Objektový návrh knihkupectví

Dominik Žárský

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik Žárský**
Osobní číslo:	**A18031**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační technologie v administrativě**
Forma studia:	**Prezenční**
Téma práce:	**Objektový návrh knihkupectví**
Téma práce anglicky:	**An Object-Oriented Design of a Bookstore**

Zásady pro vypracování

1. Analyzujte funkcionalitu internetových knihkupectví.
2. Proveďte analýzu požadavků aplikace pro prodej fyzických knih.
3. Připravte návrh aplikace.
4. Realizujte prototyp aplikace.
5. Navrhněte možný rozvoj aplikace.

Forma zpracování bakalářské práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. J. Arlow and I. Neustadt, UML 2 and the unified process : practical object-oriented analysis and design, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2005, pp. xxiii, 592 p.
2. I. KRAVAL, Analytické modelování informačních systémů pomocí UML v praxi. 2010, ISBN 978-80-254-6986-6.
3. D. Pilone and N. Pitman, UML 2.0 in a nutshell, 1st ed. Beijing ; Sebastopol, CA: O'Reilly Media, 2005, pp. xv, 222 p.
4. H. Podeswa, UML for the IT business analyst : a practical guide to object-oriented requirements gathering, 2nd ed. Australia ; United States: Course Technology/Cengage Learning, 2010, pp. xxv, 372 p.
5. M. Seidl, *UML @ classroom : an introduction to object-oriented modeling*. New York, NY: Springer Berlin Heidelberg, 2015

Vedoucí bakalářské práce: **doc. Ing. Radek Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **19. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Martin Sysel, Ph.D. v.r.
garant oboru

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 31. 5. 2021 Dominik Žárský v.r.

ABSTRAKT

Tato bakalářská práce se bude zabývat objektovým návrhem elektronického knihkupectví. Součástí práce bude především vytvoření samotného objektového návrhu za použití jazyka UML a vytvoření prototypu této aplikace pomocí technologií pro tvorbu webových aplikací. Probrány budou v krátkosti i možnosti rozšíření navržené aplikace o doplňkovou funkcionalitu. Tyto součásti budou taktéž doplněny o teorii v oblasti jazyka UML a použitých webových technologií. Práce bude taktéž obsahovat rozbor aktuálního stavu na trhu elektronických knihkupectví, tedy používané technologie, obecná funkcionalita a příklady aktuálně fungujících knihkupectví.

Klíčová slova: knihkupectví, elektronický obchod, UML, objektový návrh, Laravel, webová aplikace

ABSTRACT

This bachelor thesis is going to focus on object-oriented design of a bookstore. The thesis will primarily comprise the creation of such object-oriented design using the Unified Modeling Language (UML) as well as creation of a prototype of this application using standard technologies for web application development. Options of expanding the designed application further will be touched upon as well. The aformentioned parts will also be amended by a theoretical part describing the UML and the technologies used for creating the application prototype. The thesis will also include an analysis of the current state of the electronic bookstore market, for example of used technologies, their general functionality and examples of current electronic bookstores.

Keywords: bookstore, e-shop, UML, object-oriented design, Laravel, web application

Rád bych poděkoval doc. Ing. Radkovi Šilhavému, PhD. za vedení této práce a rady i připomínky v průběhu jejího vypracování.

Andreas šel lovit lva, ale sám byl svou kořistí sežrán…

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Elektronická knihkupectví se v posledních letech stávají velmi oblíbenými místem pro nákup jak fyzických, tak například i elektronických knih a do budoucna by mohly jistě mohly určité míře nahradit fyzické obchody s knihami. Hlavním důvodem je rychlost, a hlavně jednoduchost nákupu v takovémto knihkupectví. Zatímco při nákupu knihy v kamenné prodejně se člověk musí do této prodejny dopravit, na místě knihu manuálně nalézt a zakoupit, v případě elektronického nákupu je kroků v tomto procesu značně méně. Zákazník otevře web knihkupectví ve svém webovém prohlížeči, v krátkém čase nalezne knihu podle názvu či autora, vše elektronicky zaplatí a zboží je mu do několika dnů dovezeno přímo domů. Aby však tento proces bez chyb fungoval tak, jak byl nyní popsán, musí být ono internetové knihkupectví kvalitně navrženo a naprogramováno. A právě těmito úkony se bude tato bakalářská práce zabývat.

V první části práce bude zpracována analýza současné situace na trhu elektronických knihkupectví v České republice i v zahraničí, ta bude obnášet krátké zhodnocení situace světových i tuzemských elektronických knihkupectví, porovnáním možných způsobů jejich implementace či jejich základní funkcionality. Dále se bude teoretická část práce zabývat problematikou jazyka UML při návrhu a vývoji softwaru, budou popsány různé součásti jazyka UML, včetně diagramů, které budou posléze užívány v praktické části k návrhu samotné vlastní aplikace.

Následně bude proveden vlastní návrh knihkupectví zaměřujícího se na prodej fyzických knih, ten bude vypracován za použití jazyka UML a dalších nástrojů pro softwarovou analýzu za účelem přípravy podkladu pro tvorbu vlastního softwarového řešení elektronického knihkupectví. Výstupem objektového návrhu budou UML diagramy zahrnující například analýzu funkčních a nefunkčních požadavků., modely případů užití aplikace, a podobně. Podle tohoto návrhu bude následně vytvořen prototyp elektronického knihkupectví, jakožto ukázka možnosti implementace zcela vlastního řešení postaveného na PHP frameworku Laravel s použitím dalších technologií, jako je HTML5, CSS3, JavaScript pro vývoj front-endu aplikace a SQL databáze pro uložení aplikačních dat.

I. TEORETICKÁ ČÁST

1 SOUČASNÁ ŘEŠENÍ NA TRHU INTERNETOVÝCH KNIHKUPECTVÍ

Elektronická knihkupectví jsou v dnešní době již široce rozšířena a hojně užívána nejen v České republice, ale i v zahraničí. Mnoho předních knihkupeckých sítí provozuje kromě kamenných prodejen i webové systémy pro prodej knih, a to jak fyzických, tak i elektronických knih.

1.1 Používané technologie a systémy

Technologie a systémy používané pro provoz elektronických knihkupectví se v zásadě mnohdy neliší od systémů užívaných klasickými internetovými obchody (e-shopy) prodávajícími například oblečení, elektroniku či jiné zboží. Ačkoliv neexistuje mnoho e-commerce systémů vytvořených přímo pro potřeby provozu elektronických knihkupectví, mnoho z nabízených řešení je možné upravit či do nich nainstalovat dodatečné moduly za účelem vytvoření kompetentní platformy, která by k tomuto účelu dobře sloužila.

Některé systémy navržené přímo pro provoz internetových knihkupectví mohou být provozovány v modelu Software-as-a-Service (SaaS), tedy poskytované jejich výrobcem jakožto placená služba. Příkladem takovýchto služeb mohou být v zahraničí zejména systémy IndieCommerce a IndieLite provozované Americkou asociací prodejců knih (American Booksellers Association). Tyto systémy jsou v jádru prakticky totožné, ovšem Lite verze postrádá nějakou část funkcionality plné verze systému a je určena převážně pro menší knihkupectví. Tyto platformy umožňují prodávat jak fyzické knihy, tak i e-knihy a audioknihy. Hojně jsou využívány například ve Spojených státech.[1]

Jak již bylo zmíněno, k vytvoření elektronického knihkupectví lze také použít klasické e-commerce řešení, což jsou například systémy jako Magento, Shopware, Shopify nebo například PrestaShop. Existuje jich však mnohem více a mnohdy jsou poskytovány v mnoha variantách. Tyto systémy zpravidla bývají modulární a lze je rozšířit o moduly a pluginy, které jim dodají určitou funkcionalitu. Některé řešení jsou také open-source (například již zmiňovaný PrestaShop), takže je možné pracovat přímo se zdrojovým kódem a upravit si e-shop pro potřeby prodeje knih přímo.[2]

Další možností je využití klasických Content Management Systémů (CMS) jako je WordPress, Drupal či Joomla. Do takovýchto systémů lze rovněž instalovat rozšíření, která

jim přidají funkce elektronických obchodů. Příkladem takového rozšíření je například WooCommerce, e-commerce plugin vytvořený pro CMS WordPress.[2]

