedu sběr požadavků, které rozdělím na **primární** a volitelné. Smyslem tohoto rozdělení je identifikovat požadavky, jejichž naplněním vznikne jádro e-shopu pro prodej outdoorového oblečení.

Následně provedu v kapitole 4 průzkum možností realizace e-shopu se zaměřením na finanční stránku jednotlivých řešení, jejich dlouhodobou udržitelnost a rozšiřitelnost. V závěru kapitoly provedu výběr řešení, kde porovnám využití existujících řešení s možností vlastní implementace.

Na základě cílů a **primárních** požadavků definovaných v kapitole 3, provedu v kapitole 5 návrh takovéhoto řešení, přičemž budu klást důraz na budoucí rozšiřitelnost funkcionalit a výkonnostní škálovatelnost systému.

Pro realizaci návrhu vyberu v kapitole 6 vhodné technologie, pomocí kterých jádro e-shopu pro prodej outdoorového oblečení v kapitole 7 implementuji. Výsledné řešení v kapitole 8 podrobím uživatelskému testování.

Kapitola 2
Online prodej outdoorového vybavení

V této kapitole se zaměřím především na finanční situaci trhu s outdoorovým vybavením ve spojení s probíhajícím přelivem zákazníků z kamenných prodejen do online prostoru, dále na zasazení e-shopu do online prostoru, outdoorové trendy a postavení výrobců.

2.1 Finanční stav trhu

Evropský trh s outdoorovým vybavením vykazuje značný růst, dle zveřejněných dat v projektu State of Trade společnosti European Outdoor Group [1] v roce 2022 přesáhl hodnotu 6 miliard EUR, přižemž vykázal meziroční nárůst ve výši 6,31 %.

Značný podíl tohoto trhu představuje online prodej, dle průzkumu Outdoor Consumer Report [2] společností Deloitte a ISPO, 34 % zákazníků preferuje nákup outdoorového vybavení online, namísto kamenné prodejny. Pro stejný průzkum zmiňuje marketingový ředitel Terina Group, Imanol Munoz, že pandemie COVID-19 přivedla k online nakupování i věkové skupiny, které ho do té doby nevyužívaly.

2.2 Forma online prezentace

Samotná volba online prostoru jako prodejního kanálu však stačit nemusí, důležitý je i způsob provedení.

Catherine Lambert [3] v případové studii zaměřující se na rebranding společnosti Sleepyhead na DockATot popisuje, jakým způsobem ovlivnila komunikace rebrandingu na hlavní stránce e-shopu chování zákazníků. Rebrandovaná firma se potýkala s poklesem tržeb v důsledku zmatení zákazníků rebrandingem, najala tedy marketingový a vývojový tým Swanky's Growth Accelerator, který souběžně s existující domovskou stránkou spustil i novou stránku, která lépe komunikovala rebranding a přesměroval na ni část zákatníků. Stránky s novou domovskou stránkou vykazovaly 71,44 % nárůst tržeb na uživatele, za čímž stál 32,92 % nárůst počtu uživatelů, kteří si přidali produkt do košíku a 42,19 % nárůst počtu transakcí.

Mimo provedení samotných stránek je podstatné i komplexní zasazení e-shopu do online prostoru, jako využívání sociálních sítí, analytických nástrojů, placené propagace či kvalitní optimalizaci pro vyhledávače (SEO).

2.3 Nové segmenty

Díky technologickému pokroku a trendům vznikají i nová odvětví trhu s outdoorovým vybavením. V posledních letech se těší narůstající oblibě v outdoorové komunitě odlehčování vybavení, což definuje poměrně nový segment trhu nazývaný ultralight, zaměřující se na maximální odlehčení výbavy potřebné na trek.

Dle průzkumu The Pacific Crest Trail Gear Guide (2023 Survey) [4], nesl v roce 2023 na zádech každý hiker zdolávající americkou turistickou stezku Pacific Crest Trail průměrně 3,594 kg.

Takto odlehčená výbava stále musí být odolná a funkční, tudíž je kladen důraz na nejmodernější materiály a kvalitní provedení, cena takovéhoto vybavení se pohybuje řádově i ve vyšších desítkách tisíc korun. [4]

2.4 Postavení výrobců

Vedle mainstreamových výrobců, jako například Rab, Sea to Summit či Patagonia, mů- žeme pozorovat vysokou poptávku po produktech menších výrobců, takzvaných „cottage brands“, z USA například Durston či Senchi Designs, u jejichž produktů běžně poptávka převyšuje nabídku, mnohdy i u do EU nákladně dovezených a proclených produktů, u kterých dle vyjádření lokálních prodejců i majitele značky Durston, Dana Durstona [5], často není prostor pro uspokojivou marži lokálních prodejců, což vytváří prostor na trhu pro nové lokální výrobce.

U úspěšných českých prémiových výrobců Tilak, Patizon či Filip Raboch je kladen důraz na prezentaci faktu, že jsou jejich produkty vyrobeny v České Republice, podobně jsou na tom další evropští výrobci.

Kapitola 3
Definice cílů a sběr požadavků

Na základě průzkumu existujících internetových obchodů zabývajících se prodejem prémiového outdoorového oblečení, zejména obchodů provozovaných přímo výrobci takovéhoto oblečení, jsem definoval cíle a shromáždil požadavky, které představují soubor funkcionalit obchodu zaměřujícího se na tento segment trhu.

3.1 Cíle projektu

Během průzkumu jsem identifikoval následující primární cíle jako optimální pro fungování e-shopu nabízejícího ultralehké outdoorové oblečení. Později definuji jádro tohoto systému, které implementuji.

- Umožnit zákazníkům efektivní nákup outdoorového oblečení na internetu.
- Vytvořit online kanál pro prezentaci výrobce.
- Umožnit budoucí funkční rozšiřitelnost a výkonnostní škálovatelnost vznikajícího systému.

3.2 Požadavky systému

V požadavcích systému definuji dílčí požadavky, které společně slouží k naplňení cílů projektu. Požadavky jsou rozděleny na volitelné a **primární**, které slouží k poskytnutí základních funkcionalit internetového obchodu, jako jsou:

- správa a procházení produktů,
- vytváření, platba a správa objednávek,
- uživatelské účty.

3.2.1 Role

Role definují jednotlivé skupiny uživatelů, které budou se sytémem interagovat následovně:

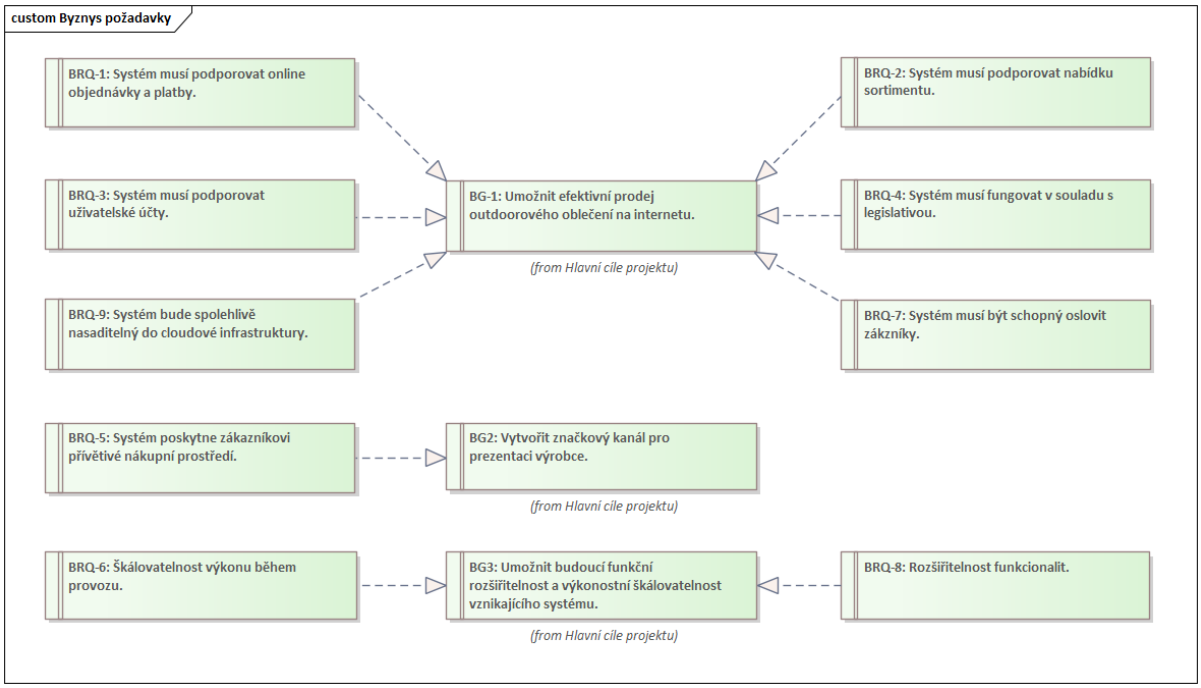
- **Administrátor** spravuje e-shop, což znamená jak správu produktů, obsahu, objednávek a recenzí, tak přístup k analytickým datům o chování uživatelů a stavu systému.
- **Neregistrovaný zákazník** může prohlížet produkty, obsah webu a vytvářet objednávky. Dále může vytvořit uživatelský účet a přihlásit se do něj, čímž se z něj stane registrovaný zákazník.
- **Registrovaný zákazník** může po přihlášení navíc oproti neregistrovanému zákazníkovi prohlížet své objednávky přímo v uživatelském účtu, psát recenze na produkty, které zakoupil, a může ukládat své doručovací a fakturační údaje.

3.3 Byznys požadavky

Pro naplnění byznys cílů definovaných v kapitole 3.1 definuji následující byznys požadavky:

- **BRQ-1: Systém musí podporovat online objednávky a platby.**
- **BRQ-2: Systém musí podporovat nabídku sortimentu.**
- **BRQ-3: Systém musí podporovat uživatelské účty.**
- **BRQ-4: Systém musí fungovat v souladu s legislativou.**
- **BRQ-5: Systém poskytne zákazníkovi přívětivé nákupní prostředí.**
- **BRQ-6: Systém musí umožnit během provozu škálování výpočetního výkonu infrastruktury, na které je nasazen, v závislosti na jeho vytížení.**
- **BRQ-7: Systém musí být schopný efektivně oslovit zákazníky.**
- **BRQ-8: Systém musí v budoucnu umožnit přidávání nových funkcionalit bez nutnosti větších zásahů do jeho již existujících částí.**
- **BRQ-9: Systém bude spolehlivě nasaditelný do cloudové infrastruktury.**

Diagram na obrázku 3.1 zobrazuje, jakým způsobem byznys požadavky naplňují byznys cíle definované v kapitole 3.1. U vybraných cílů je pro přehlednost uváděn pouze stručný název, jejich plné znění je k dispozici výše.



Obrázek 3.1. Diagram byznys cílů a jejich plnění pomocí byznys požadavků.

3.3.1 Funkční požadavky

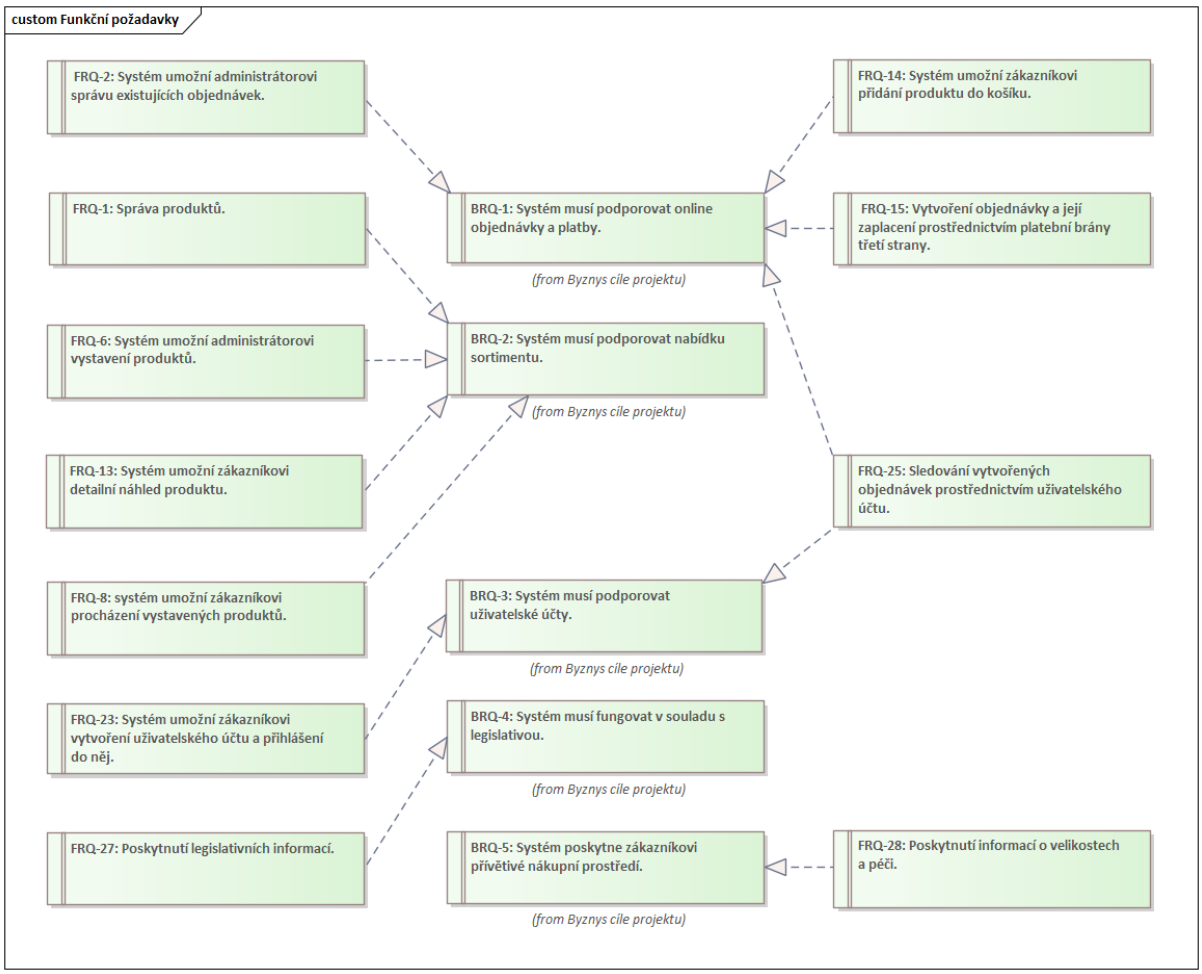
Jednotlivé funkční požadavky slouží jako výčet funkcionalit, které má plně implementovaný systém poskytovat.

- **FRQ-1: Systém umožní administrátorovi spravovat vystavené produkty a jejich skladové zásoby.**

- **FRQ-2: Systém umožní administrátorovi správu existujících objednávek.**
- FRQ-3: Systém umožní administrátorovi reagovat na recenze.
- FRQ-4: Systém umožní administrátorovi rozhodnout o publikaci uživatelské recenze.
- FRQ-5: Systém umožní administrátorovi sledovat metriky e-shopu, jako návštěvnost a chování zákazníka.
- **FRQ-6: Systém umožní administrátorovi vystavení produktů.**
- FRQ-7: Systém umožní administrátorovi rozřazení produktů do kategorií.
- **FRQ-8: systém umožní zákazníkovi procházení vystavených produktů.**
- FRQ-9: systém umožní zákazníkovi specifikaci vlastního produktu pomocí konfigurátoru.
- FRQ-10: Systém umožní zákazníkovi porovnání produktů pomocí srovnávače.
- FRQ-11: Systém umožní zákazníkovi filtrovat produkty na základě jejich specifikace.
- FRQ-12: Systém umožní zákazníkovi fulltextové vyhledávání.
- **FRQ-13: Systém umožní zákazníkovi detailní náhled produktu, včetně popisu, ceny včetně DPH, velikostí, váhy a jejich dostupnosti.**
- **FRQ-14: Systém umožní zákazníkovi přidání produktu do košíku v požadovaném množství a specifikaci.**
- **FRQ-15: Systém umožní zákazníkovi vytvoření objednávky a její zaplacení prostřednictvím platební brány.**
- FRQ-16: Systém umožní zákazníkovi přidání poznámky k objednávce.
- FRQ-17: Systém umožní zákazníkovi sledování stavu objednávky prostřednictvím přímého odkazu.
- FRQ-18: Systém umožní zákazníkovi vytvořit recenzi produktu, který zakoupil.
- FRQ-19: Systém umožní zákazníkovi podat reklamaci pomocí uživatelského rozhraní.
- FRQ-20: Systém umožní zákazníkovi vytvořit žádost o opravu produktu pomocí uživatelského rozhraní.
- FRQ-21: Systém umožní zákazníkovi přihlášení a odhlášení odběru newsletteru.
- FRQ-22: Systém umožní zákazníkovi využít různé lokalizace e-shopu, modifikující jazyk, měnu a možnosti dopravy.
- **FRQ-23: Systém umožní zákazníkovi vytvoření uživatelského účtu a přihlášení se do něj.**
- FRQ-24: Systém umožní zákazníkovi hlídání dostupnosti pomocí funkce „hlídací pes“.
- **FRQ-25: Systém umožní přihlášenému zákazníkovi sledování vytvořených objednávek prostřednictvím uživatelského účtu.**
- FRQ-26: Systém umožní přihlášenému zákazníkovi uložení a správu doručovacích a fakturačních údajů.

- **FRQ-27: Systém umožní poskytnout legislativní informace, jako jsou obchodní podmínky, reklamační řád, záruční podmínky, GDPR, cookies, platební a doručovací možnosti.**
- **FRQ-28: Systém umožní poskytnout doporučení ohledně výběru vhodné velikosti a péči o produkt.**
- FRQ-29: Systém poskytne zákazníkovi odhad výše ceny dopravy a celních poplatků již v košíku.
- FRQ-30: Systém poskytne zákazníkovi informace o dodavatelských řetězcích a jejich uhlíkové stopě.
- FRQ-31: Systém informuje zákazníka o změně stavu objednávky emailem.

Způsob, jakým **primární** funkční požadavky naplňují byznys požadavky zachycuje diagram na obrázku 3.2. U vybraných cílů a požadavků je pro přehlednost uváděn pouze stručný název.