Poslední z možností je vytvoření kompletně nového a vlastního systému pro prodej knih. Využití této varianty může být výhodné, protože je zde možno lépe reflektovat specifické potřeby knihkupectví zadávajícího požadavek na systém, alespoň co se funkcionality webové aplikace týče. Nevýhodou užití této varianty je větší nákladnost a časová náročnost na implementaci.[3]

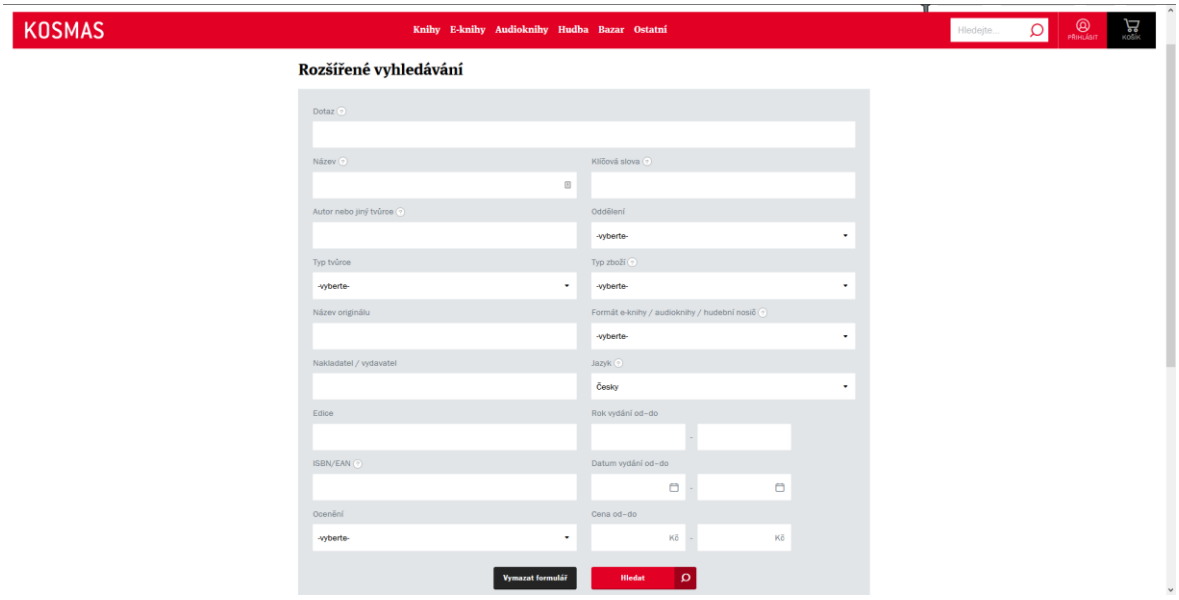
K naprogramování vlastního elektronického knihkupectví se většinou využívají klasické webové technologie, kterými jsou například vytvořeny i již zmíněné dříve e-commerce systémy. Na front-endu to bývá HTML, CSS a JavaScript (většinou za pomocí frameworku jako je React nebo Vue), na back-endu poté některý ze server-side programovacích jazyků a SQL databáze. K účelu programování back-endu takovýchto webových aplikací jsou nejčastěji využívány jazyky PHP, Python, Ruby či C#. Tyto jazyky nabízejí také mnoho frameworků, jako jsou kupříkladu Symfony či Laravel pro PHP, Flask a Django pro Python nebo ASP.NET pro C#. Využití takovýchto frameworků tvorbu back-endu aplikace značně ulehčuje.[4]

1.2 Analýza funkcionality internetových knihkupectví

Stejně jako jiné e-commerce systémy, i elektronická knihkupectví obsahují mnoho standardní funkcionality, která se v takovýchto systémech objevuje. To se může týkat zejména funkcí vyhledávání knih dle různých parametrů, tzv. *košíku*, zpracování plateb, ať už platební kartou nebo jinou platební metodou, možnost založení účtu uživatelem, a podobně.

1.2.1 Vyhledávání

Možnost vyhledávání knih z databáze internetového knihkupectví je jednou z mnoha výhod tohoto typu knihkupectví oproti kamenné prodejně. Některá elektronická knihkupectví nabízejí jednoduché vyhledávání, které podle zadaného výrazu prohledá celý katalog knih, hledajíce shodu, ale mohou nabízet například i rozšířené vyhledávání. V rámci takového vyhledávání je možno hledat například podle kategorie, do níž se kniha řadí (například historické či odborné knihy), ale i podle autora, roku vydání, ISBN či ceny. Na Obrázku 1 je zobrazen příklad takového vyhledávání z webových stránek internetového knihkupectví Kosmas.[5]



Obrázek 1 – Příklad funkce *Rozšířené vyhledávání*

1.2.2 Košík

Funkce košíku je nepochybně jednou z nejdůležitějších funkcí v e-commerce systémech obecně.[6]

Úloha košíku v rámci elektronického knihkupectví je v základu velmi jednoduchá. Je jí udržování informací o produktech, které chce zákazník nakoupit a následně i zpracování celé objednávky ve spolupráci s použitým objednávkovým systémem. V souvislosti s touto skutečností může pak košík udržovat i informace o zákazníkovi. Takovými informacemi mohou být například jméno a příjmení, fakturační adresa nebo údaje o platební kartě. Zákazník zde může také upravovat množství objednaných produktů nebo produkty z košíku odstraňovat.[6][7]

1.2.3 Katalog knih

Hlavní strana většiny internetových knihkupectví zobrazuje katalog knih. Katalog tradičně poskytuje informace o knihách, mezi nimi například jméno autora knihy, rok vydání, kategorii, cena, a podobně. Podle těchto informací je také možné položky katalogu rozdělit do podsekcí, a tak knihy kategorizovat pro zvýšení přehlednosti katalogu.

1.2.4 Další funkcionalita

Mezi další funkcionalitu elektronických knihkupectví je možné zařadit například práci s uživatelskými účty. Uživatel má možnost se do systému knihkupectví zaregistrovat a případně přihlásit. Toto usnadňuje objednávkový proces, jelikož uživatel není nucen

opakovaně zadávat například fakturační údaje či platební metodu, vše je uloženo v databázi knihkupectví v rámci uživatelova účtu. Uživatel může také zobrazit minulé objednávky nebo si uložit stav košíku na pozdější nákup.

Některá internetová knihkupectví začala v posledních několika letech nabízet i možnost nákupu elektronických knih nebo audioknih, pro takovéto případy je nutné umožnit zákazníkům stažení těchto zdrojů v rámci e-commerce systému. Dále je možné uvést také funkcionalitu *„hlídacího psa“*, který, pokud jej zákazník aktivuje, zákazníka může informovat o dostupnosti zboží, případných slevách, a podobně.

Je nutné zmínit, že každé e-commerce řešení, ať už se jedná o klasický e-shop nebo o elektronické knihkupectví, musí obsahovat administrativní funkcionalitu. Administrátor musí mít k dispozici systém, pomocí kterého může například přidávat do katalogu knihy, měnit stav dostupnosti knih, upravovat metadata produktů, a podobně. Taková funkcionalita může být řešena několika způsoby, tradičně je možno mít takovýto systém zakomponován přímo v používaném e-commerce řešení, jak je v základu implementováno v dříve uvedených systémech (např. PrestaShop či Shopify). Druhou možností je propojení systému s jinou aplikací (například desktopovou) pomocí REST API.

1.3 Internetová knihkupectví v České republice

V České republice existuje velké množství internetových knihkupectví. Mezi nejznámější patří například knihkupectví Kosmas, Knihy.cz, Luxor nebo MegaKnihy. Většina z nich nabízí standardní funkcionalitu, která byla již dříve zmíněna. Rozdíly jsou mezi nimi hlavně v designu stránky, případně v použitých technologiích na frontendu i backendu, jinak se funkcionalitou příliš neliší. Z konkrétních rozdílů je možno uvést například možnost recenzování knih, to je možné v knihkupectví Kosmas, Luxor a MegaKnihy (z výše uvedených), knihkupectví Knihy.cz takovouto funkcionalitu neposkytuje. Ačkoliv všechny z nich mají v nabídce audioknihy, v době psaní této práce prodává e-knihy pouze knihkupectví Kosmas.