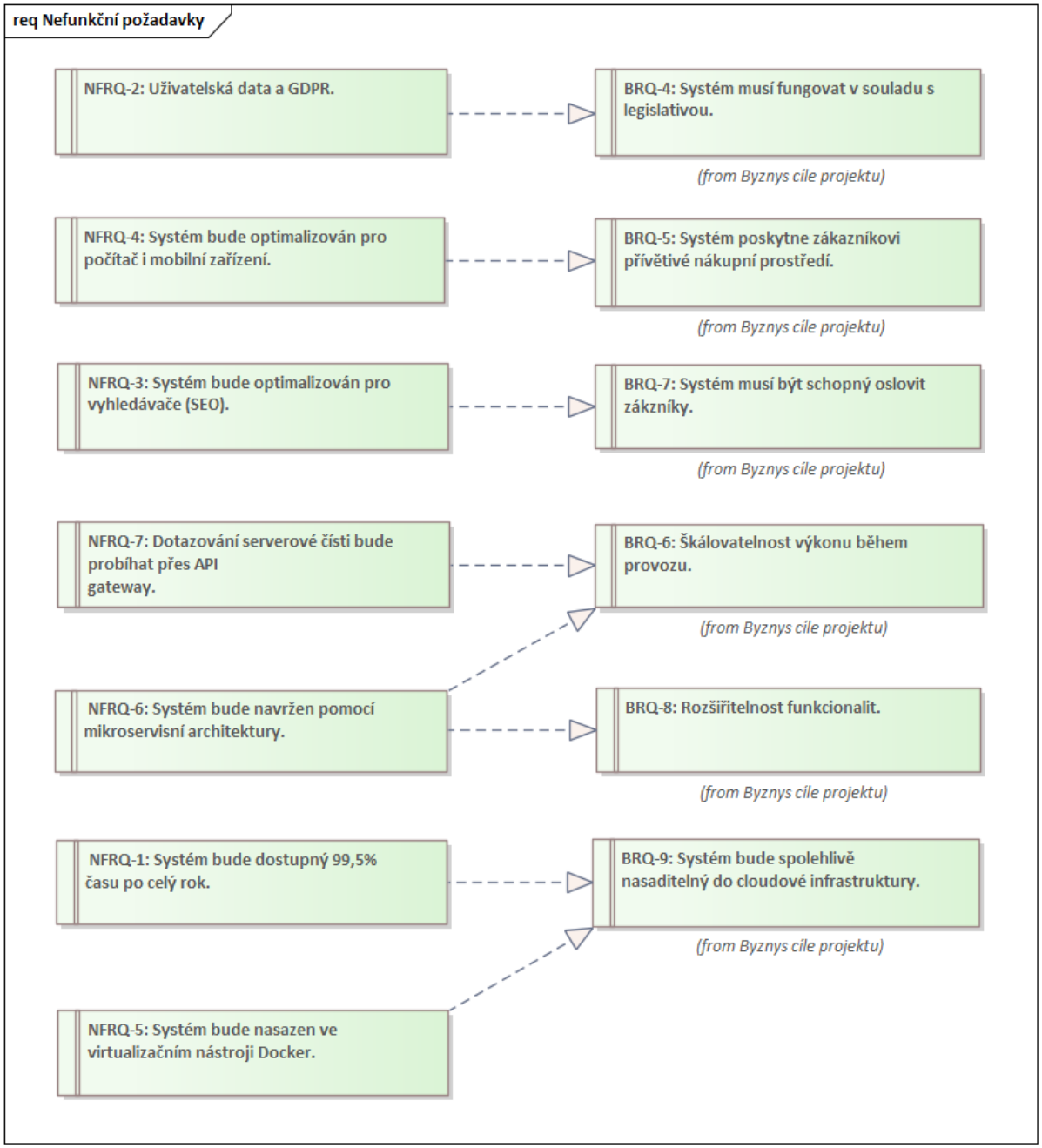
Obrázek 3.2. Diagram zachycující plnění byznys požadavků prostřednictvím funkčních požadavků.

3.3.2 Nefunkční požadavky

Nefunkční požadavky definují kvalitativní kritéria a vlastnosti realizovaného řešení. Požadavky NFRQ-5, NFRQ-6 a NFRQ-7 jsou platné pouze pro případ vlastní implementace.

- NFRQ-1: Systém je dostupný 99,5% času po celý rok.
- **NFRQ-2: Systém zajišťuje ocharnu uživatelských dat před neoprávněným vniknutím a pracuje s nimi dle GDPR.**
- NFRQ-3: Systém bude optimalizován pro vyhledávače (SEO).
- NFRQ-4: Systém bude optimalizován pro počítač i mobilní zařízení.
- **NFRQ-5: Serverová část systému bude nasazena ve virtualizačním nástroji Docker.**
- **NFRQ-6: Systém je navržen pomocí mikroservisní architektury.**
- **NFRQ-7: Systém je navržen tak, že dotazování serverové čísti bude probíhat přes API gateway.**

Způsob, jakým nefunkční požadavky naplňují byznys požadavky zachycuje diagram na obrázku 3.3. U vybraných cílů a požadavků je pro přehlednost uváděn pouze stručný název.



Obrázek 3.3. Diagram zachycující plnění byznys požadavků prostřednictvím nefunkčních požadavků.

Kapitola 4
Výběr řešení

V této kapitole se zabývám možnostmi zřízení internetového obchodu a porovnám využití existujícího řešení s vlastní implementací. Existujících řešení je nespočet, proto jsem se zaměřil na vybrání reprezentativních zástupců, které následně porovnám s vlastní implementací. Do kalkulace zahrnuji plán obsahující služby nezbytné pro naplnění cílů definovaných v kapitole 3.1 a se započítáním případné slevy při závazku na jeden rok, též zohledním případné počáteční náklady na zprovoznění. S ohledem na nepřeberné množství doplňků, které nejsou součástí základních plánů porovnávaných řešení, nebudu přímo započítávat náklady na jejich využití do porovnání, některé balíčky pouze ilustrativně zmíním. Ceny převedu z cizích měn na české koruny (CZK) dle kurzů ČNB [6] ke dni 15.5.2025. Nebudu uvádět náklady, které jsou pro všechna řešení společná, jako je pořízení domény.

4.1 Shopify

Shopify nabízí internetový obchod formou as Service (**SaaS**), tedy kompletní řešení od infrastruktury po implementaci. Dle žebříčku nejoužívanějších e-commerce platforem napříč deseti Evropskými státy od společnosti Tembi [7] se umístil celkově na druhém místě za open-source platformou WooCommerce, v kategorii SaaS se tedy jedná s podílem 32 % o nejpopulárnější platformu ve zkoumaném regionu. Jeho popularita v kombinaci s formou SaaS indikuje vysokou bezpečnost platformy a **nabízí více než 8 000 rozšíření** dávajících Shopify poměrně vysokou modularitu, přinejmenším oproti jeho méně populárním konkurentům. [8] Modularita je samozřemě omezena způsobem, jakým je rozšíření předem implementováno.

Některá rozšíření jsou dostupná zdarma, jiná se pohybují většinou v řádech stovek až tisíců korun měsíčně, přičemž průměrná měsiční cena rozšíření je dle webu Meetanshi [9] 1495 CZK. [10] Nutné je také uvážit jednorázovou platbu za šablonu poskytující uživatelské rozhraní, Shopify sice nabízí zdarma 6 šablon vhodných pro prodej oblečení, které jsou ale s aritmetickým průměrným hodnocením 39 % poměrně nepopulární. O poznání lépe je na tom 6 nejpopulárnějších šablon ve stejné kategorii, které při průměrné popularitě 93 % provozovatele obchodu vyjdou **jednorázově na 7193 - 8992 CZK**. [11]

Pro naplnění cílů definovaných v kapitole 3.1 jsem identifkoval jako optimální plán Shopify **Advanced za 7281 CZK měsíčně**, zejména kvůli službě Shopify Markets, díky které je možné s tímto plánem dokupovat lokalizované trhy nad rámec třech trhů, které jsou ve všech plánech zdarma. [12] Shopify Markets nabízí mimo jiné modifikaci e-shopu na základě lokace zákazníka, od měn, jazykové lokalizace, cen, a sortimentu, po vzhled obchodu či bannery. [13] Navíc díky plánu Advanced získá provozovatel nižší poplatky z plateb přes **platební bránu Shopify Payments**. Použití platebních bran třetích stran sice Shopify umožňuje, ale účtuje si k poplatkům třetích stran ještě další své poplatky, tudíž se tato možnost finančně příliš nevyplatí. [12] Jako značnou nevýhodu považuji **poplatek ve výši 2 % za konverzi měn** ve spojitosti s možností výběru pouze v eurech (EUR), nebo CZK. [14, 15]

Shrnutí:

- Sazby karet vydaných v evropském hospodářském prostoru (EHP): 1,6 % + 6,50 CZK
- Konverzní polatek: 2 %
- 7193 - 8992 CZK %
- 7282 CZK měsíčně

4.2 WooCommerce

Jak jsem již uvedl v kapitole 4.1, WooCommerce je lídrem žebříčku webu Meetanshi [9] nejoužívanějších e-commerce platforem napříč deseti Evropskými státy, stejně dobře si dle webu Yagura [16] vede celosvětově s podílem 38,7 %. WooCommerc je oblíbený zejména proto, že je zdarma. Jedná se totiž o **open-source plugin pro WordPress**. [17] **Rozšíření, kterých je na WooCommerce Marketplace více než tisíc, už jsou ale většinou placené**. Stejně tak více než 100 šablon nabízejících uživatelské rozhraní. [18] Stejně jako u Shopify, je e-shop běžící na platformě WooCommerce modifikovatelný jen do takové míry, jakou rozšíření umožňují.

Velkou výhodou oproti Shopify je **možnost využití platebních bran třetích stran bez dalších polatků**, díky čemuž je možné dosáhnout poplatků ve výši menší než 1 % z plateb uskutečněných pomocí spotřebitelských karet vydaných v EHP. [19]

Jeho popularita je však i jeho slabinou, jelikož WordPress pohání 43,6 % všech webových stránek, je i častým cílem kybernetických útoků. V roce 2023 95,5 % úspěšných kybernetických útoků na systémy pro správu obsahu cílilo právě na WordPress. [20] [21]

Hlavní důvody infekcí stránek byly rozděleny následovně:

- 39.1 % používalo zastaralé verze.
- 13,97 % používalo alespoň jeden zranitelný plugin nebo šablonu.
- 49,21 infikovaných stránek obsahovalo zadní vrátka, jako jsou falešné pluginy nebo vložené administrátorské uživatele, což hackerům umožňovalo znovu získat přístup i po vyčištění.

[22] Jako prevence útoků cílících na zastaralé verze částí systému je potřeba jejich pravidelná aktualizace, což při jejich různorodém původu často může vést k jejich následné nekompatibilitě. [23]

V situaci, kdy je cílem mít propracovaný e-shop, bude potřeba množství pluginů, s čímž roste i bezpečnostní riziko. Takovéto řešení není ideální, pokud chci zajistit bezpečnost uživatelských dat, což je i mou zákonnou povinností. [24]

Dále je pro mne překážkou nulová zkušenost s WooCommerce a WordPressem, což je značná nevýhoda oproti SaaS řešením, či vlastní implementaci pomocí technologií, které mi jsou vlastní.

Tyto nevýhody jsou pro mne dostatečným důvodem k tomu WooCommerce jako možné řešení již nezvažovat, proto se nebudu pouštět ani do cenového odhadu nákladů na provoz infrastruktury, jako je hosting či databáze.

4.3 Shoptet

Mezi zvažovaná řešení jsem zahrnul i další **SaaS** řešení Shoptet proto, že dominuje především českému a slovenskému e-commerce trhu. [23, 25] Zaměření na užší segment trhu oproti Shopify mu umožňuje získat větší podíl této části trhu.

Shoptet nabízí **373 doplňků a 21 šablon**, což je oproti Shopify řádově méně, na druhou stranu **umožňuje integraci platebních bran třetích stran pouze s paušálním poplatkem**. [26, 27] Například jako doplněk je možné přidat do e-shopu platební bránu Comgate, jež nabízí poplatky ve výši menší než 1 % z plateb uskutečněných pomocí spotřebitelských karet vydaných v EHP. [19] Shoptet si za to účtuje měsíční poplatek ve výši 200 CZK bez DPH. [27]

Pro naplnění cílů definovaných v kapitole 3.1 jsem ze standartních plánů identifikoval jako nejoptimálnější plán Enterprise za 3811 CZK bez DPH měsíčně, obsahující mnoho podstatných doplňků jako Pokročilé SEO, Cizí jazyky, Fotorecenze, Cizí měny a další. [28]

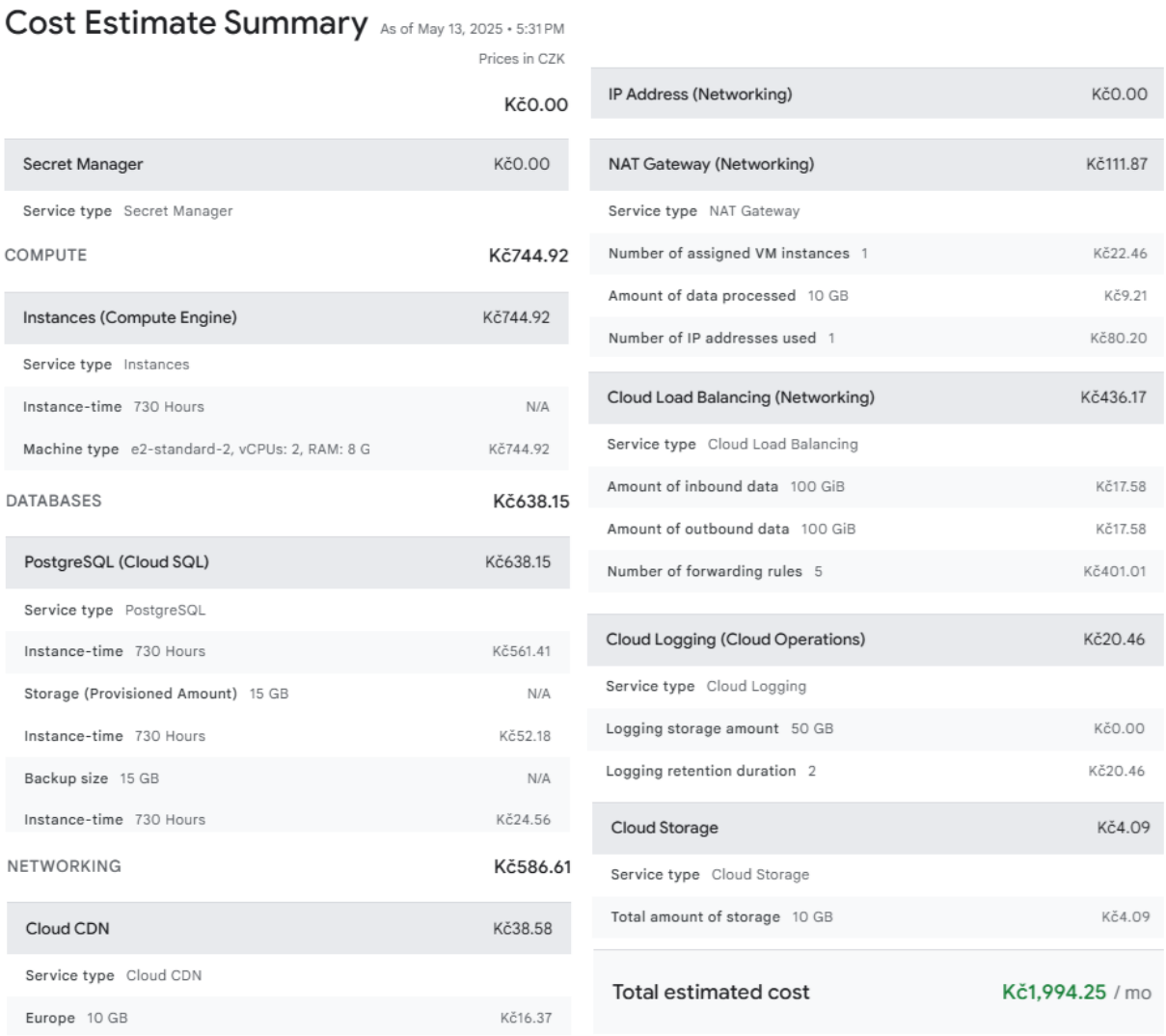
Přesto ale tento plán nenabízí možnost rozšiřování na úrovní Shopify, tomu by mohl konkurovat spíše plán Premium, který nabízí mimo jiné přístup k aplikačnímu interface (API), upravitelné šablony či analytické nástroje. **Takovéto řešení Shoptet nabízí za cenu začínající od 12 000 CZK měsíčně**, pravděpodobně bez DPH, jak je tomu i u ostatních cen uváděných na stránkách firmy Shoptet. Nicméně pro konkrétní nacenění, včetně poplatků pltební brány, je potřeba individuální nabídka. [29, 30]

4.3.1 Vlastní řešení

Při použití vlastního řešení implementovaném na základě vhodného návrhu se prakticky nekladou meze budoucímu rozšiřování funkcionalit e-shopu.

Vlastní řešení přináší i maximální míru svobody při volbě platební brány, což znamená **poplatky pro platby pomocí spotřebitelských karet vydaných v EHP nižší než 1 %**, například pomocí platební brány Comgate, jak jsem již zmiňoval například v kapitole 4.2. [19]

Finanční náklady na provoz spočívají především ve výpočetním výkonu a infrastruktuře potřebné k provozu e-shopu, modelový příklad nasazení jsem vytvořil pomocí Google Cloud (GCP) kalkulátoru [31], který poskytuje i orientační odhad měsíčních provozních nákladů. **Výsledná konfigurace včetně ceny 1994 CZK měsíčně je na obrázku 4.1.**



Cost Estimate Summary — As of May 13, 2025 • 5:31 PM

Prices in CZK

Kč0.00

Secret Manager	Kč0.00
Service type Secret Manager	
COMPUTE	Kč744.92
Instances (Compute Engine)	Kč744.92
Service type Instances	
Instance-time 730 Hours	N/A
Machine type e2-standard-2, vCPUs: 2, RAM: 8 G	Kč744.92
DATABASES	Kč638.15
PostgreSQL (Cloud SQL)	Kč638.15
Service type PostgreSQL	
Instance-time 730 Hours	Kč561.41
Storage (Provisioned Amount) 15 GB	N/A
Instance-time 730 Hours	Kč52.18
Backup size 15 GB	N/A
Instance-time 730 Hours	Kč24.56
NETWORKING	Kč586.61
Cloud CDN	Kč38.58
Service type Cloud CDN	
Europe 10 GB	Kč16.37
IP Address (Networking)	Kč0.00
NAT Gateway (Networking)	Kč111.87
Service type NAT Gateway	
Number of assigned VM instances 1	Kč22.46
Amount of data processed 10 GB	Kč9.21
Number of IP addresses used 1	Kč80.20
Cloud Load Balancing (Networking)	Kč436.17
Service type Cloud Load Balancing	
Amount of inbound data 100 GiB	Kč17.58
Amount of outbound data 100 GiB	Kč17.58
Number of forwarding rules 5	Kč401.01
Cloud Logging (Cloud Operations)	Kč20.46
Service type Cloud Logging	
Logging storage amount 50 GB	Kč0.00
Logging retention duration 2	Kč20.46
Cloud Storage	Kč4.09
Service type Cloud Storage	
Total amount of storage 10 GB	Kč4.09
Total estimated cost	Kč1,994.25 / mo

Obrázek 4.1. Odhad ceny nasazení vlastního řešení do Google Cloud. Zdroj: [31]

Přestože se finální cena může lišit v závislosti na vytížení e-shopu a mohlo by být nutné některé části infrastruktury posílit, tak by i při takovémto posílení neměla být cena za provoz násobně vyšší než cena 1994 CZK z provedené konfigurace, což jsem pomocí kalkulátoru ověřil. [31]

4.4 Volba řešení

Předchozí kapitoly potvrzují zřejmý fakt, že vlastní řešení jednoznačně převyšuje SaaS řešení v možnostech rozšiřitelnosti a modularity e-shopu. Dále jsem se rozhodl na základě skutečností uvedených v kapitole 4.2 WooCommerce jako možnost vyloučit.

Budu zde tedy přímo srovnávat finanční náklady na zřízení a provoz internetových obchodů realizovaných pomocí Shopify, Shoptet a vlastního řešení. Nutno podotknout, že do nákladů na provoz řešení Shoptet a Shopify vůbec nezahrnuji případné použití placených rozšíření, ale pouze cenu plánů umožňujících naplnění cílů definovaných v kapitole 3.1.