1.4 Internetová knihkupectví v zahraničí

Asi nejvýznamnější a nejznámější internetové knihkupectví v zahraničí je Amazon, který začal knihy prodávat nejprve ve Spojených státech, následně se však rozšířil i do mnoha dalších zemí světa. V souvislosti s velkých úspěchem původně pouze elektronické verze těchto knihkupectví, kde knihy Amazon prodával společně se svými ostatními produkty

v jejich hlavním elektronickém obchodě, se společnost rozhodla provozovat také síť kamenných knihkupectví – Amazon Books. V nabídce Amazonu jsou k dispozici kromě klasických tištěných knih například e-knihy v mnoha formátech (například pro čtečky Kindle, ale také ve formátech EPUB či PDF) či audioknihy. Amazon nabízí oproti například českým knihkupectvím placené členství v programu *Amazon Prime*. Díky této funkcionalitě mohou platící členové získat například možné získat různé slevy či dopravu zdarma v rámci objednávky z Amazonu.[8]

Z dalších zahraničních online knihkupectví je možné uvést například Barnes & Noble, které má, podobně jako Amazon, kromě tištěných knih, e-knih a audioknih v nabídce například i filmy či deskové hry. Podobně jako Amazon, i tento prodejce má k dispozici možnost členství, které předplatitelům může přinést různé benefity při nakupování knih v jeho obchodě.[9]

2 JAZYK UML

Jazyk UML (Unified Modeling Language) je využíván pro tvorbu vizuálních modelů systému. Využití má především v oblasti modelování objektově orientovaných systémů, je však možné jej využít i v dalších oblastech softwarového inženýrství, díky jeho rozšiřovacím mechanismům.[10]

Modely jsou v jazyce UML vyjadřovány pomocí různých diagramů. V rámci UML existuje několik druhů diagramů, ty mohou popisovat například strukturu či chování systému nebo jeho uživatelů, ale i mnoho dalších vlastností a vztahů v rámci analyzovaného systému.[11]

2.1 Vztahy mezi objekty v UML

Vztahy (nebo také relace) zobrazují, jak jsou spolu objekty propojeny v rámci UML modelu. Každá relace má v rámci jazyka UML vlastní značku a zachycuje sémantický vztah mezi objekty, jímž náleží.[10]

Těmito vztahy mohou být například: [12]

- Závislost (Dependency) – jedná se o slabý vztah mezi entitami, kde jedna z entit využívá druhou (je na ní závislá).
- Asociace – tento vztah mezi entitami je silnější než dříve uvedená závislost. Tento vztah většinou značí, že vztah mezi dvěma entitami je dlouhodobější.
- Agregace – agregace je silnější forma asociace. Takový vztah indikuje, že jedna entita vlastní druhou entitu.
- Generalizace (zobecnění) – vyznačuje dědičnost mezi entitami, kdy jedna entita dědí vlastnosti druhé.[10]
- Kompozice – je velmi silný vztah mezi entitami, ve své podstatě silnější forma agregace.[10]

2.2 Unified Process

Již dříve bylo zmíněno, že jazyk UML je pouze modelovací nástroj (jazyk), nikoliv metodika modelování. K tomuto účelu zde slouží Unified Process (UP). UP a UML jsou úzce spojené a užívané společně v rámci softwarové analýzy a vývoje.[10]

2.3 Požadavky a případy užití

Požadavky jsou v rámci jazyka UML v nejjednodušší rovině dvojího typu:

- Funkční požadavky
- Nefunkční požadavky

Je však možné je dále dělit do jiných, specifičtějších, kategorií.[10]

Zatímco funkční požadavky určují, jak se má systém chovat, či co má dělat – čili v důsledku se jedná o požadavky na funkčnost systému, nefunkční požadavky se zabývají omezeními systému, které nesouvisejí s jeho přímou funkčností. Tyto typy požadavků se řadí do Modelu požadavků. V rámci analýzy požadavků dále existuje ještě jeden model – model případů užití.[10]

2.3.1 Analýza požadavků

V rámci analýzy (nebo taky inženýrství) požadavků se provádí zejména zjišťování a dokumentace požadavků na softwarové řešení. V rámci tohoto cíle se zjišťuje účel použití onoho řešení ze strany uživatele, stejně tak, jako způsob jeho užívání. Takováto analýza bývá často prvním krokem návrhu softwarového řešení v rámci UP. Požadavky by měly být přesně a správně definované, jejich cílem je vyjádřit, co má systém dělat. Funkční ani nefunkční požadavky neřeší, jak by jich měl onen systém dosáhnout.[10][11]

2.3.1.1 Funkční požadavky

Jak již bylo zmíněno, funkční požadavky definují přímou funkčnost systému, tedy popisují žádané funkce systému.[10]

Pokud by se kupříkladu jednalo o funkční požadavky na systém internetového diskusního fóra, bylo by jako příklad funkčních požadavků možné uvést například možnost registrace a přihlášení uživatele do systému, správa příspěvků uživatelem i administrátorem, možnost přidání nového příspěvku, možnost přidávat k příspěvkům komentáře, a podobně.

2.3.1.2 Nefunkční požadavky

Nefunkční požadavky se zabývají omezeními či specifikacemi implementace systému. [10]

Může se tedy jednat například o hardwarové a softwarové požadavky, například na server, požadavky na latenci, load balancing, zabezpečení, způsob ukládání dat, a další podobné

požadavky. V některých případech se mohou nefunkční požadavky zaobírat i omezeními při procesu vývoje systému.

2.3.2 Modely případů užití

Diagramy případů užití mají zachycovat implementaci funkčních požadavků v systému. Jejich výsledkem je vizuální podání informací o tom, co má systém dělat a jak mezi sebou interagují některé jeho prvky, jako jsou aktéři a objekty.[12]

Podle Jamese Rumbaugha [10] je případ užití „specifikace posloupností činností včetně proměnných posloupností a chybových posloupností, které systém, podsystém nebo třída může vykonat prostřednictvím interakce s vnějšími aktéry.“ Případy užití bývají zpravidla vyjadřovány z pohledu aktéra a také jsou aktérem iniciovány (systém zde pouze „vyřizuje“ aktérovy požadavky).

2.3.2.1 Aktéři

V rámci UML se jako aktér označuje vnější entita, která pracuje s navrhovaným systémem. Často toto bývá člověk, například v roli zákazníka či administrátora systému, může ale jít i o jiný systém, čas, a podobně. [10]

U aktérů může docházet ke generalizaci (zobecnění). Jedná se o přenesení vlastností i propojených entit (jako jsou například případy užití) z jednoho aktéra na druhého. Jako příklad lze opět uvést diskusní fórum, kde se nachází několik uživatelských rolí, je zde uživatel-diskutér, moderátor a administrátor. Uživatel má například možnost přihlásit se, založit diskusi, komentovat již založené diskuse, upravovat svůj uživatelský profil, a podobně. Všechny tyto vlastnosti po něm dědí i výše postavení aktéři (moderátor a administrátor), kteří kromě těchto vlastností mají i své další, které jsou specifické pouze pro jejich role. Toto obecně zabraňuje redundanci v rámci modelu aktéři-případy užití.[10]

2.3.2.2 Scénář případu užití

Průběh případu užití specifikujeme ve scénáři případu užití. Takovýto scénář bývá většinou zapisován do tabulky a obsahuje informace jako jsou aktéři, kteří ve scénáři vystupují, vstupní a výstupní podmínky scénáře, možné z něj vycházející alternativní scénáře, a samozřejmě také průběh (či popis) scénáře. Kromě těchto základních informací je možné tabulku scénáře rozšířit i o dodatečné informace. Hlavní scénář, jak již bylo uvedeno, se nadále může větvit na množství alternativních scénářů. Tato skutečnost se často odvíjí od

možných rozhodnutí a akcí aktéra nebo limitací systému v rámci hlavního scénáře. Jako příklad je možné uvést situaci, kdy v rámci hlavního scénáře probíhá registrace uživatele do systému. V momentě, kdy uživatel pochybí v jednom z kroků hlavního scénáře (nemůže přejít na další krok), v tomto případě by to mohlo být zadání neplatné emailové adresy nebo telefonního čísla do registračního formuláře, spouští se z tohoto bodu alternativní scénář, který na situaci reaguje. Systém by v tomto případě upozornil uživatele na pochybení při registraci a vyzval jej, aby údaje opravil.[10]