Jak jsem již uvedl v kapitole 4.3, pro nacenění Shoptet Premium je potřeba individuální nabídka jak pro paušální cenu, tak pro výši poplatků za platby. Proto jako

paušální cenu použiji cenu, kterou Shoptet uvádí jako startovní, ke které přičtu DPH. Podobný přístup volím u platebních poplatků, použiji sazbu ve výši 0,98 %, poskytovanou platební bránou Comgate, stejně jako u vlastního řešení. Konverzní poplatek u Shopify pro výběr měn jiných než EUR a CZK nezohledňuji.

Jelikož se zaměřuji na poměrně specifický segment trhu outdoorového oblečení, ve kterém převyšuje poptávka nabídku, tak odhad počtu prodejů zakládám na odhadované produkci vyšších desítek kusů měsíčně. Pro modelový příklad volím 80 kusů při průměrné ceně 2500 CZK.

uvedeno v CZK	Shopify	Shoptet	vlastní
měsíční paušál	7282	14520	1994
platební poplatky	3720	1960	1960
měsíčně celkem	11002	16480	3954
jednorázově	7193	0	0

Tabulka 4.1. Náklady na realizaci a provoz zvažovaných řešení.

Z tabulky 4.1 je zřejmé, že vlastní řešení je pro mne nejvýhodnější i po finanční stránce. A to i přes to, že jsem volatilní položky zohlednil tak, že hrají ve prospěch SaaS řešení. Tudíž jsem se rozhodl pro realizaci vlastního řešení.

Kapitola 5
Návrh

Jádro systému definuji jako soubor cílů a **primárních** požadavků definovaných v kapitole 3, pro nějž vytvořím analytický návrh, podle kterého systém implementuji.

Definuji případy užití a následně navrhnu základní strukturu systému. Nejdříve pomocí jednotlivých entit a relací navrhnu analytický doménový model, který následně rozdělím do vícero menších modelů. Aplikační logika bude tedy rozdělena mezi vícero komponent, takzvaných mikroslužeb, což minimalizuje provázanost jednotlivých částí systému. To umožňuje efektivněji škálovat výkon na základě vytížení jednotlivých komponent. Za předpokladu, že komponenta zachová existující aplikační rozhraní, můžeme měnit její aplikační logiku, aniž by to ovlivňovalo ostatní komponenty. Při dekomponování systému vycházím ze znalostí nabytých v předmětu „Návrh softwarových systémů“

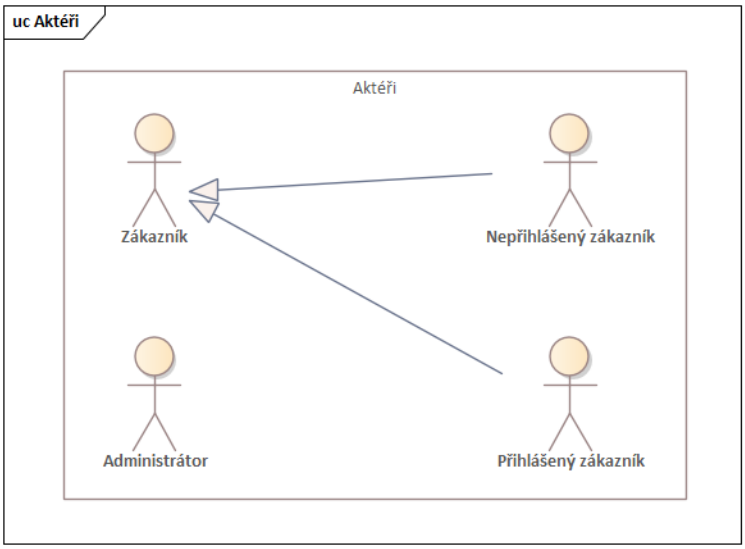
5.1 Případy užití

V této kapitole uvedu aktéry, kteří budou se systémem interagovat a následně pro ně definuji případy užití, neboli use case, které reprezentují funkcionality, jež jim implementované jádro systému umožní.

5.1.1 Aktéři případů užití

Aktéři reprezentují role, jež budou náležet jednotlivým uživatelům a na jejichž základě jim systém zpřístupní dané funkcionality. Veškeré aktéry znázorňujě diagram na obrázku 5.1, jsou jimi:

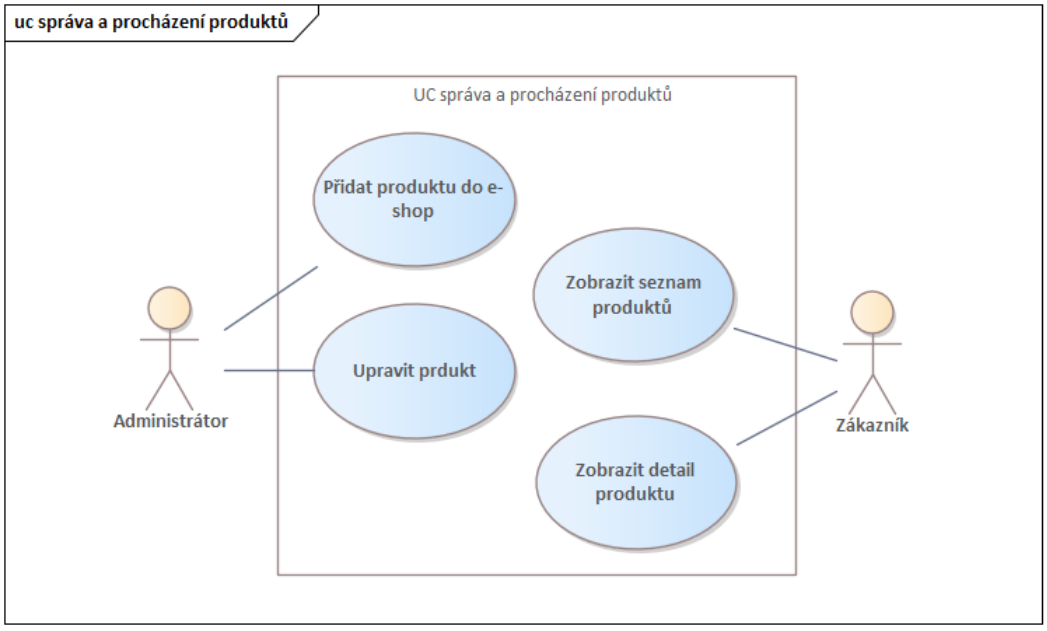
- Administrátor, jež pomocí uživatelského účtu s příslušnou rolí spravuje systém.
- Zákazník, pomocí kterého definuji funkcionality, jež budou umožněny přihlášeným i nepřihlášeným zákazníkům.
- Nepřihlášený zákazník, reprezentující zákazníka, který není přihlášen do systému a rozšiřuje pravomoce aktéra Zákazník o příslušné funkcionality.
- Přihlášený zákazník, reprezentující zákazníka, který má vytvořený zákaznický účet a přihlásil se pomocí něj do systému. Rozšiřuje pravomoce aktéra Zákazník o příslušné funkcionality.



Obrázek 5.1. Katalog aktérů a příslušné dědičnosti.

5.1.2 UC správa a procházení produktů

Diagram na obrázku 5.2 zachycuje funkcionality týkající se správy sortimentu a jeho procházení. Produkty přidává a upravuje Administrátor. Přidané produkty mohou procházet všichni zákznící, tedy aktéři typu Zákazník, případně si mohou zobrazit jejich detailní náhled.

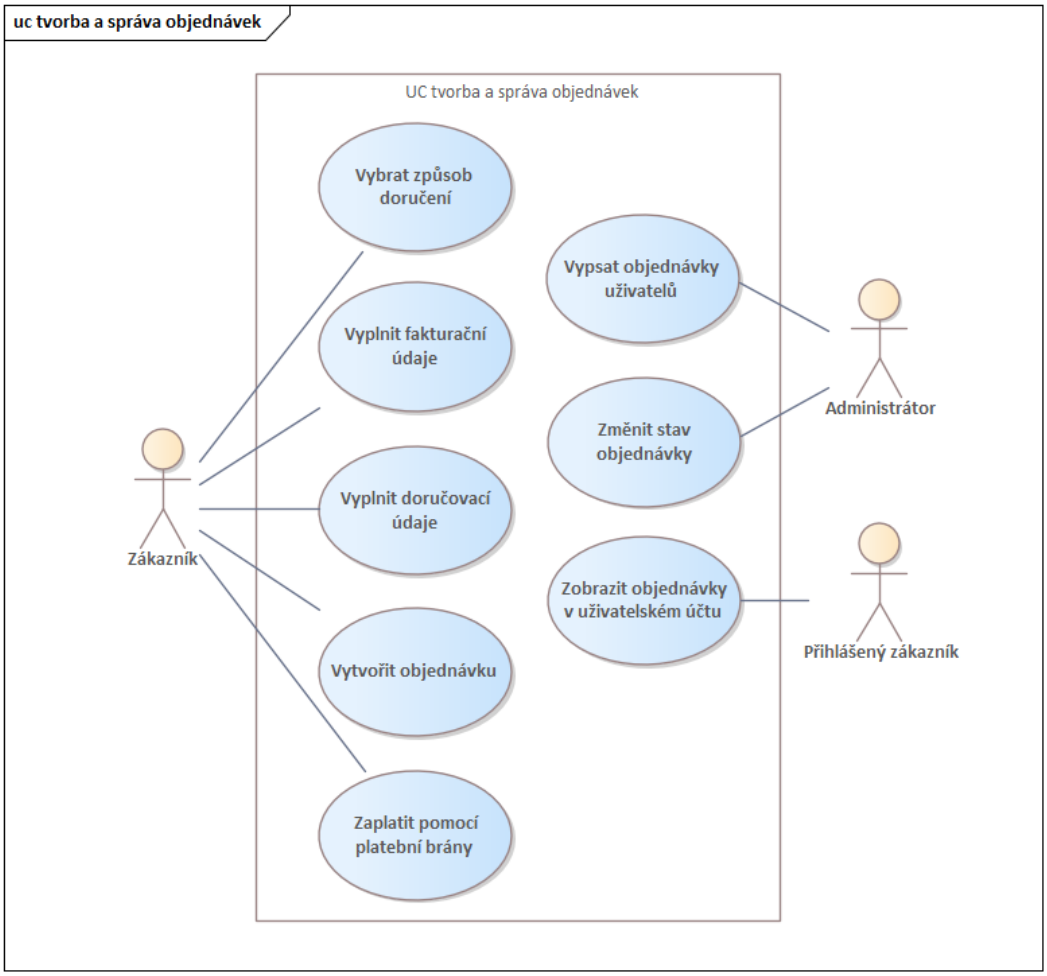


Obrázek 5.2. Diagram případů užití tykajících se správy a procházení produktů.

5.1.3 UC tvorba a správa objednávek

Obrázek 5.3 zobrazuje diagram případů užití reprezentujích funkcionality týkající se tvorby a správy objednávek. Pro aktéra zákazník diagram definuje interakce potřebné pro vytvoření a zaplacení objednávky. Přihlášený zákazník má navíc možnost zobrazit své objednávky v uživatelském účtu.

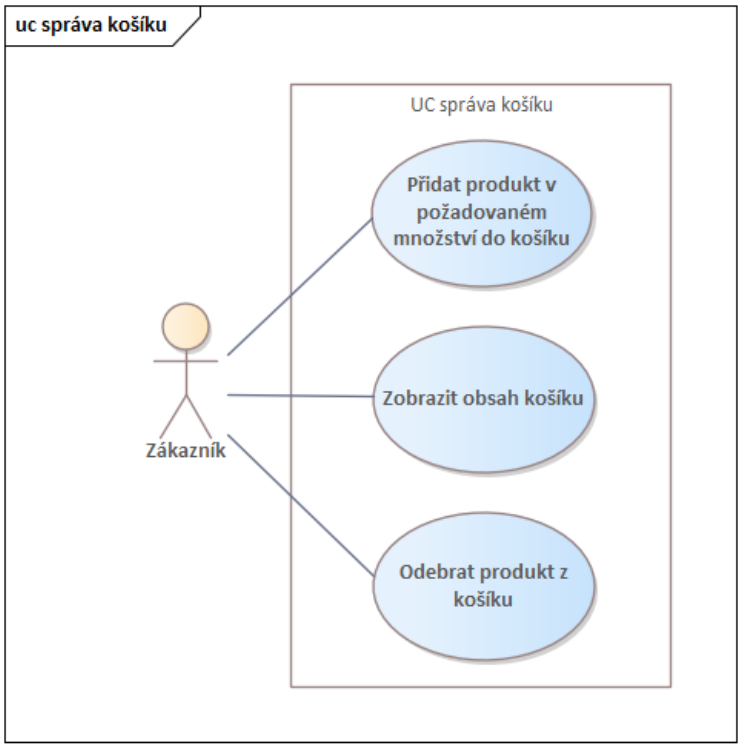
Pro aktéra typu Administrátor diagram definuje funkcionality potřebná ke správě jejich objednávek, jako jejich vypsání a změnu jejich stavu.



Obrázek 5.3. Diagram případů užití tykajících se tvorby a správy objednávek.

5.1.4 UC správa košíku

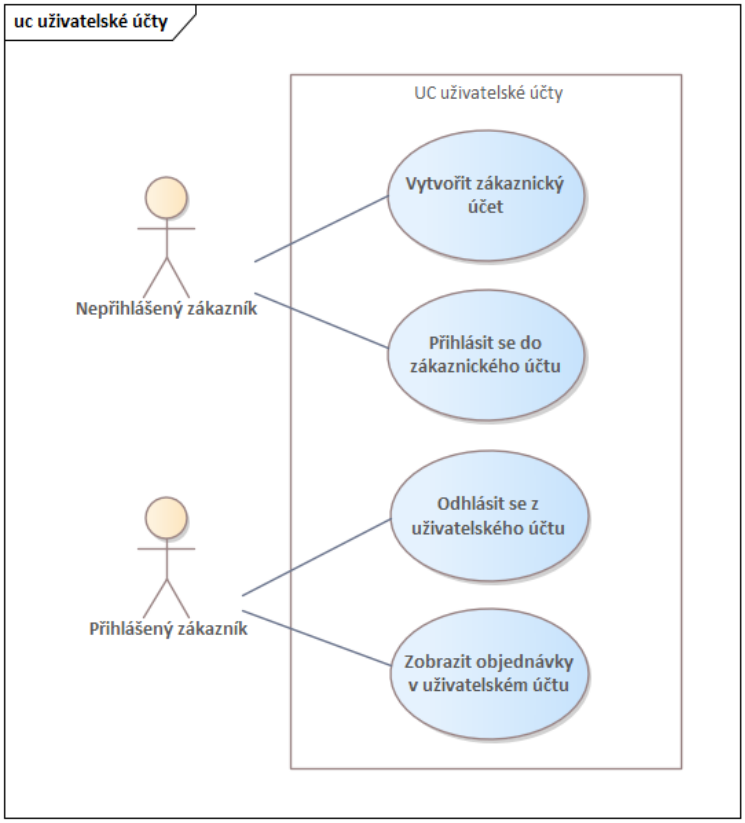
Diagram případů užití na obrázku 5.4 definuje funkcionality pro správu košíku. Aktér Zákazník může přidávat do košíku produkty v požadovaném množství, odebírat je a zobrazovat obsah košíku.



Obrázek 5.4. Diagram případů užití tykajících se správy košíku.

5.1.5 UC uživatelské účty

Obrázek 5.5 zachycuje diagram případů užití znázorňující funkcionality týkajících se uživatelských účtů. Aktér Nepřihlášený zákazník může uživatelský účet vytvořit, nebo se do účtu přihlásit, čímž se z něj stává aktér Přihlášený zákazník, kterému systém umožní zobrazení historie svých objednávek a odhlášení se z uživatelského účtu.



Obrázek 5.5. Diagram případů užití tykajících se uživatelských účtů.

5.2 Analytický doménový model

Na základě požadavků jsem navrhl analytický doménový model, který jsem následně rozdělil do tří menších modelů představujících tři mikroslužby, které budu následně implementovat. Funkcionalitu jednotlivých mikroslužeb a jednotlivé modely představím v následujících kapitolách.

5.2.1 User MS

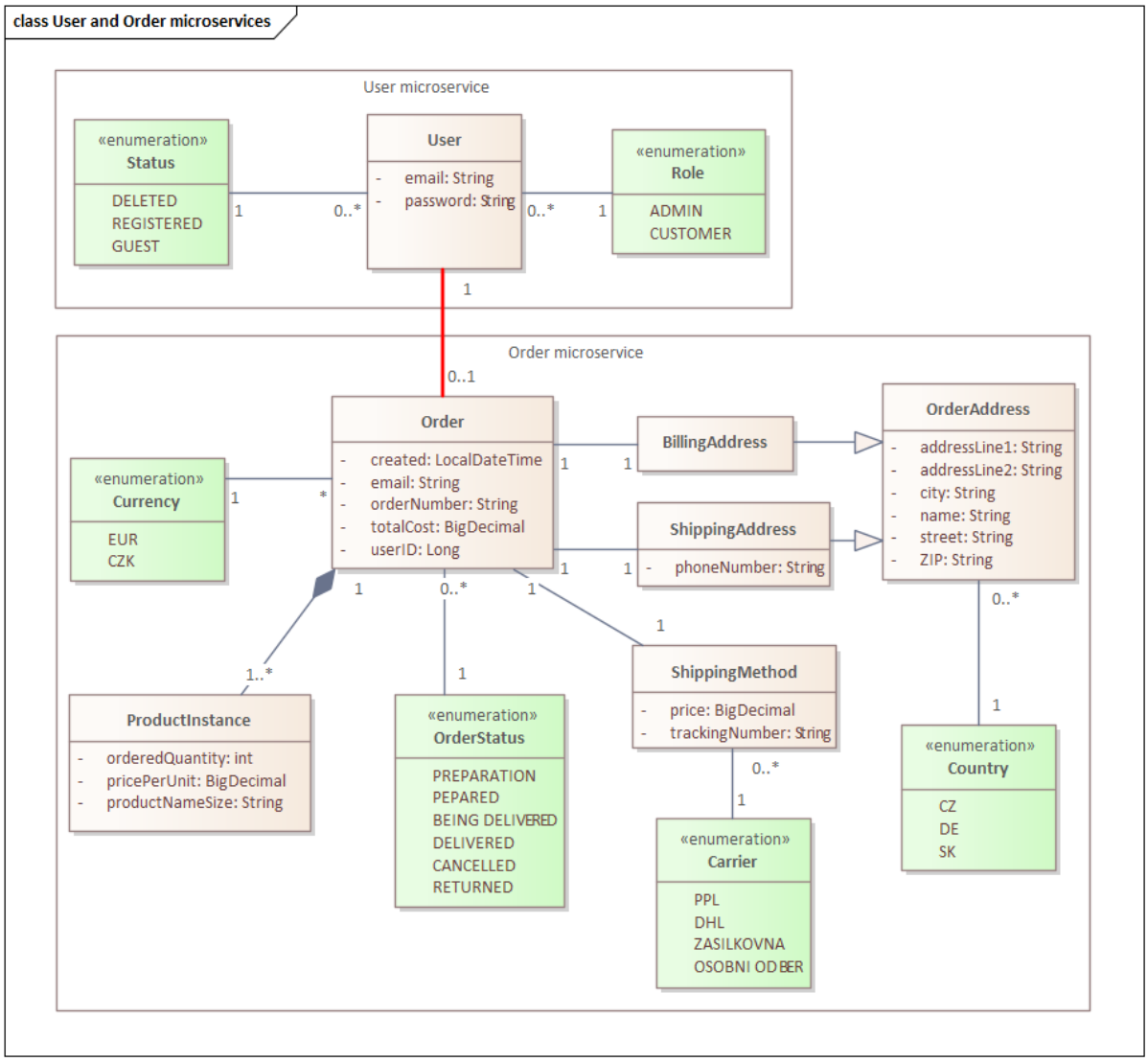
Schéma s názvem „User microservice“ na obrázku 5.6 představuje mikroslužbu spravující uživatelské účty, registraci a přihlašování uživatelů. Její aplikační logika je vystvěna nad enitou User, představuje uživatelský účet s příslušnou rolí, emailem a heslem. Dále má uživatel přiřazený status, pomocí kterého může provozovatel uživateli účet zablokovat, případně poskytuje možnost provést takzvaný „soft-delete“, kdy se uživatelský účet označí jako smazaný, ale interně data zůstanou zachována.

5.2.2 Order MS

Schéma s názvem „Order microservice“ na obrázku 5.6 představuje mikroslužbu spravující objednávky, skládá se z následujících entit:

- Order: Hlavní entita mikroslužby Order, na kterou se váže zbytek entit. Nese informace o objednávce zákazníka.
- ProductInstance: Váže se k objednávce a reprezentuje jednolivé položky, jež objednávka obsahuje.

- ShippingMethod: Nese doručovací informace dané objednávky.
- OrderAddress: Nese informace o adrese, které se na Order vážou prostřednictvím BillingAddress a
- ShippingAddress, které od OrderAddress dědí.
- ShippingAddress: Představuje adresu, na kterou má být objednávka doručena a rozšiřuje OrderAddress o telefonní číslo.
- BillingAddress: Představuje fakturační adresu objednávky.



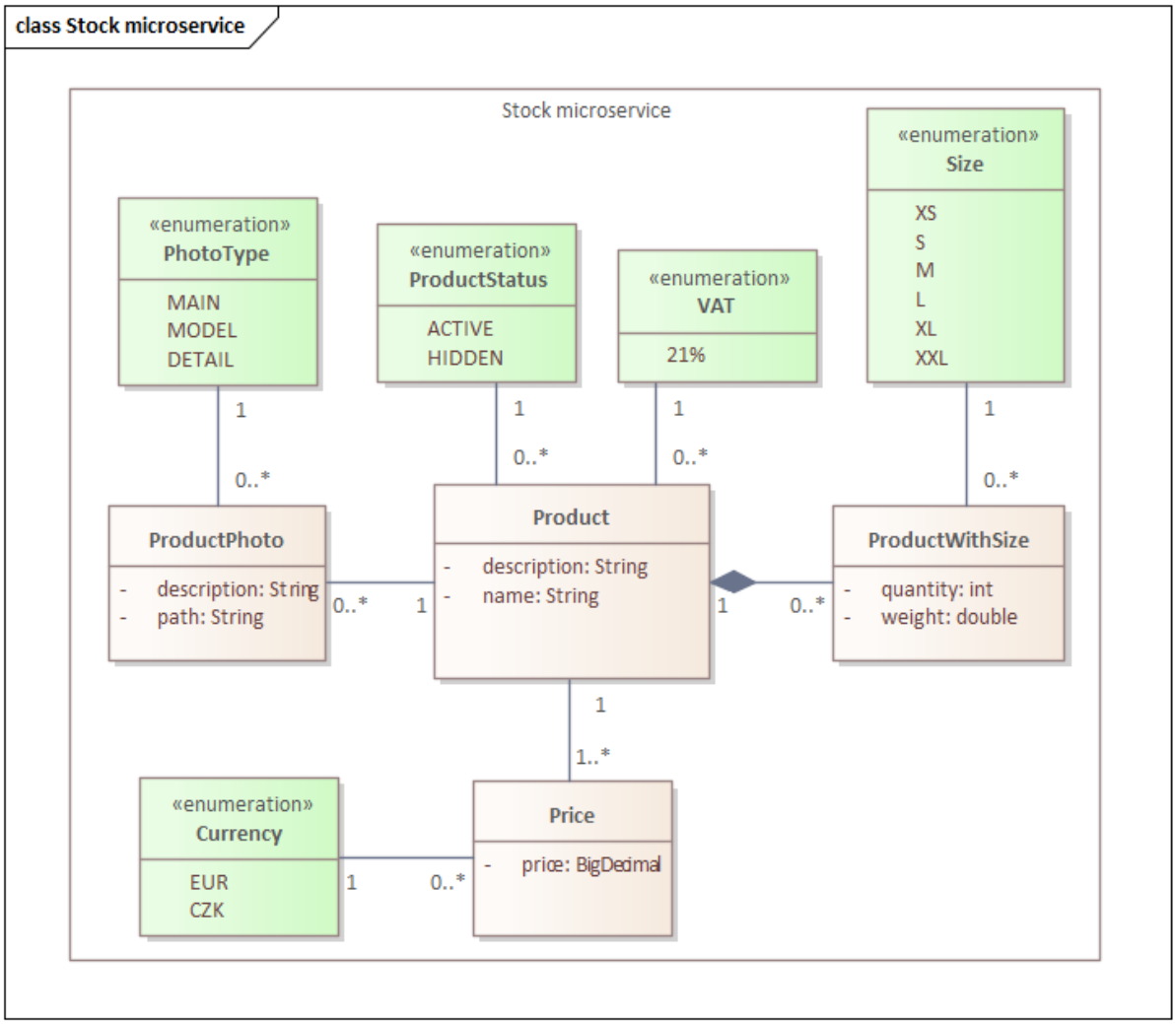
Obrázek 5.6. Analytický doménový model mikroslužeb User a Order.

5.2.3 Stock MS

Schéma s názvem „Stock microservice“ na obrázku 5.7 představuje mikroslužbu spravující produkty nabízené v e-shopu, skládá se z následujících entit:

- Product: Hlavní entita mikroslužby Stock, na kterou se váže zbytek entit. Nese informace o konkrétním produktu.

- ProductPhoto: Váže se k produktu, přiřazuje k produktu obrázky uložené v souborovém systému.
- Price: Reprezentuje ceny produktu v různých měnách.
- ProductWithSize: představuje konkrétní velikost produktu, uchovává informace o jejích skladových zásobách a váze produktu v dané velikosti.



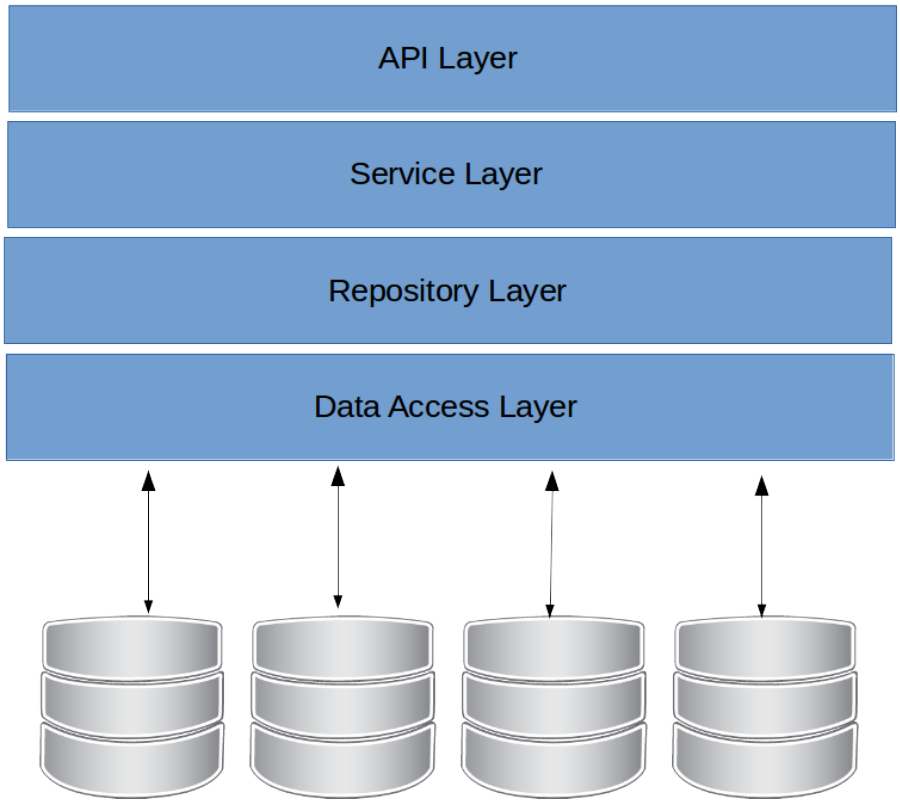
Obrázek 5.7. Analytický doménový model mikroslužby Stock.

5.3 Architektura mikroslužeb

Jednotlivé mikroslužby implementuji dle vícevrstvé architektury, zobrazené na obrázku 5.8. Jednotlivé vrsty mají následující význam:

- API Layer - aplikační rozhraní systému.
- Service Layer - veškerá byznys logika dané mikroslužby.
- Repository Layer - poskytuje operace nad databázovými entitami.
- Data Acess layer - poskytuje přístup k databázovým entitám.

Konkrétní implementaci vrstev popíši v kapitole 7.



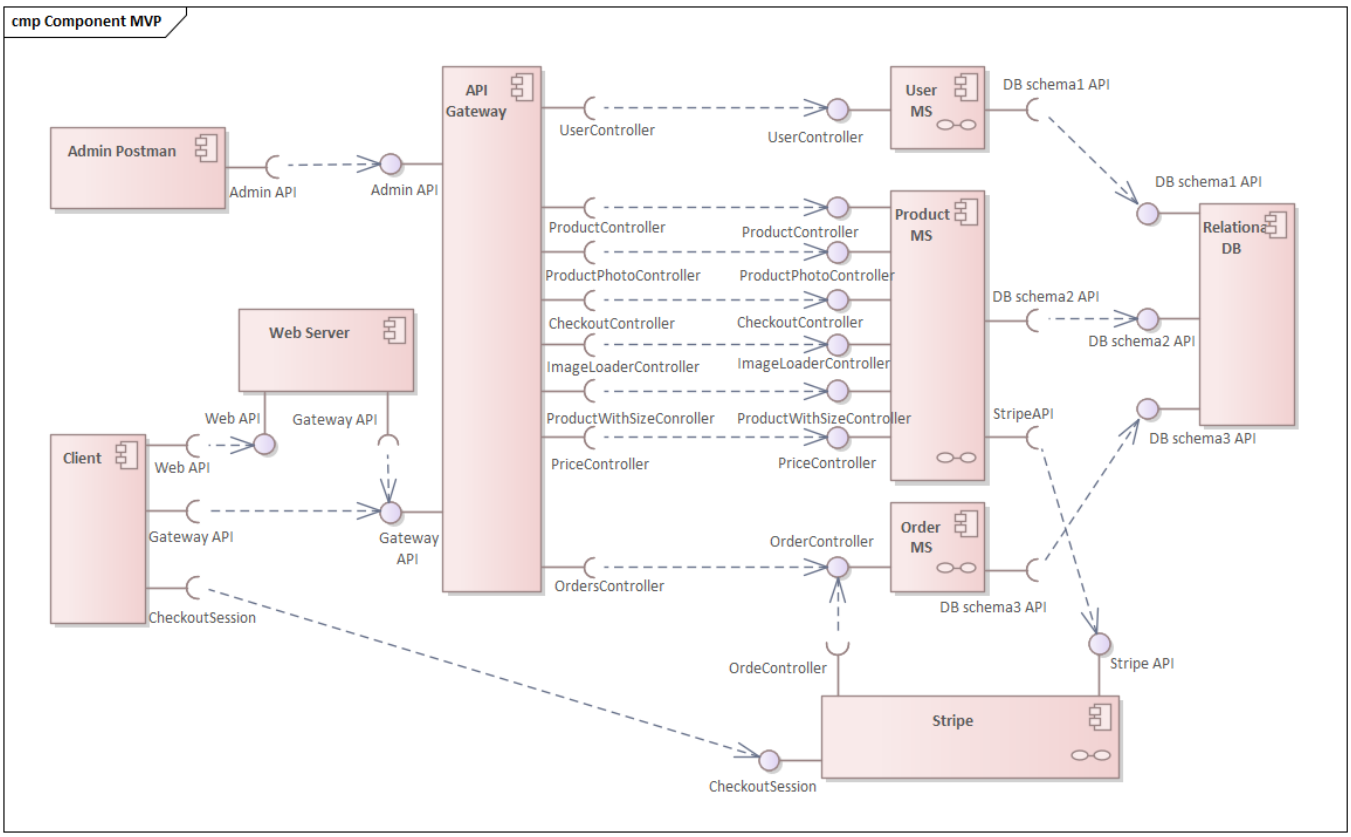
Obrázek 5.8. Ilustrační schéma vícevrstvé architektury. Zdroj: [32]

5.4 Komponenty systému

Komponenty vystavěné nad schématy analytického doménového modelu jsou spolu s dalšími komponentami figurujícími v systému znázorněny v komponentovém diagramu na obrázku5.9. Jednotlivé komponenty popisuji níže:

- Relational DB - ukládá data do třech různých schámat, ke kterým přistupují mikroslužby User MS, Product MS, Order MS. Schámata odpovídají class diagramům 5.6 a 5.7.
- User MS - odpovídá User microservice na class digramu 5.6, poskytuje správu uživatelských účtů, přihlašování a registraci.
- Product MS - odpovídá Product microservice na class digramu 5.6, poskytuje správu sortimentu a vytváří Stripe session pro tvorbu objednávek.
- Order MS - odpovídá Stock microservice na class digramu 5.7, poskytuje existující objednávky a ukládá nové objednávky na základě webhooků přjatých od Stripe.
- Stripe - služna třetí strany, starající se o vytváření a přijetí plateb za objednávky.
- API Gateway - stará se o autentizaci uživatelů a agreguje endpointy User MS, Product MS a Order MS pod jednou URL.
- Web Server - stará se o server-side rendering webových stránek, potřebná data získává přes Api Gateway.

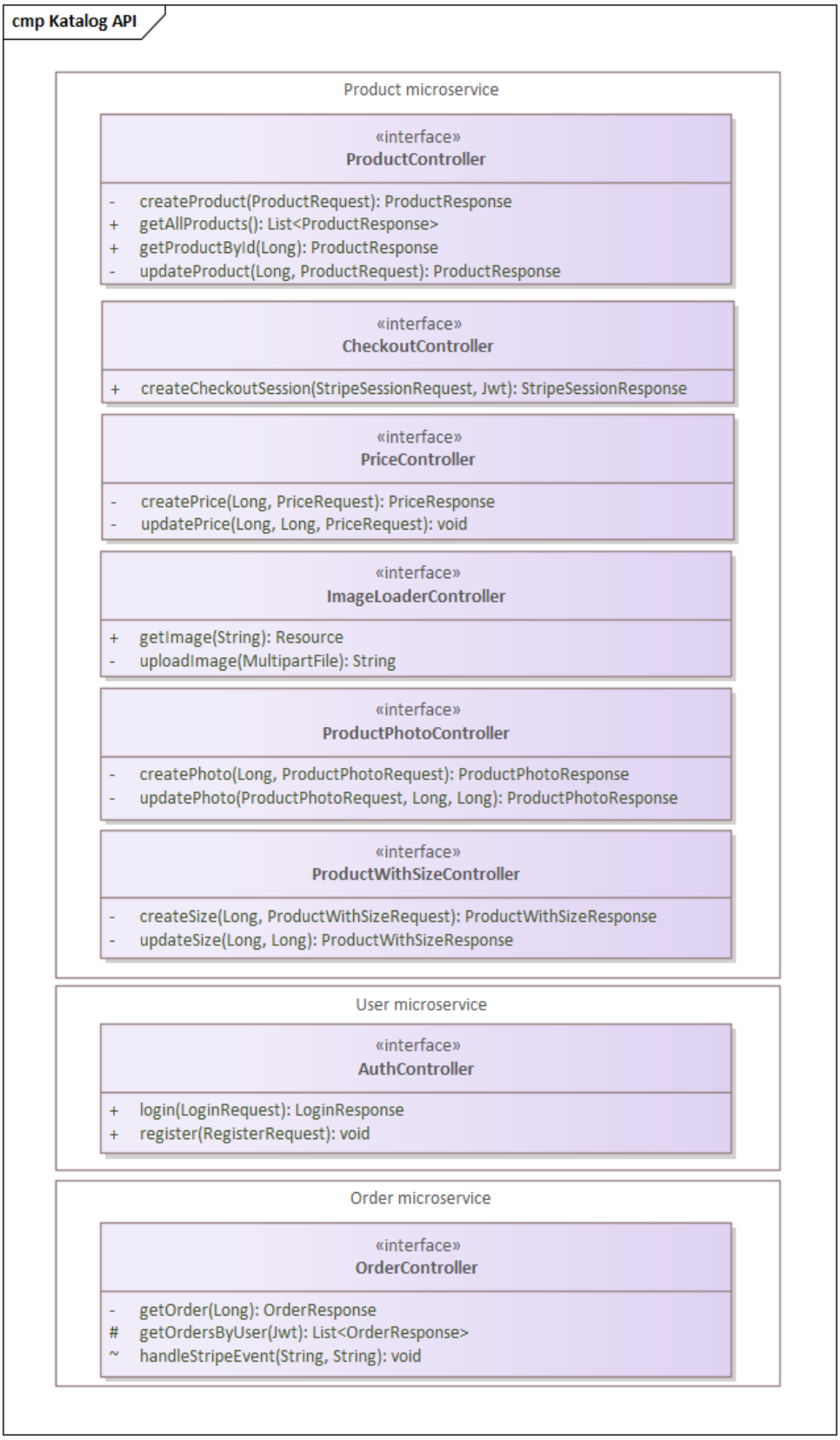
- Client - představuje prohlížeč zákazníka, stránky získává z komponenty Web Server, případně při client-side renderingu získává data přes Api Gateway.
- Admin postman - představuje aplikaci Postman na počítači správce systému.



Obrázek 5.9. Diagram komponent jádra systému.

Mnou implementované komponenty poskytnou aplikační rozhraní popsané v API katalogu na obrázku 5.10, autorizace přístupu k jednotlivým controllerům je zde znázorněna pomocí následujících značek:

- + controllery přístupné všem rolím.
- - controllery přístupné pouze administrátorovi.
- # controllery přístupné pouze přihlášenému uživateli.
- (vlnovka) controllery přístupné pouze službám třetích stran, jako je Stripe.



Obrázek 5.10. Katalog apliačních rozhraní implementovaných mikroslužeb.