Název: **Přihlášení do systému**		
ID: UC000		
Charakteristika: Zachycení ukázkového případu užití		
Primární aktér: Uživatel		
Vedlejší aktéři: CMS		
Vstupní podmínky: Uživatel se musí chtít přihlásit		
Výstupní podmínky: Uživatel musí být potvrzen administrátorem		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel se snaží přihlásit
2	CMS	CMS zobrazí přihlašovací formulář
3	Uživatel	Uživatel zadá jméno a heslo
4	CMS	CMS ověří platnost údajů v databázi
5	CMS	CMS potvrdí platnost údajů
6	Uživatel	Uživatel přistoupí do systému
7	-	UC končí
Alternativní scénáře: UC000a – Ověření selhalo		

Obrázek 2 – Příklad scénáře případu užití

2.4 Diagram tříd

Diagram tříd je využíván k popisu prvků systému a jejich vztahů a závislostí. Jelikož jsou takové diagramy určeny k modelování statické struktury systému, dříve zmíněné vztahy mezi jeho prvky se v průběhu času nemění. Tyto třídy obvykle odpovídají struktuře tříd v rámci samotné (objektově-orientované) softwarové implementace projektu.[10]

Diagram tříd se skládá z objektů, které popisují vlastnosti třídy. Tyto objekty jsou v UML reprezentovány obdélníkem rozděleným na několik polí, které mohou zobrazovat atributy nebo operace objektu. Objekty mohou mezi sebou mít různé vztahy, jak je již popsáno v kapitole 2.1.[12]

2.4.1 Atributy tříd

Atribut je vlastnost, kterou třída má. V kontextu programování navrhovaného systému se může jednat o proměnnou třídy. Stejně jako je možné přímo v programu definovat typ proměnné a její další vlastnosti, i u objektu v rámci diagramu tříd je toto možné zapsat. Nabízí se například možnost definovat viditelnost atributu (private/protected/public), jeho jméno, datový typ (integer, string, boolean, apod.), ale i například defaultní hodnotu atributu.[11]

Zápis atributu proto může vypadat například takto:

+ isAlive : boolean = true

2.4.2 Operace tříd

Operace jsou funkce či metody, kterými třída disponuje. Některé vlastnosti notace mají atributy i operace společné, například opět již zmiňovanou viditelnost, název či typ. Narozdíl od atributů však operace obsahují parametry (vstupní hodnoty).[11]

Kupříkladu může být zápis operace reprezentován následně:

+ getBalance(accountId : int) : double

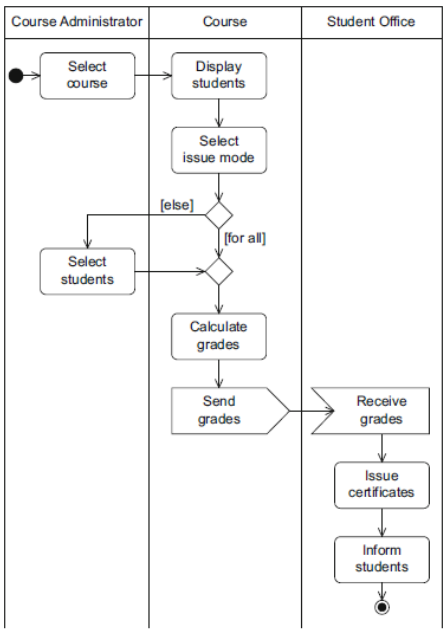
2.5 Sekvenční diagram

Sekvenční diagram je typ diagramu, který zobrazuje interakce mezi objekty v rámci jakési sekvence. Hlavními hybateli tohoto diagramu jsou zprávy, jejichž posloupnost v rámci komunikace mezi objekty dává sekvenčnímu diagramu smysl a reflektuje následnost této komunikace v čase. Objekty sekvenčního diagramu mohou být jednotlivé třídy definované v diagramu tříd, zprávami pak například jejich operace. Součástí diagramu bývají často i instance aktérů. Sekvenční diagramy mohou také reprezentovat případ užití, přesto ale nebývají přesným zobrazením případu užití tak, jak byl původně sepsán, nýbrž jeho průběh a chování zachycují pomocí použití analytických tříd. Ve výsledku se tak sekvenční diagram nezabývá například interakcí uživatele se systémem.[10][13]

V rámci sekvenčního diagramu se může kromě objektů a zpráv objevovat několik dalších prvků. To mohou být například větve a smyčky (branches, loops). Větve, jak již název napovídá, poskytují možnost větvení v rámci sekvence. Diagram je tak možné fragmentovat a vytvářet například alternativní sekvenci. Každý takový fragment je vybaven operátorem přechodu (guard), který rozhoduje o tom, zda bude sekvence fragmentu vykonána či nikoliv. Takovýto operátor může být například podmínka (if/else), může se však dále jednat i o smyčku (loop) či její přerušení (break) nebo například switch.[10][11]

2.6 Aktivitní diagram

Aktivitní diagram, nebo též někdy diagram aktivit, je diagram, který zobrazuje model procesu jakožto aktivitu. Tímto tedy zprostředkovává zobrazení chování zobrazovaného prvku a lze jej využít ve spojení s již v minulých kapitolách uvedenými součástmi UML, jako jsou například třídy, případy užití či samotné operace. Aktivity v rámci tohoto diagramu jsou složeny z uzlů, které bývají zpravidla spojeny hranami (nodes, edges). Obecně slouží uzly k vyjádření aktivity a hrany k zobrazení posloupnosti procesů v rámci aktivity. Podobně jako sekvenční diagram, i diagram aktivit může obsahovat větvení, to je vázáno na podmínky dané v diagramu. Větvení je realizováno pomocí rozhodovacích a slučovacích uzlů, které jsou reprezentovány diamantovou notací.[10]



Obrázek 3 – Příklad aktivitního diagramu s větvením [11]

3 TECHNOLOGIE UŽITÉ PŘI TVORBĚ VLASTNÍHO ŘEŠENÍ

V rámci tvorby prototypu vlastního řešení je nutné tímto prototypem uvažovat přístupnou a bezpečnou webovou aplikaci, která bude zároveň i škálovatelná. Pro tyto účely bude využit framework Laravel stavěný na jazyce PHP, který zmíněné požadavky splňuje. Dále bude využit relační databázový systém MySQL pro ukládání všech dat, se kterými bude navrhovaný systém pracovat.

3.1 Laravel

Laravel je poměrně nový, avšak široce využívaný, open-source PHP framework, který vyniká svou širokou škálou již zabudované funkcionality, ať už své vlastní, nebo převzaté například z PHP frameworku Symfony, a podobných. Laravel využívá softwarovou architekturu typu MVC, která logicky rozděluje aplikaci na tři součásti – Model, View a Controller (MVC). V této koncepci model reprezentuje data, se kterými aplikace pracuje, v případě Laravelu se jedná o tabulky v databázi a vztahy mezi nimi. Controller (někdy také řadič) zpracovává požadavky uživatele na aplikaci, podle požadavku poté může provádět manipulaci s daty s využitím propojeného modelu a aktualizovat odpovídající view (někdy také pohled). Je tedy možné říci, že controller pracuje jako jakýsi prostředník mezi modelem a view. Poslední komponent MVC je view, který reprezentuje uživatelské prostředí aplikace, do kterého jsou pomocí controlleru injektována data. Jako příklad by bylo možné v případě elektronického knihkupectví uvést zobrazení seznamu knih – uživatel vyhledá kategorii knih, controller pro operaci s knihami tento požadavek zpracuje a vyžádá si od modelu data o knihách v této kategorii, následně uživateli vyrenderuje view s kýženými knihami. Laravel je distribuován pod MIT licencí.[14][15]

3.1.1 Práce s databázemi

Laravel podporuje všechny běžně užívané SQL databázové systémy, jako jsou MySQL, PostgreSQL i Microsoft SQL Server. V rámci tohoto frameworku existují tři způsoby práce s databází:

- Klasickou PHP implementaci SQL dotazů (přes PDO extension)
- Query Builder
- Eloquent ORM

Všechny tyto přístupy využívají rozšíření PDO (PHP Database Objects) a lze je v rámci aplikace libovolně kombinovat. Query Builder poskytuje jednoduché prostředí k vykonávání různých operací nad databází, ať je to výběr dat, jejich třídění, vkládání či mazání záznamů, zároveň chrání proti zranitelnostem typu SQL injection. Má podporu i tzv. *raw queries*, kde je možné kombinovat funkce Query Builderu a klasického SQL dotazu nebo jeho části. Příklad takové funkcionality lze demonstrovat takto:

```
$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();
```