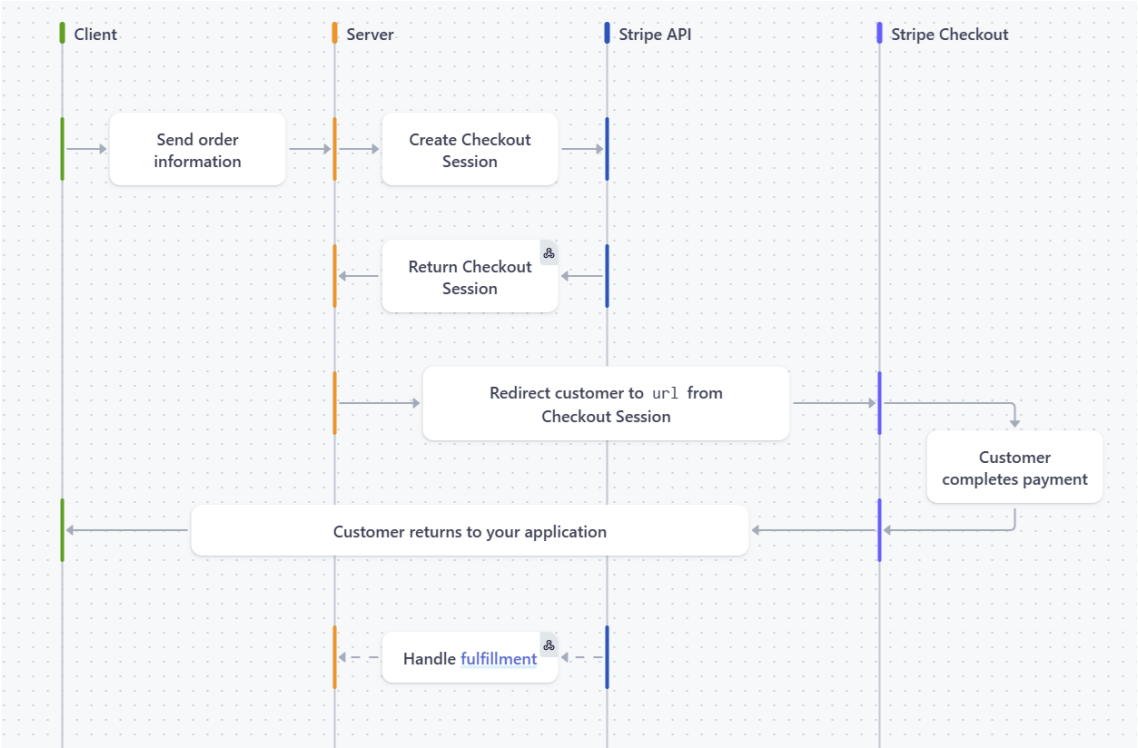
5.5 Integrace Stripe

Pro implementaci jádra jsem se rozhodl v rámci ověření konceptu prozatím implementovat platební bránu Stripe, jelikož kromě platební brány nabízí kompletní proces tvorby objednávky, tedy získání všech informací potřebných pro tvorbu objednávky od zákazníka a následné zpracování platby.

Problém, který řeším, spočívá v tom, že během procesu tvorby objednávky musí mikroslužba Stock MS ověřit skladovou dostupnost a případně vyskladnit položky, ale údaje o provedení objednávky musí uložit mikroslužba Order MS, přičemž případné předání údajů mezi mikroslužbami musí pro zachování jejich nízké závislosti proběhnout asynchronně. S tím mi pomůže Stripe, pokud ho vhodně integruji do svého systému.

Na obrázku 5.11 je zobrazen sekvenční diagram zpracování objednávky pomocí Stripe. Server zde přijímá informace o objednávce, tedy obsah košíku zákazníka. O přijetí tohoto požadavku se postará Stock MS, která ověří skladovou dostupnost, vyskladní položky, vytvoří Stripe Session a přesměruje na ni zákazníka. Zákazník pomocí Stripe Session vyplní doručovací a fakturační údaje a provede platbu.

Následně Server přijímá od Stripe prostřednictvím webhooku informace z vytvořené objednávky, tyto informace přijme Order MS a vytvoří v systému novou objednávku. Díky takto implementovanému řešení provedou Order MS a Stock MS nutné úkony, přičemž asynchronní komunikace zůstane zachována, aniž bych musel využít další technologie pro asynchronní komunikaci.



Obrázek 5.11. Sekvenční diagram zpracování objednávky a příslušné platby pomocí Stripe. Zdroj: [33]

5.6 Wire-frames

Před implementací uživatelského rozhraní je vhodné mít povědomí alespoň o rozložení prvků a funkcionalitách jednotlivých stránek, k tomu slouží takzvané wireframy, statické

návrhy jednotlivých stránek, vytvořené ať už ručně na papír, nebo pomocí digitálního nástroje. [34]

V příloze A jsou k nahlédnutí wireframy vytvořené jako součást návrhu pro implementaci této práce. Některé wireframy sdílejí záhlaví a zápatí pro navigaci napříč e-shopem. Význam samotných obrazovek je následující.

- Obrázek ?? představuje návrh domovské stránky e-shopu, s výpisem produktů a ilustrační grafikou.
- Obrázek ?? představuje návrh stránky s registračním formulářem.
- Obrázek ?? představuje návrh s přihlašovacím formulářem.
- Obrázek ?? představuje návrh detailu produktu.
- Obrázek ?? představuje návrh stránky s obsahem košíku.
- Obrázek ?? je snímek obrazovky ze Stripe Demo, analogicky Stripe použiji v implementaci.
- Obrázek ?? představuje návrh stránky uživatelského profilu s výpisem objednávek.

Kapitola 6
Použité technologie

V této kapitole představím soubor klíčových technologií, které využívám pro implementaci serverové a klientské části.

6.1 Verzování

Pro verzování využívám technologii Git, open-source distribuovaný verzovací systém, což znamená, že vývojář má na svém počítači kompletní kopii repozitáře, který se nachází na vzdáleném serveru. Já jsem se pro tento projekt rozhodl využít vzdálený server poskytovaný službou GitHub.

Git nabízí například možnosti vytváření nových větví ze stávajících, jejich opětovné slučování či přidávání a rušení změn jednotlivých větví. Lokální a vzdálenou verzi je potřeba ručně synchronizovat, čímž můžou při práci ve více lidech vznikat konflikty, které jsou potřeba dodatečně řešit.

Aby práce s repozitářem byla systematická, existují takzvané workflows, které definují, jak s verzovacím systémem pracovat. I když síla těchto přístupů vynikne zejména při práci v týmu, rozhodl jsem se pro přehlednost a případnou budoucí týmovou práci využít workflow Gitflow.

Gitflow je charakteristický tím, že z hlavní master větve vychází develop větev a z té jsou vytvářeny větve pro implementaci jednolivých funkcionalit, takzvané features. Features se poté slučují zpět do develop větve a ve chvíli, kdy je develop větev připravena k nasazení do produkce, vytvoří se z ní takzvaná release větev, ve které se odladí finální nedostatky, aniž by to bylo narušeno přidáváním feature větví do develop větve. Ve chvíli, kdy je release větev odladěna, je připojena do master větve a tím nasazena do produkce. Pokud se vyskytne problém s aktuální produkční verzí, z master větve se vytvoří takzvaná hotfix větev, kde se chyba co nejrychleji opraví a následně se hotfix větev sloučí opět do masteru. [35]

6.2 Server

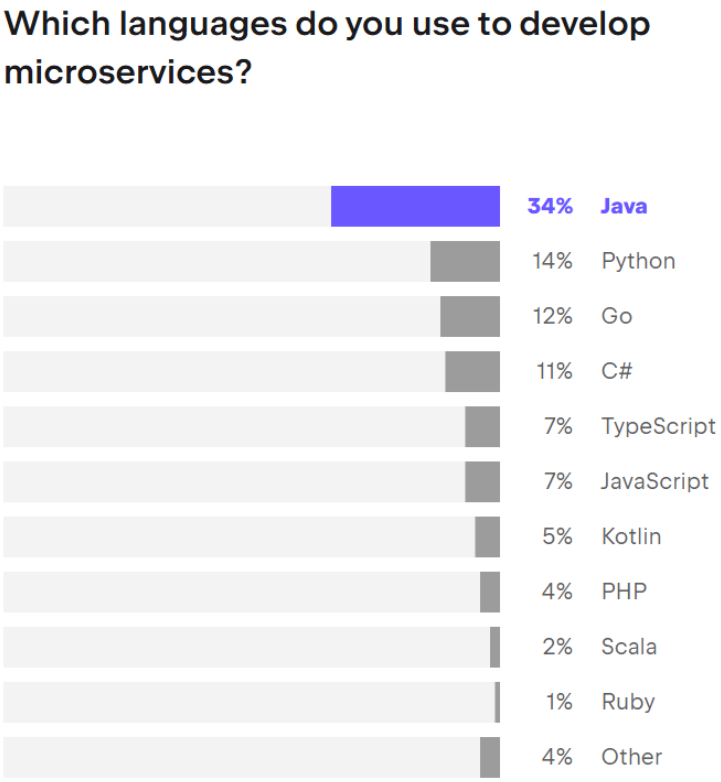
6.2.1 Databáze

Pro ukládání dat na serverové straně jsem zvolil open-source relační databázi PostgreSQL, která díky ACID vlastnostem (atomicita, konzistence, izolace a trvanlivost), nabízí vysokou odolnost vůči chybám, protože neúplné změny nejsou nikdy uloženy. [36] Toho budu využívat během psaní transakčních metod v servisní vrstvě, což popíši v kapitole 7.1.2.

6.2.2 Programovací jazyk

Pro implementaci serverové části aplikace jsem zvolil programovací jazyk Java, který byl představen v roce 1995 a řadí se mezi nejpoužívanější programovací jazyky na světě,

v žebříčku PYPL PopularitY of Programming Language [37] se v květnu 2025 umístil na druhém místě. [38] Díky tomu existuje nepřeberné množství dokumentace, tuoriálů, knihoven a frameworků, které lze při implementaci využít. Dle průzkumu společnosti JetBrains [39] Java žebříčku popularity jazyků pro implementaci mikroslužeb naprosto dominovala, což je zobrazeno na obrázku 6.1.



Obrázek 6.1. Popularita programovacích jazyků pro implementaci mikroslužeb. Zdroj: [39]

Programovací jazyk Java nabízí také spoustu technologických výhod.

- Jeho spouštění je nezávislé na platformě díky Java Virtual Machine (JVM).
- Java je objektově oriantovaný jazyk, což umožňuje znovupoužitelnst částí kódu.
- Umožňuje vícevláknového zpracování.
- Automaticky spravuje paměť pomocí garbage collectoru.

Samozřejmě jsou pro ni charakteristické i určité nevýhody.

- Java je oproti jiným jazykům pomalejší, protože spotřebovává značné množství paměti.
- Jeho náročnost na operační paměť je tedy další nevýhodou.
- Nevýhodou je i pomalé spouštění kvůli inicializaci JVM. [38]

Analogicky bych mohl prezentovat výčet výhod a nevýhod dalších programovacích jazyků používaných pro implementaci mikroslužeb, jako jsou například Python, Go, C# či TypeScript, ale silným agrumentem je vždy zejména zkušenost programátora s danou technologií. A pravě Java byla jazykem nejvíce spojeným s mým studiem na ČVUT FEL. [39]

Na základě výše uvedených argumentů jsem se rozhodl právě pro využití programovacího jazyka Java.

6.2.3 Framework

Jak jsem již zmínil v předchozí kapitole, díky silné komunitě vývojařů používajících Javu existuje velké množství veřejně dostupných zdrojů, které je možné využít pro vlastní implementaci. Mezi tyto zdroje patří i open-source framework Spring Boot, který je rychlou a snadnou cestou pro vývoj Java aplikací. Díky auto-konfiguraci obsahuje přednastavené paramatry a eliminuje složité konfigurace z tradičních Spring aplikací. Prostřednictvím Spring Boot starters nabízí specifikaci závislostí spíše na základě potřebných funkcionalit, než na základě konkrétních knihoven a verzí, jelikož Spring Boot starters představují díky traznitivním závislostem balíčky vzájemně kompatibilních knihoven a jejich verzí.[40]

Zároveň Spring Boot nabízí podporu mnoha technologií ze Spring ekosystému, jako Spring Security, či Spring Data. Je nadstavbou nad Spring Framework, tudíž využívá jeho klíčové principy Inversion Of Control (IoC) a Dependency Injection (DI). IoC je realizováno prostřednictvím Spring kontejneru, který se stará o vytváření objektů a jejich životní cyklus. Též se stará o propojování objektů pomocí DI, což je vkládání závislostí na základě toho, co objekty potřebují k vykonávání svých funkcionalit.[41]

S frameworkem Spring Boot jsem se též hojně setkával během svého studia na ČVUT FEL a je široce využíván v oblastech, kterým bych se rád v budoucnu věnoval. Zároveň je na základě výše uvedených argumentů vhodný pro implementaci serverové strany aplikace, proto jsem se rozhodl pro jeho využití jako výchozí technologie pro implementaci.

6.2.4 API

Jako komunikační rozhraní jednotlivých mikroslužeb jsem se rozhodl pro použití REST API, což je Application Programming Interface (API), odpovídající následujícím šesti principům architektonického stylu Representational State Transfer (REST).

- Uniform interface - musí zaručit, že každý zdroj má svůj vlastní unikátní identifikátor (URI). Všechny dotazy dotazující identická data musí být též identické.
- Bezstavovost - server zpracovává každý dotaz naprosto izolovaně, nezávisle na všech předchozích dotazech. Každý dotaz tedy musí obsahovat všechny infromace potřebné k jeho zpracování.
- Oddělení klient - server - kompletní nezávislost mezi klientem a serverem, jediné co klient zná, je URI dotazovaných prostředků a server pouze vrátí dotazovaná data, žádnou jinou cestou spolu nekomunikují.
- Vrstvený systém - mezi klientem a serverem se mohou nacházet další vrstvy, starající se například o zabezpečení, nebo logiku. Servery si mohou data navzájem předávat a agregovat je do odpovědí, ale klient neví, jestli komunikuje přímo s koncovou aplikací a naopak.
- Uložitelnost do cache - data, u kterých je to možné, by mělo být možné ukládat do mezipaměti u klienta, nebo na vrstvách mezi klientem a zdrojem dat.
- Kód na požádání (volitelné) - server může na vyžádání dynamicky rozšířit funkčnost klienta prostřednictvím zaslaného spustilného kódu, například JavaScriptu, nebo Java appletu.

Dodržení těchto principů při implementaci rozhraní sebou přinaší důležité benefity, díky bezstavovosti a uložitelnosti do cache je systém velmi dobře škálovatelný. Vrstvení systému společně s oddělením serveru a klienta umožňuje měnit, či rozšiřovat vnitřní funkcionality serverové aplikace, ale zároveň zachovávat nezměněné komunikační rozhraní, což poskytuje značnou flexibilitu. Nezávislost na použité technologii umožňuje použití odlišných programovacích jazyků na serverové a klientské straně.[42, 43, 44]

6.3 Klient

Při výběru technologií pro tvorbu klientské části aplikace jsem musel zohlednit způsob, jakým probíhá komunikace s REST API, jaká data potřebuji ukládát a nutnost budoucí optimalizace pro vyhledávače SEO, což je optimalizace webové stránky pro takzvané crawlery, nástroje vyhledávačů, které prochází webové stránky a vyhodnocují, pro jaká vyhledávání se má daná stránka zobrazovat a na jaké pozici. Pokud má tedy webová stránka kvalitní SEO, bude se pro relevantní klíčova slova zobrazovat mezi prvními výsledky vyhledávání, což je pro internetový obchod klíčové [45].

6.3.1 Next.js

Jelikož některé vyhledávače využívají crawlery, které neumí spouštět Javascript, je vhodné kvůli SEO využít statické a dynamické renderování, kde se HTML stránka s daty sestaví již na serverové straně a crawler ji tak může zpracovat. [46] Zároveň je ale pro komponenty, které nejsou důležité pro vyhledávače, stále vhodné využívat vykreslování na straně klienta, kdy se nemusí načítat celá nová stránka, ale ze severu se načtou potřená data a vykreslí se nová komponenta, což snižuje vytížení serveru. Proto jsem se rozhodl pro React framework Next.JS, který všechny tyto funkcionality nabízí. [47].

Kapitola 7
Implementace

V této kapitole popíši, jaké problémy jsem při implementaci řešil a vysvětlím, jakým způsobem a pomocí jakých technologií jsem tak učinil.

7.1 Server

7.1.1 Persistentní vrstva

Abych mohl v mikroslužbách pracovat s databází, využívám Spring Boot starter balíček Spring Data JPA, který umožní pomocí jedné závislosti přidat do projektu kompletní řešení. S databázovými zápisy mohu tak namísto psaní SQL dotazů pracovat jako s Java objekty pomocí metod a atributů, a to díky objektově-relačnímu mapování (ORM), jehož chování definuje rozhraní Jakarta Persistence (JPA), implementované frameworkem Hibernate, který zajišťuje komunikaci s databází prostřednictvím Java Database Connectivity (JDBC). [48, 49]

Konkrétně jako repozitář využívám CrudRepository ze Spring Data JPA, což je abstrakční vrstva nad JPA, která nabízí implementvané operace pro zápis, čtení, aktualizaci a mazání záznamů. Navíc je možné přidávat další databázové dotazy jako signatury Java metod. [50] Použití CrudRepository demonstruji na ukázce 7.1, kde pomocí signatur Java metod přidávám operace pro získání všech objdanávek přiřazených danému uživateli na základě jeho unikátního identifikátoru či emailu.

```
public interface OrderRepository extends CrudRepository<Order, Long> {
    Optional<List<Order>> findAllByUserId(@Param("user_id") Long userId);
    Optional<List<Order>> findAllByEmail(@Param("email") String email);
}
```

Ukázka 7.1. Ukázka použití CrudRepository pro správu databáze.

7.1.2 Servisní vrsta

V servisní vrstvě se nachází veškerá logika jednotlivých mikroslužeb, využívá operací poskytovaných rozhraním persisitentní vrsty, které volá pomocí transakčních metod, které respektují ACID vlastnosti databáze. Mimo interakce s persistentní vrstvou využívám servisní vrstvu pro komunikaci se službami třetích stran, konkrétně se službou Stripe, což popisuji v kapitole 5.5.

Atributy, metody, a třídy servisní vrstvy dokumentuji pomocí Javadoc komentéřů přímo v kódu.

7.1.3 Implementace API

Pro implementaci aplikačního rozhraní mikroslužeb používám balíček Spring Boot Starter Web, který obsahuje vestavěný Tomcat server pro komunikaci prostřednictvím protokolu HTTP, což je synchronní způsob komunikace, který je pro implementaci REST API běžnou praxí. HTTP metody pak specifikují příslušné operace následovně:

- GET - čtení záznamu.
- POST - vytvoření záznamu.
- PUT - aktualizace prostřednictvím kompletního nahrazení záznamu.
- PATCH - aktualizace záznamu upravením jeho části.
- DELETE - smazání záznamu.

Hlavičky HTTP dotazů mohou obsahovat množství metadat, jako autentizační a autorizační tokeny, stavové kódy informující o výsledku prováděné operace, či formát přenšených dat. Pro přenos dat jsem zvolil formát JavaScript Object Notation (JSON) a o jeho mapování na Java objekty specifikujícími strukturu přenášených dat (DTO) se starjí HTTP Message Convertory z Spring Web MVC, které využívají k tomuto mapování knihovnu Jackson. [51] [52]

7.1.4 Validace

V samotných DTO pomocí anotací z knihovny Jakarta Bean Validation určuji formu přijímaných dat, jako povinná pole či strukturu atributů, například formát emailu či hesla. Tímto přístupem zabezpečuji, že serverová strana aplikace zpracuje pouze data v požadovaném formátu.