Obrázek 4 – Příklad Query Builder dotazu s použitím raw query

V tomto příkladu z dokumentace Laravelu je využita třída DB reprezentující Query Builder, která vybírá z tabulky uživatelé pomocí SQL dotazu určitá data, která jsou následně dále zpracovávána pomocí funkcí Query Builderu a následně zapsána do proměnné.[16][17]

Jednou z nejdůležitějších součástí Laravelu je Eloquent ORM (Object Relational Mapper), který do velké míry zjednodušuje práci s databází. Eloquent využívá modelů aplikace pro operace s databází. Každá tabulka v relační databázi je reprezentována právě jedním modelem (pokud se nejedná o tzv. *pivot table*). Díky této struktuře je také možné jednoduše definovat vztahy mezi tabulkami, Eloquent ve výchozím nastavení podporuje mnoho typů základních i pokročilých vztahů v relační databázi (One-To-Many, Many-To-Many, One-To-One, ale i polymorfní vztahy).[18]

3.1.2 Šablony Blade

Pro renderování views využívá Laravel šablonovacího jazyka Blade. Blade šablony jsou do určité míry z hlediska injekce dat z controlleru podobné šablonovacímu jazyku Jinja známého z Python frameworku Flask. V rámci Blade šablony je možné dynamicky renderovat data přijatá z controlleru, ale mohou obsahovat i vlastní PHP logiku, toto vše bez snížení rychlosti načítání stránky díky efektivnímu využití cache. Součástí tohoto šablonovacího enginu je také možnost tvorby jednotlivých znovupoužitelných komponentů a rozšiřování šablon (renderování šablony v šabloně). Blade také podporuje různé direktivy, které se objevují v klasických programovacích jazycích, jako jsou if/else a switch

statementy, while a for smyčky, a mnoho dalších. Tyto direktivy jsou uvozeny znakem zavináče a jsou zpracovávány přímo v šabloně.[19]

3.1.3 Artisan CLI

Artisan CLI (Command Line Interface) je příkazový řádek Laravel aplikací. Implementace Artisan konzole je součástí každého základního Laravel projektu a přináší škálu nástrojů, od příkazů pro tvorbu součástí aplikace, jakou jsou modely, controllery, blade šablony nebo unit testy, až po, programátorem definované, vlastní příkazy. Artisan taktéž funguje jako samostatný server, který zajišťuje běh aplikace na lokálním systému. Pomocí příkazu *php artisan serve* je možné v průběhu několika sekund mít k dispozici celou aplikaci lokálně pro manuální testování.[20]

3.1.4 Testování

Pro testování používá Laravel knihovnu PHPUnit, která je obsažena v základní instalaci Laravelu. Současná verze PHPUnit 9 podporuje velké množství testů, jako jsou unit testy (česky také jednotkové testy), feature testy či databázové testy, přestože byl původně stavěn především pro již zmiňované unit testy. Testovat je možné všechny části aplikace díky faktu, že v Laravelu je PHPUnit dále rozšířen o další komponenty, ať už z jiných testovacích frameworků, nebo své vlastní. Úspěšnost či neúspěšnost testu povětšinou rozhodují aserce. Aserce je jakýsi předpoklad stavu, který by měl nastat, jakmile je aserce provedena. Například pokud je asertováno, že se má konečná hodnota proměnné na konci testu rovnat booleovské pravdě, test projde jen v případě, že tato hodnota je vskutku *True* (booleovská pravda), naopak selže, pokud se hodnota bude rovnat *False* (booleovská nepravda). K účelu uvedenému v předchozím příkladu slouží v PHPUnit aserce *assertTrue()* a *assertFalse()*.[21][22]

3.1.5 Routing

O routing v aplikacích fungujících na frameworku Laravel se stará podpůrná fasáda *Route*, která zpracovává HTTP požadavky v aplikaci. Pokud je specifický *route* (také cesta) definován, je možné chování aplikace po zpracování takového požadavku definovat buď v controlleru, který byl této cestě přiřazen, nebo přímo jako funkci v instanci cesty. Cesty jsou definovány v souborech složky *routes* v kořenovém adresáři projektu. Pro klasické webové cesty je používán soubor *routes/web.php*. Definice cesty může vypadat například následně:

Route::get('/authors/create', 'AuthorController@create');

Route::get('/books', function () { return view('books.index'); });

V prvním z uvedených příkladů je akce, kterou má při GET požadavku na cestu */authors/create* pomocí controlleru. Naopak v druhém případě je definice této akce zapsána jako funkce, která je přímo součástí souboru *routes/web.php*. Je možné definovat i cesty pro jiné HTTP metody, jako je například POST, PUT, PATCH nebo DELETE.[23]

3.1.6 Bezpečnost

Již v základní konfiguraci je Laravel velmi bezpečný framework, který chrání stavěnou aplikaci před hrozbami, jako může být například SQL injection, Cross Site Request Forgery (CSRF) nebo Cross Site Scripting (XSS).[24]

> *Laravel features allow you to use everything securely. All the data is sanitized where needed unless you're using Laravel with raw queries. Then, you're on your own basically. The point is, Laravel gives you security for common vulnerabilities*.[24]

Jak je zmíněno ve výše uvedeném citátu ze článku o bezpečnosti Laravelu, tento framework v základu sanitizuje většinu vstupů uživatele, tedy například HTML tagy vstupující do různých formulářů či dialogových oken. Jedná se také o vstupy, kdy se uživatel může pokoušet o SQL injection, například přes přihlašovací formulář, díky sanitizaci těchto dat a dalším metodám zabránění tomuto typu útoku v knihovnách Eloquentu a Query Builderu, které byly již zmíněny v sekci 3.1.1, je tento útok takřka nemožný, tedy za předpokladu, že není v aplikaci užíváno neošetřených *raw queries*.[24]

Dalším důležitým ochranným prvkem je ochrana proti CSRF útokům. Tohoto je docíleno za použití formulářového direktivu *@csrf*, který musí být součástí každého formuláře, který je použit v Blade šablonách napříč aplikací. Tento direktiv generuje unikátní CSRF token, který zaručuje, že formulář přichází z aplikace, které token náleží, nikoliv z aplikace cizí. Dále Laravel obsahuje další bezpečnostní prvky, jako je například ochrana proti hromadnému přiřazování hodnot, která bude dále zmíněna v praktické části, šifrování cookies nebo rate limiting.[24]

II. PRAKTICKÁ ČÁST

4 OBJEKTOVÝ NÁVRH APLIKACE

V této kapitole bude proveden objektový návrh vlastního řešení aplikace internetového knihkupectví. Toho bude dosaženo hlavně za užití jazyka UML a diagramů popsaných v kapitole teoretické části zabývající se právě UML.