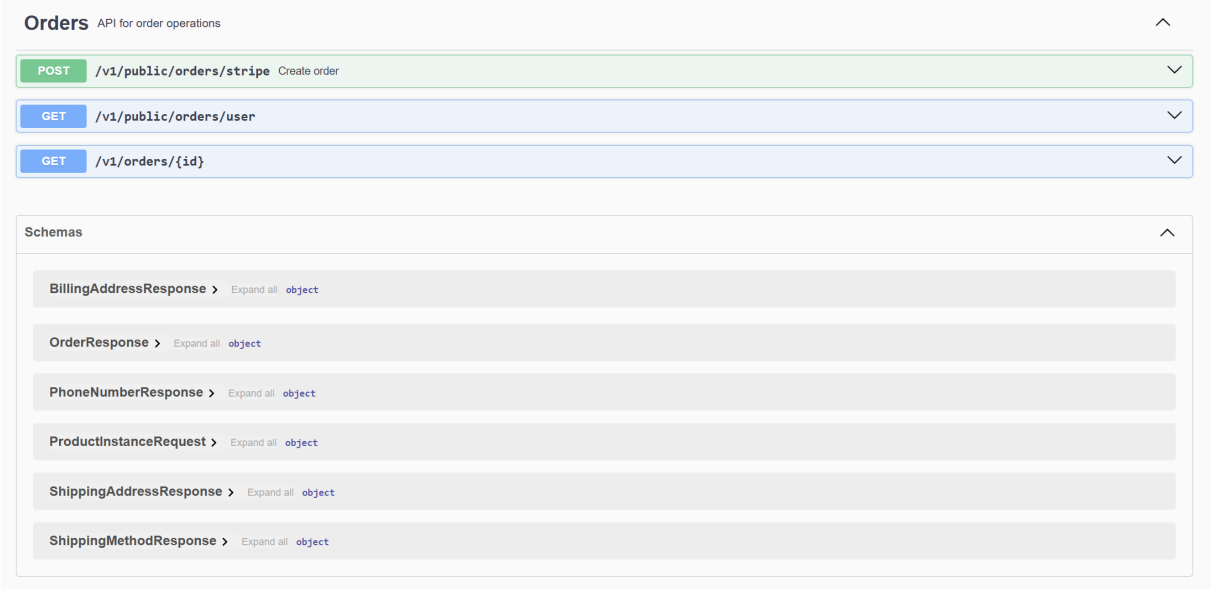
V ukázce 7.2 prezentuji použití zmíněných validačních anotací pro specifikaci toho, že atribut přijímá data pouze ve tvaru e-mailové adresy. Anotace @Email zajišťuje formát řetezce a @NotBlank že řetezec není prázdný, „null“, nebo se neskládá pouze z bílých znaků. [53, 54]

```
@Email(message = "Email must be a valid email address")
@NotBlank(message = "Email must be filled")
private String email;
```

Ukázka 7.2. Ukázka validace atributu DTO pomocí knihovny Jakarta Bean Validation.

7.1.5 Dokumentace rozhraní

Pro dokumentaci API využívám knihovnu Springdoc OpenAPI, která na základě příslušných anotací u kontrolerů a DTO sestaví OpenAPI specifikaci. Dokumentaci je následně možné prohlížet a provolávat API pomocí předpřipravených dotazů prostřednictvím Swagger UI, což je grafické rozhraní dostupné za běhu aplikace přes webový prohlížeč, jehož ukázka je k nahlédnutí na obrázku 7.1. [55, 56]



Obrázek 7.1. Ukázka dokumentace OpenAPI vizualizované pomocí Swagger UI.

7.1.6 Mapování objektů

Pro vzájemné mapování mezi DTO a objekty, se kterými pracují servisní a persistentní vrstvy jsem využil knihovnu MapStruct, nabízející generátor kódu zjednodušující mapování Java beanů založený na přístupu konvence nad konfigurací, jež zaručuje při dodržení konvencí velmi málo práce pro vývojáře. [57]

Musíme vytvořit pouze inteface mapovací třídy a jejích metod, případně doplnit, jak se mají mapovat jednotlivé atributy tříd, pokud se liší v názvu a případně doplnit mappery pro vnořené objekty. Mapstruct se postará o automatické vytvoření konkrétní implementace během kompilace. [57, 58]

V ukázce 7.3 využívám MapStruct pro vytovoření mapperu, který nabízí metodu „toResponse“ pro mapování objektu třídy Order na objekt třídy OrderResponse. Dále nabízí metodu pro mapování datové struktury „List“ obsahující objekty třídy Order na „List“ obsahující objekty třídy OrderResponse. V ukázce je také vidět použití mapperu BillibgAddressMapper, který zajišťuje mapování atributu BillingAddress.

```
@Mapper(componentModel = "spring", uses = {
        BillingAddressMapper.class,
        ...
})
public interface OrderMapper {
    @Mapping(source = "billAddress", target = "billAddressResponse")
    ...
    OrderResponse toResponse (Order order);

    default List<OrderResponse> toResponseList(List<Order> orders) {
        return orders.stream()
                .map(this::toResponse)
                .toList();
    }
}
```

Ukázka 7.3. Ukázka použití knihovny MapStruct pro vytvoření mapperu.

7.1.7 Lombok

Dalším nástrojem, řídícím se přístupem konvence nad konfigurací, který mi pomohl vyhnout se psaní množství generického kódu, je nástroj Lombok. Lombok díky jednoduchým anotacím ušetří vývojaři psaní getterů, setterů, konstruktorů, toString a dalších metod opakujících se napříč třídami.[59] Lombok se zapojuje do build procesu a během kompilace automaticky generuje Java bytecode do .class souborů na základě použitých anotací. [60]

Příklad použití anotací z nástroje Lombok pro generování getterů, setterů a bezparamterického konstruktoru je k vidění v ukázce 7.4.

```
@Getter
@Setter
@NoArgsConstructor
public class ProductRequest {
    private String description;
    private String name;
    private VisibilityStatus visibilityStatus;
    private VAT vat;}
```

Ukázka 7.4. Použití anotací @Getter, @Setter a @NoArgsConstructor pro generování getterů, setterů a bezparametrického konstruktoru u DTO ProductRequest.

7.1.8 Autentizace

Jelikož používám REST API, je vhodné použít způsob autentizace, který dodrží jeho princip bezstavovosti, což by přihlašování na bázi session porušovalo. V tomto případě se navíc jedná o mikroservisní architekturu a session instance by byla vázána na konkrétní mikroslužbu, čímž by vznikl další problém, spočívající v umožnění komunikace klienta se všemi částmi systému.

Proto jsem se rozhodl přistoupit k bezstavovému způsobu autentizace pomocí OAuth2 Resurce Serveru a JSON Web Tokenů (JWT). Za přihlašování a registraci uživatelů je zodpovědná mikroslužba User MS, o ověřování tokenů se stará API gateway.

Při registraci uživatele se do databáze uloží email a hash hesla uživatele. Následně se při přihlašování porovná hash zadaného hesla s hashem v databázi a pokud dojde ke shodě, klentovi je vydán JWT token obsahující jeho email, roli a časová razítka vytvoření a expirace. Token je následně podepsán algoritmem HmacSHA256 a tajným klíčem.

Ve chvíli, kdy se klient pokusí přistoupit k rozhraní vyžadujícímu autentizaci, poskytne serveru v HTTP hlavičce jako součást dotazu token ve tvaru „Authorization: Bearer <jwt-token>“. OAuth2 Resource Server pomocí tajného klíče a algoritmu HmacSHA256 ověří platnost podpisu a expiraci tokenu. Pokud je ověření úspěšné, sytém přistoupí k autorizaci, které se budu věnovat v kapitole 7.1.9. V případě zamítnutí přístupu na základě autentizace je klient odmítnut se stavovým kódem 401 - Unauthorized.

Některá rozhraní nevyžadují autentizaci, jako například rozhraní pro získání nabízených produktů. URL takovýchto rozhraní obsahuje prefix „.../v1/public/...“ a Security Filter Chain je konfigurován tak, aby tyto dotazy vykonal bez autentizace.

7.1.9 Autorizace

Autorizaci jednotlivých API specifikuji přímo u metod controllerů pomcí anotací z knihovny Spring Boot Security 6, jako jsou například anotace:

- @PreAuthorize(`permitAll()`) - nevyžaduje žádnou roli.
- @PreAuthorize(`hasRole('ADMIN')`) - vyžaduje konkrétní roli, v tomto případě roli ADMIN.
- @PreAuthorize(`hasAnyRole('ADMIN', 'USER')`) - vyžaduje jednu z uvedených rolí, v tomto případě ADMIN, nebo USER.

Během autentizace extrahuje OAuth2 Resource Server z JWT tokenu mimo jiné uživatelské role klienta a uloží je do Spring Security Contextu. Spring Securtity následně porovná roli uvedenou v anotaci s rolí uloženou v Security Contextu, pokud je podmínka uvedená pomocí anotace splněna, dotaz je vykonán. V opačném případě je klient odmítnut se stavovým kódem 403 - Forbidden.

V ukázce 7.5 využívám k nastavení authorizačních pravidel anotaci @PreAuthorize. JWT token metoda přijímá jako vstupní parametr „@AuthenticationPrincipal Jwt principal“.

```
@PreAuthorize("hasAnyRole('ADMIN', 'CUSTOMER')")
    @GetMapping("public/orders/user")
    public ResponseEntity<List<OrderResponse>> getOrdersByUser(
                @AuthenticationPrincipal Jwt principal) {
        ...
        return ResponseEntity.ok(orderResponses);}
```

Ukázka 7.5. Ukázka authorizace pomocí anotací z knihovny Spring Boot Security 6.

7.1.10 API Gateway

Díky API gateway je veškerá komunikace se serverem centralizována a systém tak vystupuje jako jedna komponenta, to umožňuje sjednotit řešení takzvaných cross-cutting concerns, což jsou funkcionality týkající se více modulů. Jsou jimi například:

- bezpečnost,
- CORS,
- či směrování. [61, 62]

Pro implementaci API gateway využívám Spring Cloud Gateway, postavenou na technologii Web Flux, umožňující neblokující zpracování webových požadavků. Spring Cloud Gateway poskytuje routování, filtrování a autentizaci požadavků. [61, 62, 63]

V ukázce 7.6 prezentuji směrování požadavků z vnější sítě na mikroslužbu Order MS ve vnitřní síti pomocí Spring Cloud Gateway.

```
spring:
  cloud:
    gateway:
      routes:
        - id: order-service
          uri: http://order-service:8082
          predicates:
            - Path=/order/**
          filters:
            - StripPrefix=1
```

Ukázka 7.6. Ukázka směrování požadavků prostřednictvím Spring Cloud Gateway.

7.1.11 Nasazení

Serverovou část nasazuji pomocí nástroje Docker, který mi poskytuje konzistentní a izolované prostředí pro běh jednotlivých mikroslužeb. Pomocí nástroje docker-compose vytvářím kontejnery user-service, order-service, stock-service, api-gateway a databázi PostgreSQL. Každá služba má vlastní Dockerfile a všechny jsou propojeny v rámci jedné sítě app-network. Pro předávání citlivých údajů, jako jsou přihlašovací údaje k databázi nebo JWT klíč, využívám prostředí proměnných.

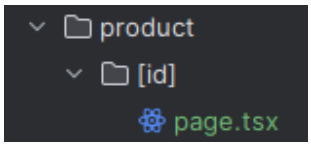
7.2 Klient

7.2.1 Routování

Srozumitelná struktura URL adres reprezentujících jednotlivé stránky webové aplikace je důležitá jak pro uživatele, tak pro SEO [64]. Pro jejich generování jsem využil App Router z Next.js, který nabízí takzvaný file-system based routing, který generuje URL adresy na základě struktury souborového systému [65]. V praxi to znamená, že se URL generují podle struktury vývojářem vytvořených projektových souborů, což velmi zpřehledňuje vývoj.

Pro situace, kdy potřebuji vytvořit unikátní URL pro stránky s identickou strukturou, ale odlišným datovým obsahem, využívám nested routes z APP Routeru. Vytvoření URL díky tomu probíhá na základě dynamicky získaných dat. [65]

Na obrázku 7.2 tuto techniku využívám pro vytvoření unikátní URL pro každou produktovou stránku na základě unikátního identifikátoru produktu.



Obrázek 7.2. Dynamické vytváření URL pomocí App Routeru.

Další součástí App Routeru jsou layouty, pomocí kterých můžeme definovat části uživatelské rozhraní (UI), které budou sdíleny pro všechny podadresáře.

V ukázce 7.7 jsem využil tuto možnost pro nastavení společných komponent Header a Footer pro všechny podadresáře. Navíc jsem ve společné hlavičce nastavil font, který bude stejně jako zbytek layoutu použit napříč celou aplikací, jelikož je tento layout umístěn v kořenovém adresáři. Obsah podadresářů se následně generuje namísto placeholderu {children}.

Pro navigaci napříč URL využívám vestavěnou Next.js komponentu <Link>, rozšiřující HTML tag <a>, její použití demonstruji v ukázce 7.8 pro navigaci z komponenty Footer do dalších subkomponent. Problematiku React komponent podrobněji vysvětlím v kapitole 7.2.2.

```
export default function RootLayout({
                                        children,
                                    }: {
    children: React.ReactNode;
}) {
    return (
        <html lang="en">
        <head>
```

```
            <linkhref="https://fonts.googleapis..."rel="stylesheet"/>
            <title>Viapex</title>
        </head>
        <body>
        <QueryProvider>
            <Header/>
            <main>{children}</main>
            <Footer/>
        </QueryProvider>
        </body>
        </html>
    );
}
```

Ukázka 7.7. Definování sdílené části uživatelského rozhraní pomocí Next.js.

7.2.2 Komponenty

V ukázce 7.7 jsem definoval společnou část UI pomocí tagů <Header/> a <Footer/>, odkazujících na komponenty, které jsem vytvořil v separátních souborech a mohu je tak opakovaně využívat napříč celou aplikací pomocí příslušného tagu. od HTML tagů se liší tím, že začínají vždy velkým písmenem, HTML tagy začínají vždy malým písmenem. React komponenty jsou části UI, které mají vlastní logiku i vzhled. Klíčová slova „export default“ určují hlavní komponentu souboru, samotný obsah komponenty implementuji pomocí jazyka JavaScript XML (JSX), což je jazyk se syntaxí podobnou HTML, který ale mimo jiné umožňuje uvnitř složených závorek psát logiku pomocí Javascriptu. [66]

V ukázce 7.8 demonstruji implementaci komponenty Footer, obsahující nativní Next.js komponenty <Link> odkazující na další URL stránky.

```
export default function Footer() {
    return (
    <footer>
        <div className="...">
            <div>
                <div><Link href="/.../shipping">Delivery...</Link></div>
                <div><Link href="/.../returns">Returns...</Link></div>
                <div><Link href="/.../terms">Terms...</Link></div>
            </div>
            ...
        </div>
    </footer>
    );
}
```

Ukázka 7.8. Implementace React komponenty Footer reprezentující rozcestník.

7.2.3 Dotazování API

Ukázka 7.7 demonstruje také obalení všech vnitřních komponent tagem <QueryProvider>, což je má vlastní komponenta obalující TanStack komponentu QueryClientProvider, dávající celé aplikaci přístup k React Query hooks prostřednictvím komponenty QueryClient. [67] Díky tomu můžou mé vlastní komponenty komunikovat s REST API,

QueryClient se stará také o ukládání dat do cache. Ve své práci využívám pro komunikaci s REST API následující TanStack Query hooks:

- Query - získává data ze serveru, odpovídá HTTP metodě GET. Nabízí stavy isPending, isError a isSuccess, v závislosti na stavu můžeme přistupovat k informacím typu error, data a isFetching.
- Mutation - modifikuje data na serveru, odpovídá HTTP metodám POST, PUT a DELETE. Poskytuje stavy isIdle, isPending, isError a isSuccess. V závislosti na stavu můžeme přistupovat k informacím typu error a data.

V hooku můžeme definovat i chování dle stavu, v jakém se hook nachází a případně pracovat s poskytnutou informací. [68, 69]

Implementaci Query demonstruji v ukázce 7.9, kde ji využívám pro načtení objednávek aktuálně přihlášeného uživatele, podrobnější popis je vysvětlen přímo v kódu pomocí komentářů.

V ukázce 7.10 hook useUserOrdersQuery využívám pro vypsání objednávek, pracuji zde také se stavem, který Query poskytuje. Na základě získaného stavu informuji uživatele o probíhajícím načítání či případné chybě.

```
export const useUserOrdersQuery = () => {
    //Načtení JWT ze stavu aplikace pomocí knihovny Zustand
    const token = useAuthStore((s) => s.token);

    return useQuery<OrderResponse[]>({
        queryKey: ['userOrders'],
        queryFn: async () => {

            //Zpracování chyby při neexistenci tokenu
            if (!token) {
                throw new Error('User is not authenticated');
            }

            //Odeslání HTTP dotazu metodou GET a připojení
            //hlavičky s JWT tokenem
            const res = await fetch(`${BASE_URL}.../user`, {
                method: 'GET',
                headers: {
                    Authorization: `Bearer ${token}`,
                },
            });

            //Zpracování případné chyby
            if (!res.ok) {...}

            //Parsování JSON odpovědi a zpracování případné chyby
            try {...}
        },
    });
};
```

Ukázka 7.9. Ukázka implementace TanStack Query useUserOrdersQuery pro získání objednávek aktuálně přihlášeného zákazníka.

```
const UserOrders = () => {
    const { data, isLoading, isError, error } = useUserOrdersQuery();

    if (isLoading) return <p>Loading orders…</p>;
    if (isError) return <p>Error: {(error as Error).message}</p>;

    return (
        <div className="w-[800px]">
            <h2 className="text-2xl font-bold mb-6">My Orders</h2>

            {data?.map((order) => (
                //Zobrazení objednávek pomocí jazyka JSX
            ))}
        </div>
    );
};
```

Ukázka 7.10. Použití Query useUserOrdersQuery pro načtení objednávek přihlášeného zákazníka, následná definice chování na základě stavu query a případné zobrazení objednávek.

Kapitola 8
Uživatelské testování

Pomocí uživatelského testování ověřujeme, zda reálný uživatel neobeznámený se systémem intuitivně zvládne jeho ovládání. Děje se tak sice v simulovaných podmínkách, ale na reálných situacích. [70] Vybral jsem tedy testovací uživatele z řad studentů a mé rodiny, zadal jim seznam úkolů, jež mají provést a během plnění jsem sbíral jejich zpětnou vazbu. Pro simulování podmínek jsem naplnil e-shop testovacími daty.

8.1 Profil testovacích uživatelů

- Tester 1: Student programu Softwarové inženýrství a technologie na ČVUT FEL, věková skupina 20 - 30 let.
- Tester 2: Student programu Softwarové inženýrství a technologie na ČVUT FEL, věková skupina 20 - 30 let.
- Tester 3: Osoba, jež není technicky založena, věková skupina 20 - 30 let.
- Tester 4: Osoba, jež není technicky založena, věková skupina 50 - 60 let.

8.2 Scénář

Testovací scénář jsem sestavil jako sekvenci úkolů, které společně pokrývají funkční požadavky definované zákazníka v kapitole 3.3.1.

1. Prostuduj možnosti dopravy a platby.
2. Prostudujte reklamační řád.
3. Registrujte se.
4. Přihlaste se.
5. Objednejte si libovolné dva kusy oblečení.
6. Zobrazte si historii objednávek.
7. Odhlaste se.

8.3 Sbíraná data

- Čas: Doba potřebná ke splnění scénáře. (minuty:vteřiny)
- Zpětná vazba: Zpětná vazba uživatele.
- Poznatky: Poznatky organizátora testu.

8.4 Výsledky testování

8.4.1 Tester 1

- Čas: 4:14
- Zpětná vazba:
  - Indikovat úspěšné přidání produktu do košíku. (přidáno)
  - Přidat možnost prokliku na produktovou stránku ze seznamu obsahu objednávky. (v budoucnu)
  - Přesměrování z uživatelského profilu po odhlášení. (přidáno)
  - Přidat změnu kurzoru při najetí na tlačítko změny jazykové mutace podmínek. (přidáno)
- Poznatky: Jelikož je Tester 1 do jisté míry znalý v oboru softwarového vývoje, snažil se hledat i drobné nedostatky a dát co nejdetailnější zpětnou vazbu, což se promítlo i poměrně dlouhé doby provedení testu. Tester měl problém rozpoznat, zda byl produkt opravdu přidán do košíku, jinak scénářem prošel bez problémů.