4.1 Analýza požadavků

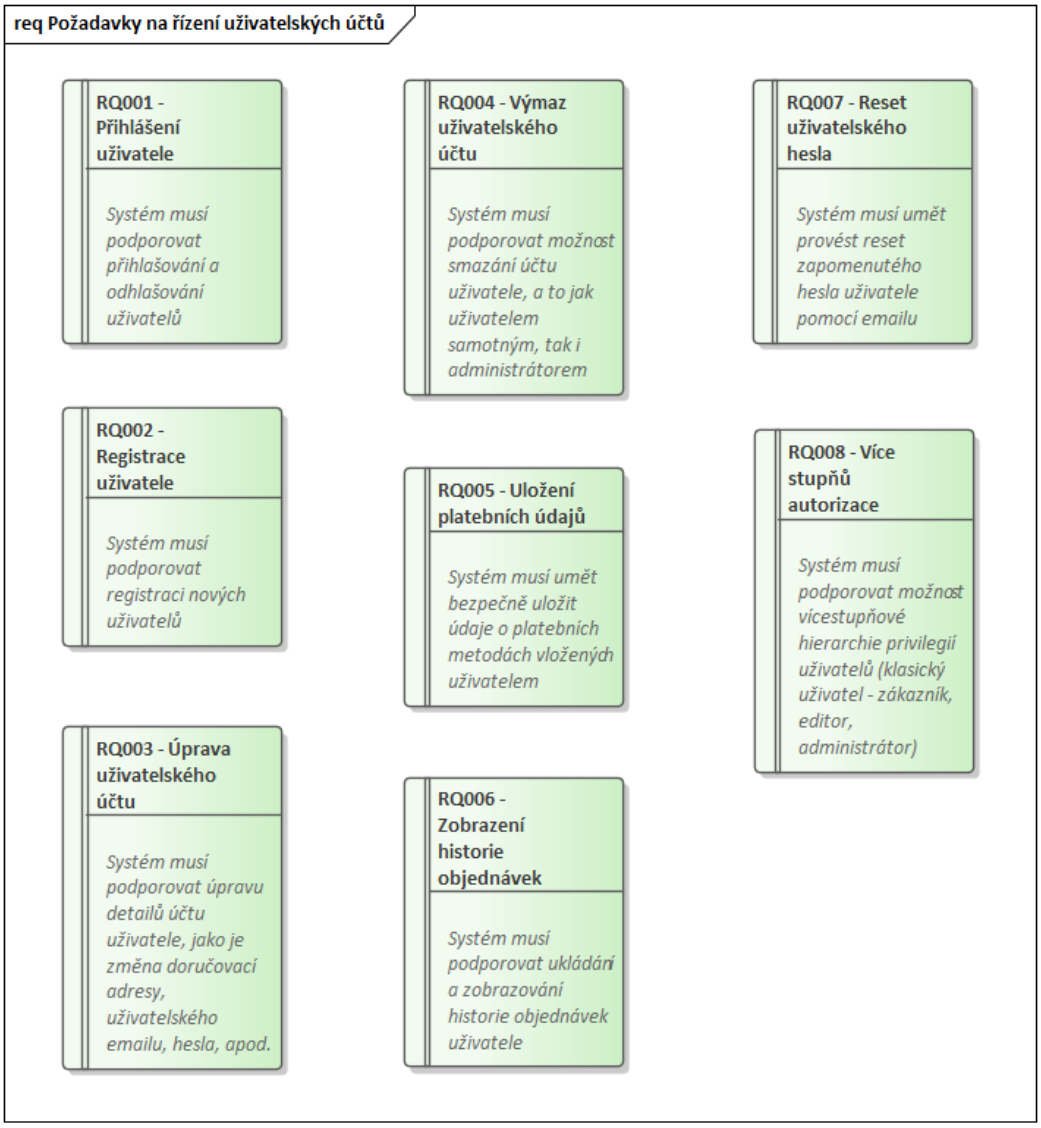
Jak již bylo uvedeno v teoretické části, požadavky se budou podle svého účelu dělit na funkční a nefunkční požadavky, z nichž funkční budou popisovat funkčnost aplikace, a nefunkční vyjadřovat omezení systému či požadavky na jeho fyzickou implementaci.

4.1.1 Funkční požadavky

Funkční požadavky byly pro větší přehlednost a výstižnost rozděleny do tří kategorií, které vypadají následně:

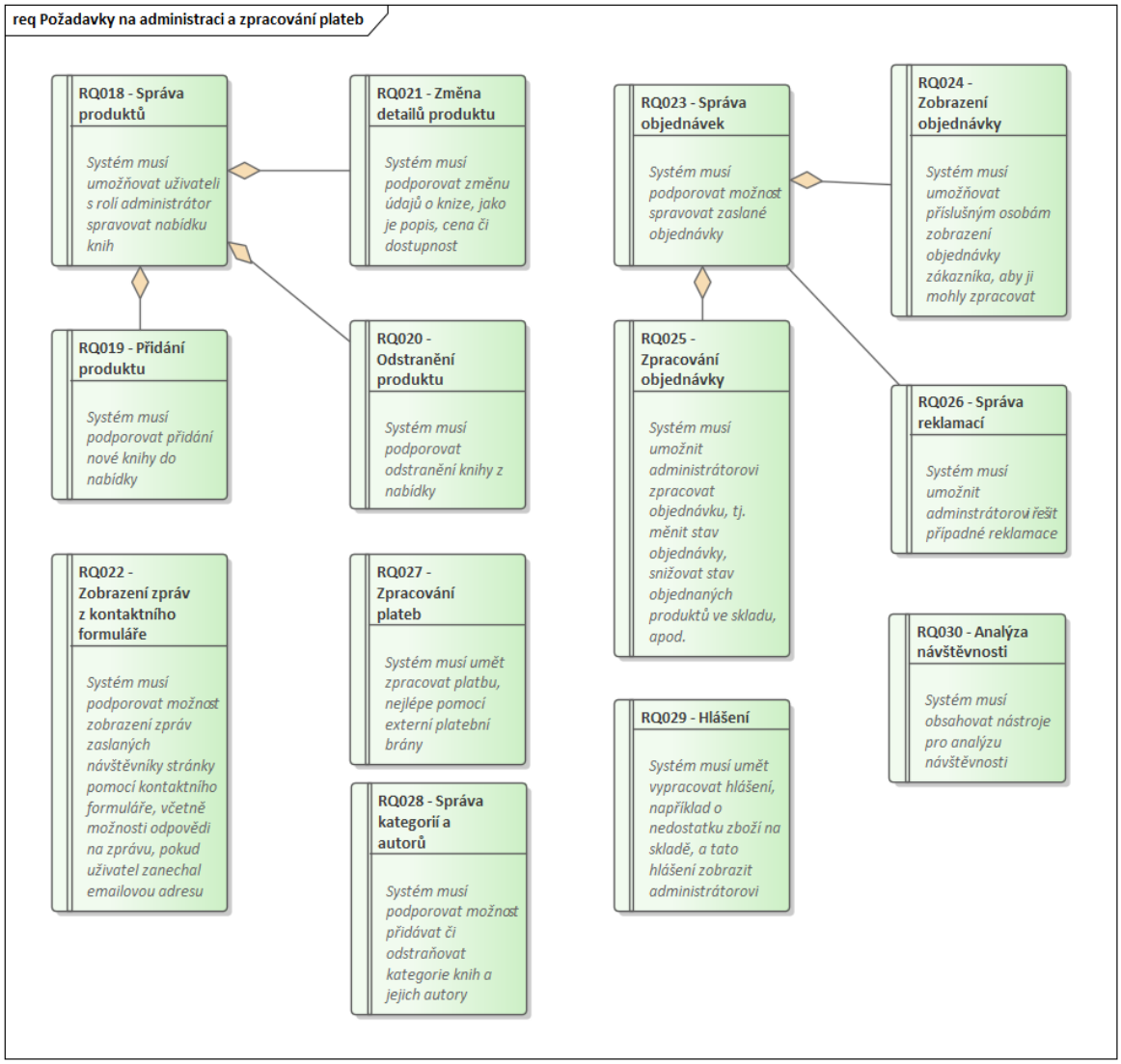
- Požadavky na řízení uživatelských účtů
- Požadavky na administraci a zpracování plateb
- Požadavky na storefront

První z uvedených kategorií se, jak již název napovídá, zaměřuje na funkčnost spojenou s řízením uživatelských účtů v rámci systému. Jedná se jak o běžné uživatelské účty, tak i o účty s vyššími privilegii, jako jsou administrátorské a editorské účty.