8.4.2 Tester 2

- Čas: 3:42
- Zpětná vazba:
  - Indikovat úspěšné přidání produktu do košíku. (přidáno)
  - Přidat přesměrování z uživatelského profilu po odhlášení. (přidáno)
  - Přidat přihlášení zároveň s provedením registrace. (zvážím do budoucna)
  - Přidat do hlavičky stránky i tlačítko pro registraci, aby nebylo nutné registrační stránku navštěvovat přes přihlašovací stránku. (pravděpodobně neimplementuji)
- Poznatky: Tester 2 se snažil dát podrobnou technickou zpětnou vazbu, což lze přičíst jeho odbornému zaměření. Měl problém rozpoznat, zda byl produkt opravdu přidán do košíku, jinak scénářem prošel bez problémů.

8.4.3 Tester 3

- Čas: 3:10
- Zpětná vazba:
  - Indikovat úspěšné přidání produktu do košíku. (přidáno)
- Poznatky: Tester 3 měl problém rozpoznat, zda byl produkt opravdu přidán do košíku, jinak scénářem prošel bez problému.

8.4.4 Tester 4

- Čas: 3:27
- Zpětná vazba:

  - Uvítal by kompletní jazykovou mutaci. (v budoucnu)
  - Indikovat úspěšné přidání produktu do košíku. (přidáno)
- Poznatky: Tester 4 měl problém rozpoznat, zda byl produkt opravdu přidán do košíku, jinak scénářem prošel bez problému.

8.5 Vyhodnocení

Všichni testovací uživatelé zvládli projít scénářem bez zásadních problémů, drobné nedostatky indikované testováním jsem již opravil, zbylé budou řešeny v budoucnu.

Kapitola 9
Závěr

Cílem této práce bylo zmapovat problematiku online prodeje outdoorového vybavení a provést sběr požadavků pro e-shop. Následně prozkoumat možné využití existujících e-commerce řešení a porovnat je s implementací vlastního řešení se zaměřením na finanční stránku, udržitelnost a rozšiřitelnost. Na základě průzkumu definovat klíčové požadavky, které musí e-shop splňovat, přičemž by se mělo jednat minimálně o procházení, správu a administraci produktů a správu a vytváření objednávek. Dále jsem měl navrhnout řešení, které odpovídá definovaným požadavkům a definovat MVP, které je jádrem takového systému. Navržené MVP jsem měl implementovat a podrobit uživatelskému testování.

Problematiku online prodeje outdoorového vybavení jsem zmapoval v kapitole 2, z průzkumu finančního stavu trhu vyplynula jeho rostoucí tendence a přeliv zákazníků z kamenných prodejen do e-shopů. Také jsem se zde věnoval vlivu formy online prezentace na chování zákazníků, na nové tržní segmenty a postavení výrobců outdoorového vybavení na trhu. Sběr požadavků a definici cílů projektu jsem provedl v kapitole 3.

Možnosti využití existujícího e-commerce řešení jsem prozkoumal v kapitole 4, zvažoval jsem dvě SaaS řešení a jedno open-source řešení. WooCommerce jsem jako open-source řešení zamítl zejména z důvodu bezpečnostních rizik a mé nulové zkušenosti s danou technologií. Shopify a Shoptet jsem v porovnání s vlastní implementací zamítl na základě nižší rozšiřitelnosti a větším finančním nárokům na provoz. Klíčové požadavky jsem definoval pomocí výběru **primárních** požadavků v kapitole 3, včetně požadavků definujících funkcionality pro procházení, správu a administraci produktů a správu a vytváření objednávek.

Na základě cílů a **primárních** požadavků jsem v kapitole 5 navrhl jádro e-shopu pro prodej outdoorového oblečení. S ohledem na budoucí rozšiřitelnost a výkonnostní škálovatelnost systému jsem serverovou část rozdělil do třech mikroslužeb, jejichž API a cross-cutting concers jsem centralizoval do API gateway. Dále jsem navrhl integraci platební brány Stripe a před API gateway vystavil web server pro server side rendering. Součástí návrhu je i katalog API a wireframeů obrazovek klientské části

Pro implementaci návrhu jsem v kapitole 6 vybral vhodné technologie, pomocí kterých jsem v kapitole 7 navržené řešení implementoval.

V kapitole 8 jsem implementovaný e-shop podrobil uživatelskému testování, během kterého byly identifikovány drobné nedostatky, které jsem opravil.

Práci bych rád dále rozvíjel, zaměřit se plánuji na implementaci volitelných požadavků definovaných v kapitole 3.1.

Literatura

[1] Chris Lines. *New trade data released by the European Outdoor Group reveals growth in 2022*. 2023-06-04. https://www.europeanoutdoorgroup.com/articles/new-trade-data-released-by-the-european-outdoor-group-reveals-growth-in-2022. Navštíveno: 2025-05-21.

[2] Stefan Ludwig, Felix Mutter, Christian Rump, Tatjana Lietz a Tim Seibert. *Outdoor Consumer Report 2021 Delloitte OutDoor by ISPO*. 2021-10. https://assets-global.website-files.com/640056047efc0c0d56b83be0/6481dac24dc3bbc4369dc627_Outdoor_Consumer_Report_2021.pdf. Navštíveno: 2025-05-21.

[3] Cathrine Lambert. *Case Study: How Swanky's Strategic Digital Marketing Services Helped DockATot Navigate an Ecommerce Rebrand*. 2021-08-27. https://swankyagency.com/ecommerce-rebrand-digital-marketing-case-study-dockatot. Navštíveno: 2025-05-21.

[4] Tyler Fox. *The Pacific Crest Trail Gear Guide (2023 Survey)*. 2025-02-14. https://www.halfwayanywhere.com/trails/pacific-crest-trail/pct-gear-guide-2023. Navštíveno: 2025-05-20.

[5] Dan Durston. *Reddit post*. 2023-03-23. https://www.reddit.com/r/DurstonGearheads/comments/xio4qk/comment/jdem1t5/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button. Navštíveno: 2025-05-21.

[6] *Kurzy devizového trhu*. https://www.cnb.cz/cs/financni-trhy/devizovy-trh/kurzy-devizoveho-trhu/kurzy-devizoveho-trhu/. Navštíveno: 2025-05-15.

[7] Michael Bugaj. *The most popular commerce platforms across ten European markets*. 2024-09-12. https://www.tembi.io/blog/commerce-platforms-european-webshops. Navštíveno: 2025-05-20.

[8] *shopify app store*. https://apps.shopify.com/?locale=cs. Navštíveno: 2025-05-20.

[9] Shivbhadrasinh Gohil. *Shopify App Store Statistics: 2025 Report by Meetanshi*. 2025-05-19. https://meetanshi.com/blog/shopify-app-store-statistics/. Navštíveno: 2025-05-20.

[10] *SEO On: AI Product Description*. https://apps.shopify.com/ai-product-copy?locale=cs&surface_detail=trendin. Navštíveno: 2025-05-20.

[11] *Browse all themes*. https://themes.shopify.com/themes?sort_by=popularity. Navštíveno: 2025--05-20.

[12] *Plány a ceny*.
https://www.shopify.com/cz/pricing. Navštíveno: 2025-05-20.

[13] Mandy Pardehpoosh. *Introducing Markets: A Unified Home for Business Expansion*. 2024-06-24.
https://www.shopify.com/blog/markets. Navštíveno: 2025-05-20.

[14] Maxim Drkoš. *Vše o nově dostupných Shopify Payments a Markets*. 2024-01-02.
https://cz.digismoothie.com/blog/vse-o-shopify-payments-a-markets. Navštíveno: 2025-05-20.

[15] *Fees and costs*.
https://help.shopify.com/en/manual/international/pricing/fees. Navštíveno: 2025-05-20.

[16] Albert Mosby. *Shopify Market Share 2025 (Data, Insights Statisctics)*. 2025-05--03.
https://www.yaguara.co/shopify-market-share. Navštíveno: 2025-05-20.

[17] *WooCommerce*.
https://wordpress.org/plugins/woocommerce/. Navštíveno: 2025-05-20.

[18] *WooCommerce Marketplace*.
https://woocommerce.com/product-category/woocommerce-extensions/?collections=product&page=1. Navštíveno: 2025-05-20.

[19] *Online payments pricing*.
https://www.comgate.cz/en/online-payments-pricing. Navštíveno: 2025-05--20.

[20] Maddy Osman. *Wild and Interesting WordPress Statistics and Facts*. 2024-08-26.
https://kinsta.com/blog/wordpress-statistics/. Navštíveno: 2025-05-20.

[21] Ben Martin, Cesar Anjos, Denis Sinegubko, Rodrigo Escobar, Tiago Pellegrini a Rianna MacLeod. *2023 Hacked Website Malware Threat Report*.
https://sucuri.net/reports/2023-hacked-website-report/. Navštíveno: 2025-05-20.

[22] Joel Olawanle. *19 steps to protect your WordPress site from threats*. 2025-04-04.
https://kinsta.com/blog/wordpress-security/. Navštíveno: 2025-05-20.

[23] *Český e-commerce trh*.
https://www.ceska-ecommerce.cz/. Navštíveno: 2025-05-20.

[24] *PCI DSS Compliance Requirements Guide Checklist*. 2024-02-23.
https://sucuri.net/guides/pci-compliance-requirements-checklist/. Navštíveno: 2025-05-20.

[25] *The State of Shoptet in 2025*. 2025-05-16.
https://storeleads.app/reports/shoptet. Navštíveno: 2025-05-20.

[26] *Shoptet Doplňky*.
https://doplnky.shoptet.cz/. Navštíveno: 2025-05-20.

[27] *Comgate*.
https://doplnky.shoptet.cz/comgate. Navštíveno: 2025-05-20.

[28] *Ceník*.
https://www.shoptet.cz/cenik/. Navštíveno: 2025-05-20.

[29] *Shoptet Premium*.
https://www.shoptetpremium.cz/cena/. Navštíveno: 2025-05-20.

[30] *Shoptet API (1.0.0).*
`https://api.docs.shoptet.com/shoptet-api/openapi`. Navštíveno: 2025-05--20.

[31] *Google Cloud's pricing calculator.*
`https://cloud.google.com/products/calculator`. Navštíveno: 2025-05-20.

[32] Gary Woodfine. *Using the Repository and Unit Of Work Pattern in .net core.* 2018-01-08.
`https://garywoodfine.com/generic-repository-pattern-net-core/`. Navštíveno: 2025-05-20.

[33] *How Checkout works.*
`https://docs.stripe.com/payments/checkout/how-checkout-works`. Navštíveno: 2025-05-21.

[34] *What is wireframing?*
`https://www.figma.com/resource-library/what-is-wireframing/`. Navštíveno: 2025-05-20.

[35] *Gitflow workflow.*
`https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`. Navštíveno: 2025-05-20.

[36] *What is PostgreSQL?*
`https://aws.amazon.com/rds/postgresql/what-is-postgresql/`. Navštíveno: 2025-05-20.

[37] *PYPL PopularitY of Programming Language.* 2025-05.
`https://pypl.github.io/PYPL.html`. Navštíveno: 2025-05-20.

[38] *Advantages and Disadvantages of Java.* 2024-06-10.
`https://www.geeksforgeeks.org/advantages-and-disadvantages-of-java/`. Navštíveno: 2025-05-20.

[39] *Microservices.*
`https://www.jetbrains.com/lp/devecosystem-2022/microservices/`. Navštíveno: 2025-05-20.

[40] Craig Walls. *Spring Boot in action.* Simon and Schuster, 2015.

[41] Rod Johnson, Juergen Hoeller, Alef Arendsen a others. *Professional Java development with the Spring framework.* John Wiley & Sons, 2009.

[42] *What is REST API?* 2025-04-24.
`https://www.ibm.com/think/topics/rest-apis`. Navštíveno: 2025-05-20.

[43] *What is RESTful API?*
`https://aws.amazon.com/what-is/restful-api/`. Navštíveno: 2025-05-20.

[44] *What is a REST API?* 2020-08-05.
`https://www.redhat.com/en/topics/api/what-is-a-rest-api`. Navštíveno: 2025-05-20.

[45] *Best practices for ecommerce sites in Google Search.* 2025-02-04.
`https://developers.google.com/search/docs/specialty/ecommerce`. Navštíveno: 2025-05-20.

[46] *Static and Dynamic Rendering.*
`https://nextjs.org/learn/dashboard-app/static-and-dynamic-rendering`. Navštíveno: 2025-05-20.

[47] *What is Next.js?*
https://nextjs.org/docs. Navštíveno: 2025-05-20.

[48] *Object/Relational Mapping*.
https://hibernate.org/orm/. Navštíveno: 2025-05-20.

[49] Luca Cambi a Eric Martin. *Difference Between JPA and Spring Data JPA*. 2024--01-08.
https://www.baeldung.com/spring-data-jpa-vs-jpa. Navštíveno: 2025-05-20.

[50] Shubham Aggarwal a Carsten. *CrudRepository*. 2025-04-03.
CrudRepository, JpaRepository, and PagingAndSortingRepository in Spring Data. Navštíveno: 2025-05-20.

[51] Zeger Hendrikse. *Http Message Converters with the Spring Framework*. 2024-05--11.
https://www.baeldung.com/spring-httpmessageconverter-rest. Navštíveno: 2025-05-20.

[52] *HTTP Message Conversion*.
https://docs.spring.io/spring-framework/reference/web/webmvc/message-converters.html. Navštíveno: 2025-05-20.

[53] *Annotation Type NotBlank*.
https://jakarta.ee/specifications/bean-validation/2.0/apidocs/?javax/validation/constraints/NotBlank.html. Navštíveno: 2025-05-21.

[54] *Annotation Type Email*.
https://jakarta.ee/specifications/platform/9/apidocs/?jakarta/validation/constraints/Email.html. Navštíveno: 2025-05-21.

[55] *springdoc-openapi v2.8.8*.
https://springdoc.org/. Navštíveno: 2025-05-21.

[56] Kevin Gilmore. *Documenting a Spring REST API Using OpenAPI 3.0*. 2025-04--08.
https://www.baeldung.com/spring-rest-openapi-documentation. Navštíveno: 2025-05-21.

[57] *MapStruct*.
https://mapstruct.org/. Navštíveno: 2025-05-20.

[58] Grzegorz Piwowarek. *Quick Guide to MapStruct*. 2024-01-18.
https://www.baeldung.com/mapstruct. Navštíveno: 2025-05-20.

[59] *Lombok features*.
https://projectlombok.org/features/. Navštíveno: 2025-05-20.

[60] *Introduction to Project Lombok*. 2025-03-17.
https://www.baeldung.com/intro-to-project-lombok. Navštíveno: 2025-05--20.

[61] *Spring Cloud Gateway*.
https://spring.io/projects/spring-cloud-gateway. Navštíveno: 2025-05-21.

[62] *Spring Cloud Gateway with Spring WebFlux*. 2024-06-07.
https://www.geeksforgeeks.org/spring-cloud-gateway-with-spring-webflux. Navštíveno: 2025-05-21.

[63] *Spring WebFlux*.
https://docs.spring.io/spring-framework/reference/web/webflux.html. Navštíveno: 2025-05-21.

[64] *URL structure best practices for Google*. 2025-03-17. https://developers.google.com/search/docs/crawling-indexing/url-structure. Navštíveno: 2025-05-20.

[65] *How to create layouts and pages*. https://nextjs.org/docs/app/getting-started/layouts-and-pages. Navštíveno: 2025-05-20.

[66] *Quick Start*. https://react.dev/learn. Navštíveno: 2025-05-20.

[67] *QueryClientProvider*. https://tanstack.com/query/latest/docs/framework/react/reference/QueryClientProvider. Navštíveno: 2025-05-20.

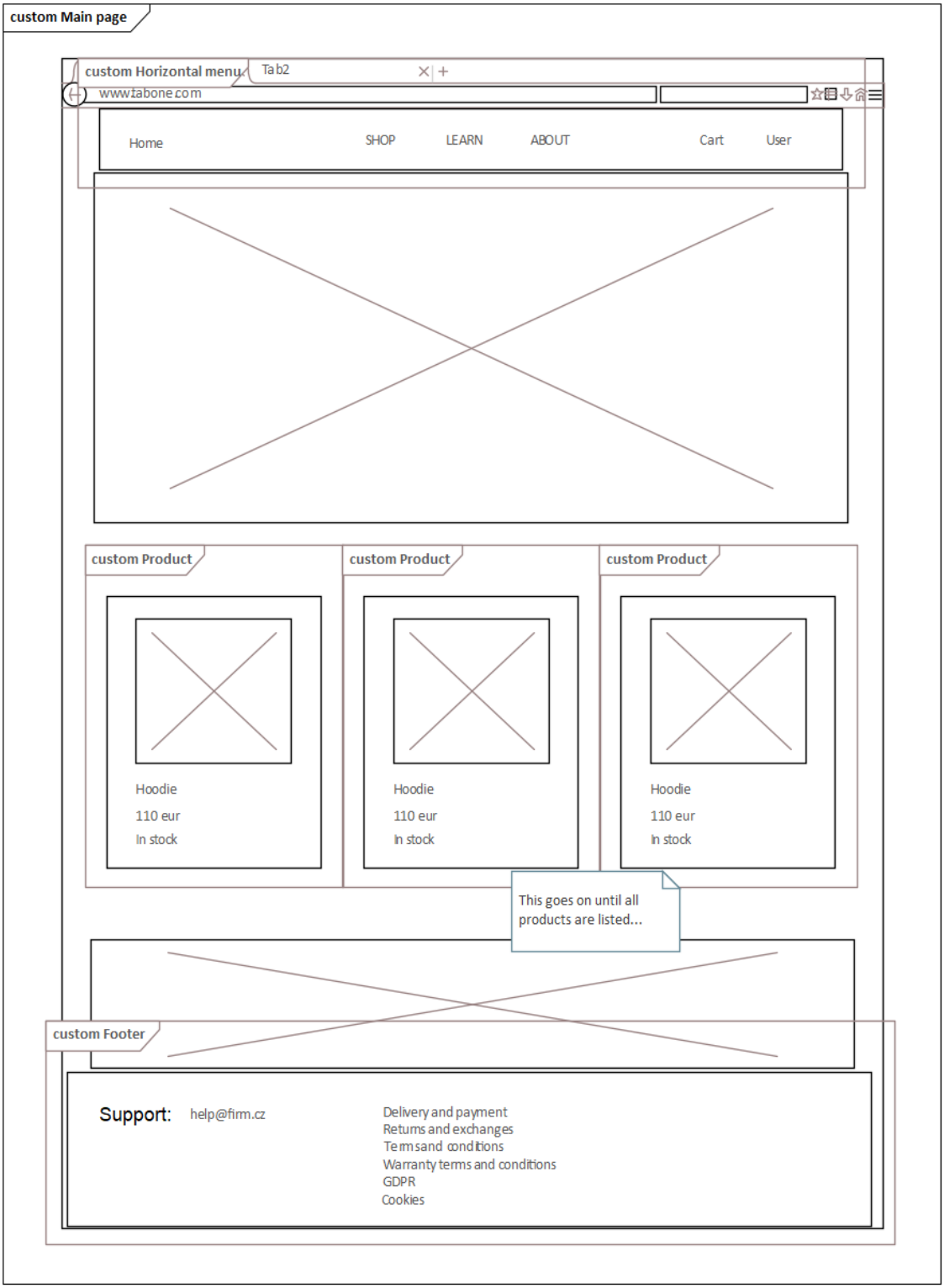
[68] *Queries*. https://tanstack.com/query/latest/docs/framework/react/guides/queries. Navštíveno: 2025-05-20.

[69] *mutation*. https://tanstack.com/query/latest/docs/framework/react/guides/mutations. Navštíveno: 2025-05-20.

[70] Michal Mikolaj. *Usability testing aneb Jak na uživatelské testování použitelnosti*. 2020-08-20. https://www.ackee.cz/blog/uzivatelske-testovani-pouzitelnosti. Navštíveno: 2025-05-21.

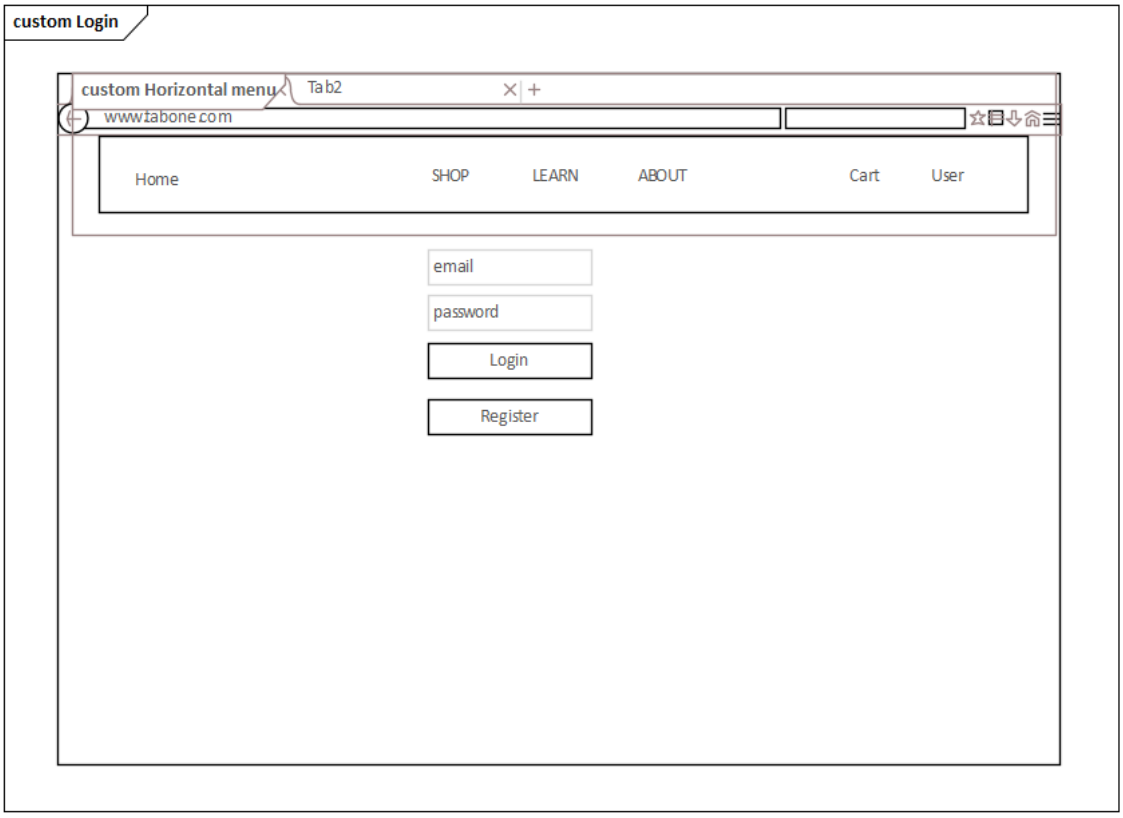
[71] *Stripe Demo*. https://checkout.stripe.dev/checkout. Navštíveno: 2025-05-20.

Příloha A
Wire-frames

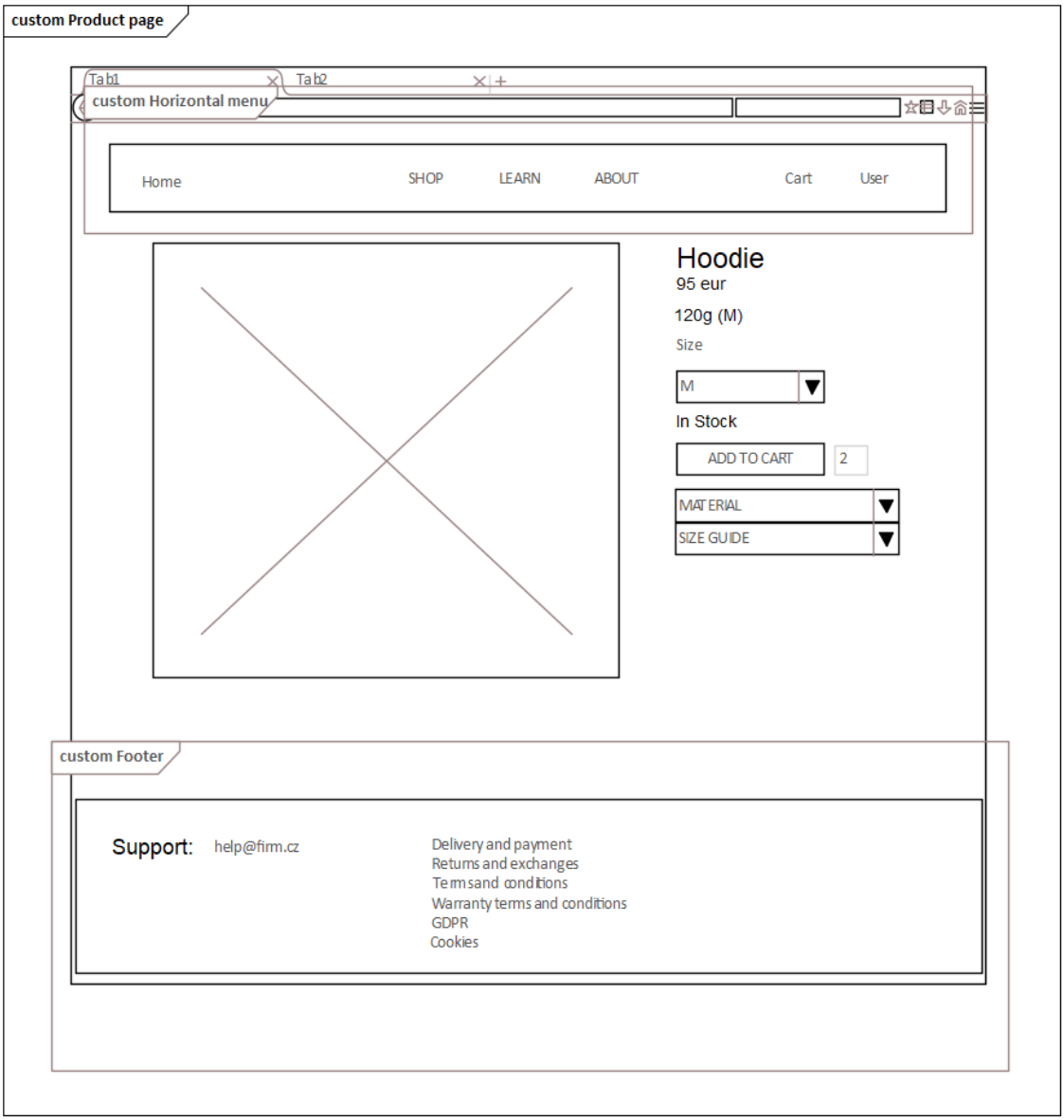


Obrázek A.1. Wire-frame domovské stránky.

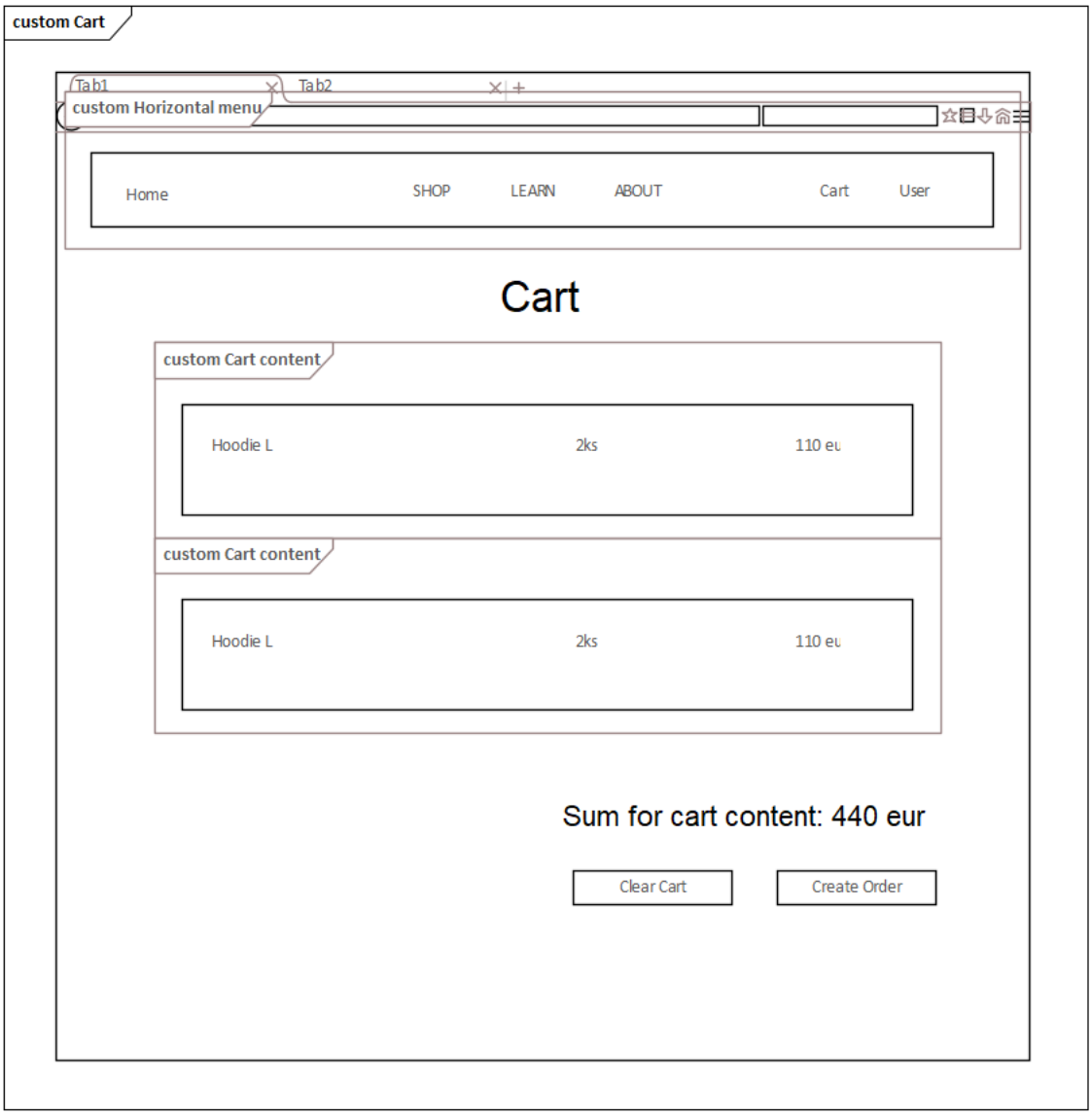
Obrázek A.2. Wire-frame regsitrační stránky.



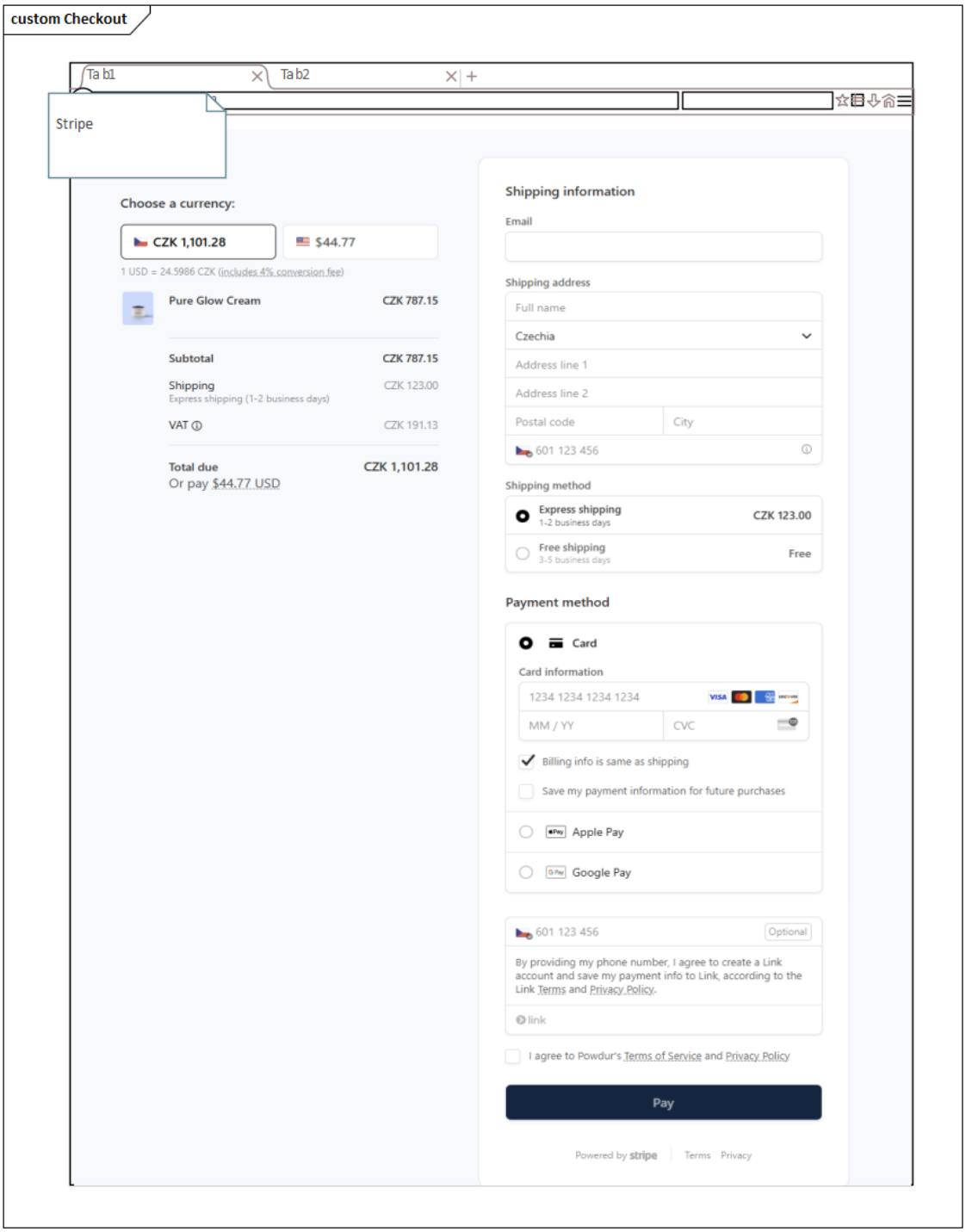
Obrázek A.3. Wire-frame přihlašovací stránky.



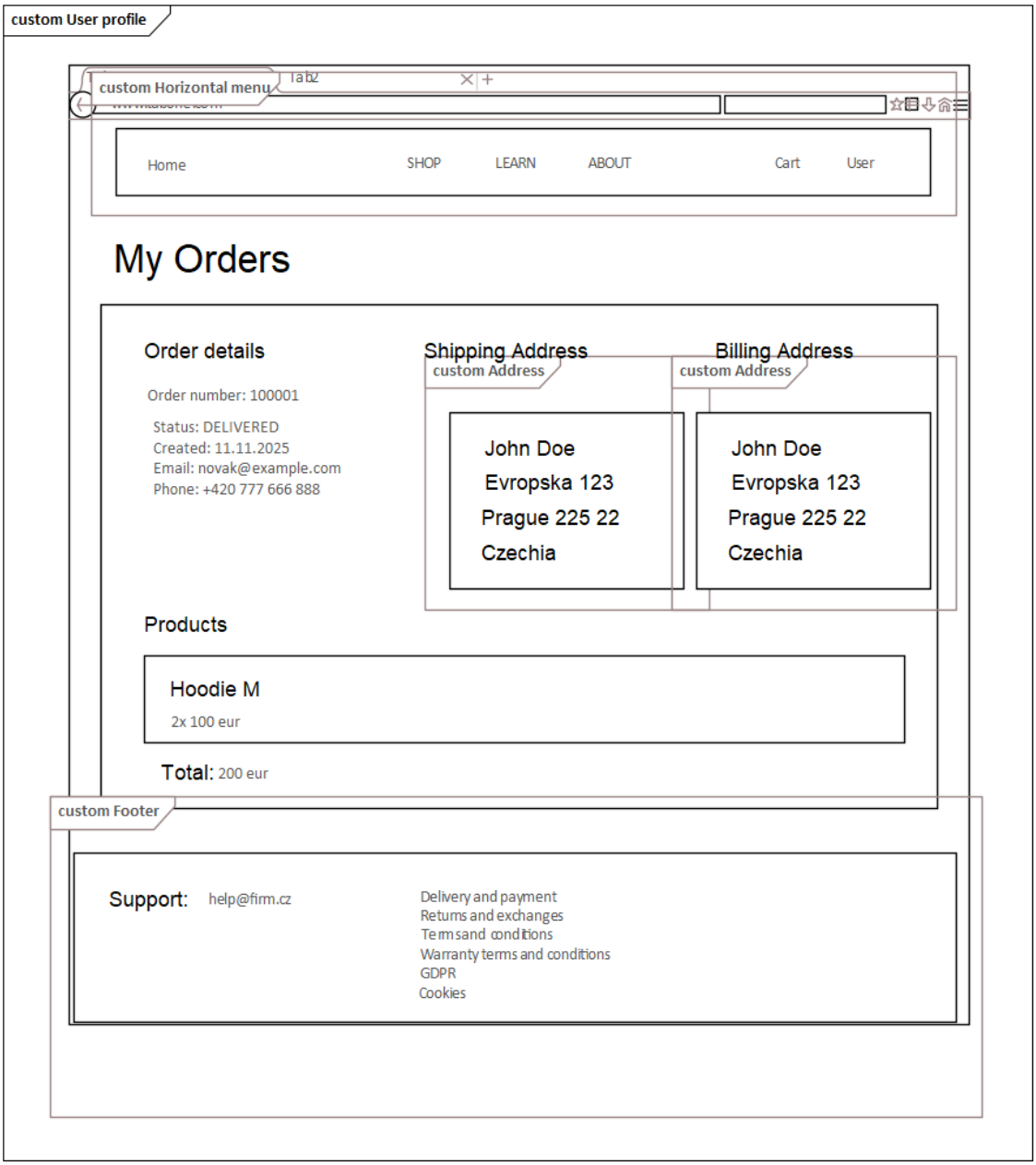
Obrázek A.4. Wire-frame produktové stránky.



Obrázek A.5. Wire-frame stránky obsahu košíku.



Obrázek A.6. Wire-frame stránky pro tvorbu objednávky, jedná se o snímek obrazkovky ze Stripe Demo. [71]



Obrázek A.7. Wire-frame stránky zákaznického profilu.