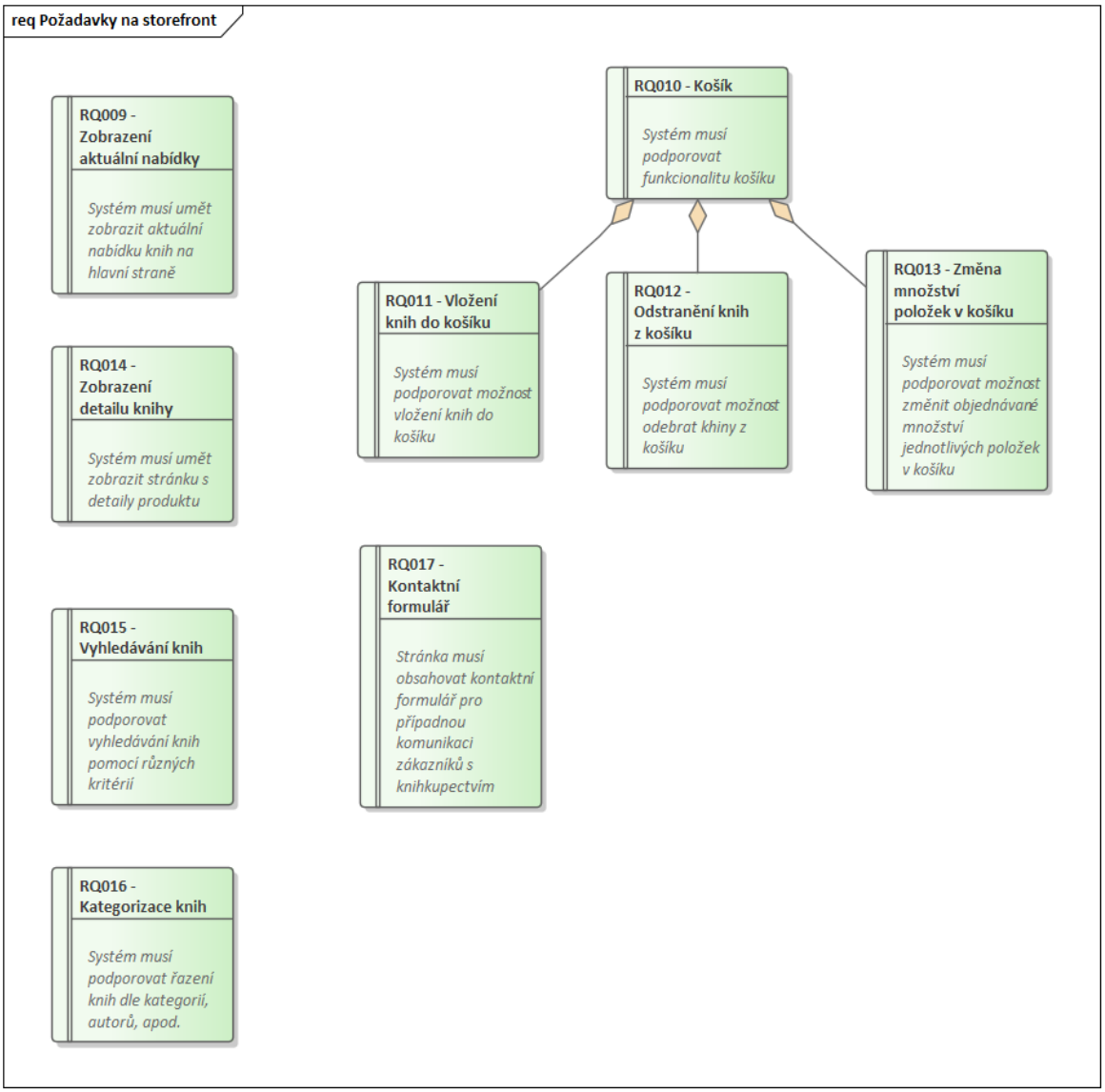
Obrázek 5 – Funkční požadavky pro řízení uživatelských účtů

Další kategorie se zaměřuje na oblast administrace systému a uskutečňování objednávek, a to včetně plateb. Ač jsou tyto platby následně odesílány platební bráně třetí strany, funkčnost pro komunikaci mezi systémem knihkupectví a platební bránou je žádoucí.



Obrázek 6 – Funkční požadavky na administraci a zpracování plateb

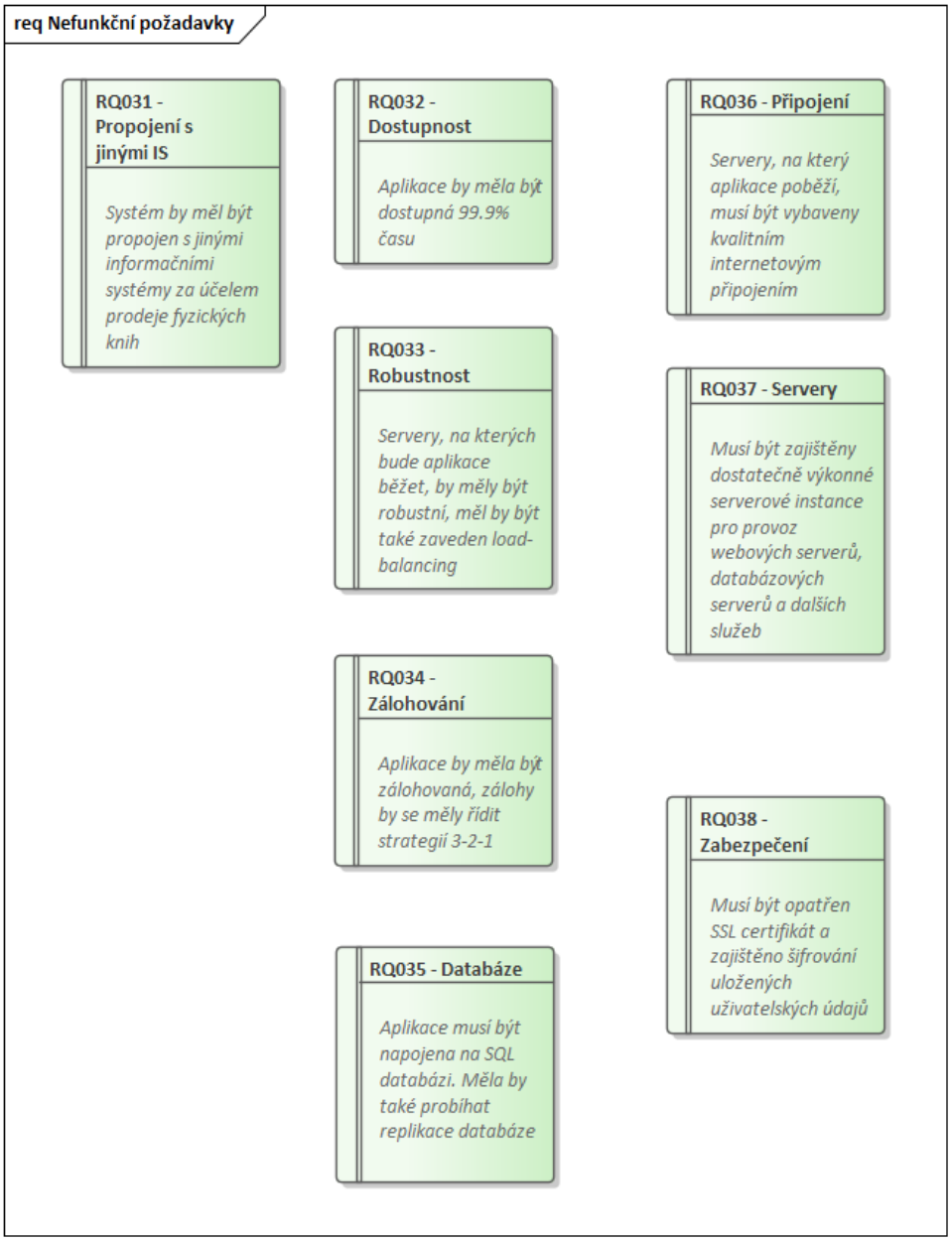
Poslední, ale neméně důležitou kategorií, jsou požadavky na hlavní stranu internetového knihkupectví, na které budou zákazníci vyhledávat knih k zakoupení.



Obrázek 7 – Funkční požadavky na storefront

4.1.2 Nefunkční požadavky

Nefunkční požadavky nebyly, kvůli svému menšímu rozsahu, seřazeny do kategorií, nýbrž jsou zobrazeny společně v rámci jednoho schématu. U těchto požadavků se jedná především o omezující požadavky systému, jako jsou nároky na provozní servery, databáze, latenci, load balancing či napojení na jiné systémy.



Obrázek 8 – Nefunkční požadavky

4.1.2.1 Realizace nefunkčních požadavků

Realizaci jednotlivých nefunkčních požadavků vyobrazuje následující tabulka, ve které je vždy ke každému požadavku uvedeno jeho navrhovaný způsob realizace.

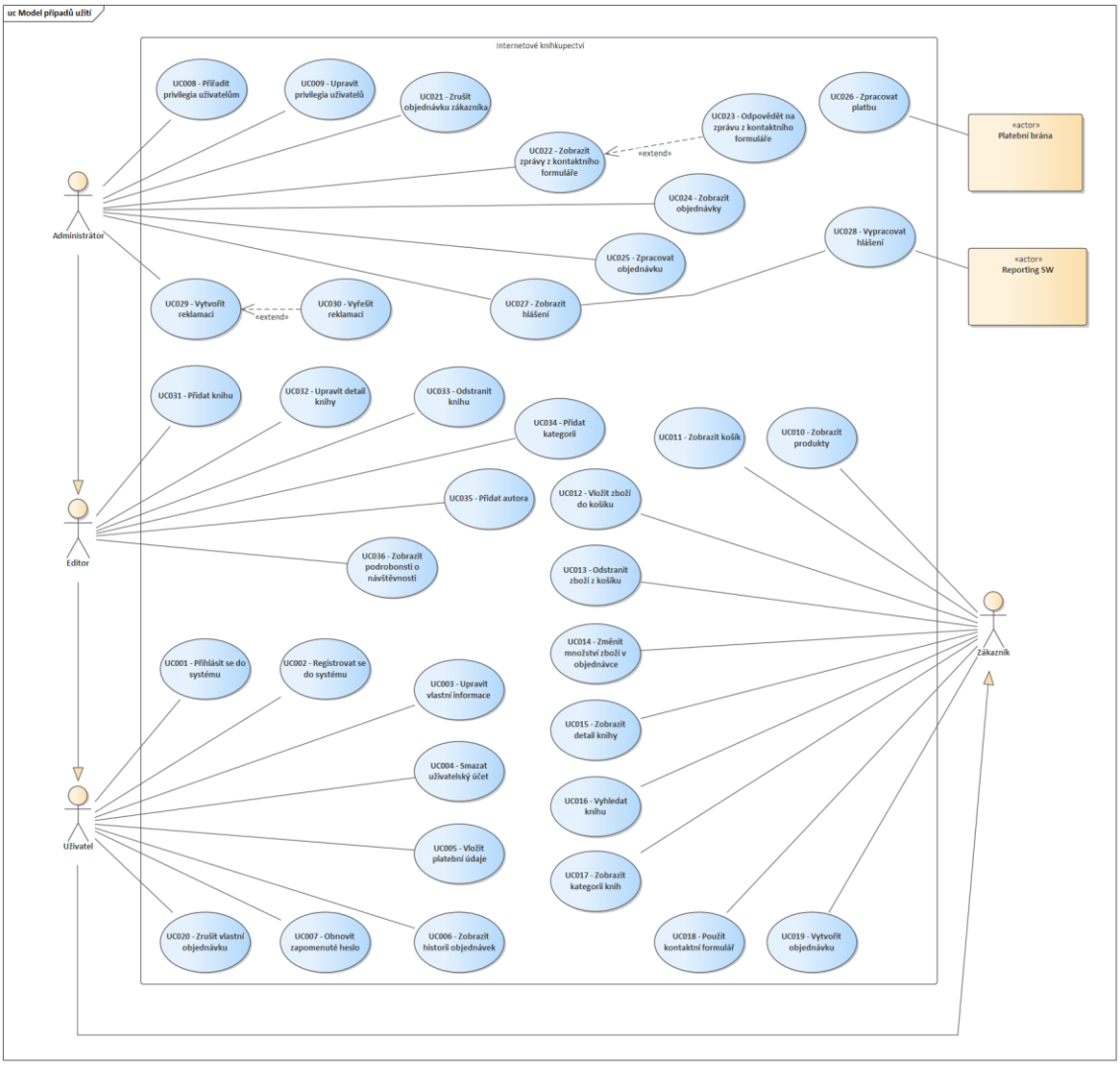
Požadavek	Realizace
RQ032 – DOSTUPNOST	Dostupnost a robustnost by měly být dosaženy zajištěním rozdělení aplikace a dalších propojených systémů mezi několik serverů, měl by být nastaven load balancer a servery by měly mít ochranu před DDoS útoky.
RQ033 – ROBUSTNOST	

RO034 – ZÁLOHOVÁNÍ	Měla by existovat automatická zálohovací strategie za použití metody 3-2-1 (3 zálohy, 2 místní a 1 na vzdáleném úložišti). Zálohovat by se měla jak databáze, tak statické soubory systému knihkupectví, jako mohou být například faktury.
RQ035 – DATABÁZE	Pro účel systému internetového knihkupectví bude použit RDBMS MySQL, zajištěna musí být i replikace databáze minimálně na dvou replikách.
RQ036 – PŘIPOJENÍ	Servery by měly mít zajištěno stabilní a kvalitní internetové připojení pro provoz aplikace a databází.
RQ037 – SERVERY	Jak již bylo zmíněno, aplikace bude rozdělena mezi několik serverů. Servery by měly vyhovovat následujícím požadavkům: - 4-8 jádrový procesor s frekvencí nad 2GHz - 16-32 GB RAM - 1Gbit Ethernet - Ubuntu Server - NGINX webserver, PHP7.3, MySQL 8 - Striktní firewallová pravidla - RAID 10 s minimálně 1TB úložištěm
RQ038 – ZABEZPEČENÍ	Připojení k serveru bude šifrované pomocí SSL/TLS, uživatelská hesla budou hashována. Pokud se budou ukládat informace o platebních kartách, musí být tyto údaje šifrovány.

Tabulka 1 – Realizace nefunkčních požadavků

4.1.3 Případy užití

Diagram případů užití vychází především z funkčních požadavků, které jsou v rámci tohoto diagramu realizovány. Je zde také vyjádřeno napojení případů užití na aktéry, mezi kterými je použito generalizace – tedy vztahu, kdy nadřazený aktér ‚dědí' vlastnosti (a tedy i případy užití) předchozího aktéra.



Obrázek 9 – Diagram případů užití

Navrhovaný systém má celkem šest aktérů, z nichž 4 jsou živí a 2 neživí. Na živých aktérech se objevuje generalizace – aktéři tedy hierarchicky dědí případy užití aktéra nižší úrovně (od zákazníka až po administrátora). Jednotliví aktéři byli popsáni takto:

1. Zákazník – Zákazník je aktér, který přichází na stránku za účelem nákupu či vyhledání knihy, není zaregistrovaný v systému knihkupectví
2. Uživatel – Uživatel je registrovaný v systému knihkupectví, má přístup k uživatelské sekci dostupné po přihlášení
3. Editor – Editor má k dispozici část administrátorských nástrojů, jeho hlavním úkolem je správa produktů (přidávání, úprava, odstraňování knih z nabídky)
4. Administrátor – Administrátor má plnou kontrolu nad systémem

5. Platební brána – Externí platební brána komunikující se systémem knihkupectví pro realizaci plateb (například Stripe, PayPal apod.)
6. Reporting SW – Reportovací software pro generování hlášení, tento software i hlášení jsou k dispozici administrátorovi systému

Požadavek / Případ užití	RQ001 - Přihlášení uživatele	RQ002 – Registrace uživatele	RQ003 – Úprava uživatelského účtu	RQ004 – Výmaz uživatelského účtu	RQ005 – Uložení platebních údajů	RQ006 – Zobrazení historie objednávek	RQ007 – Reset uživatelského hesla	RQ008 – Více stupňů autorizace	RQ009 – Zobrazení aktuální nabídky	RQ010 – Košík	RQ011 – Vložení knih do košíku	RQ012 – Odstranění knih z košíku	RQ013 – Změna množství položek v košíku	RQ014 – Zobrazení detailu knihy	RQ015 – Vyhledávání knih	RQ016 – Kategorizace knih	RQ017 – Kontaktní formulář	RQ018 – Správa produktů	RQ019 – Přidání produktu	RQ020 – Odstranění produktu	RQ021 – Změna detailů produktu	RQ022 – Zobrazení zpráv z kontaktního formuláře	RQ023 – Správa objednávek	RQ024 – Zobrazení objednávky	RQ025 – Zpracování objednávky	RQ026 – Správa reklamací	RQ027 – Zpracování plateb	RQ028 – Správa kategorií a autorů	RQ029 – Hlášení	RQ030 – Analýza návštěvnosti	RQ039 – Vytvoření objednávky	RQ040 – Zrušení objednávky
UC001 - Přihlásit se do systému	x																															
UC002 - Registrovat se do systému		x																														
UC003 – Upravit vlastní informace			x																													
UC004 – Smazat uživatelský účet				x																												
UC005 – Vložit platební údaje					x																											
UC006 – Zobrazit historii objednávek						x																										
UC007 – Obnovit zapomenuté heslo							x																									
UC008 – Přiřazení privilegií uživatelům								x																								
UC009 – Úprava privilegií uživatelů								x																								
UC010 – Zobrazit produkty									x																							
UC011 – Zobrazit košík										x																						
UC012 – Vložit zboží do košíku											x																					
UC013 – Odstranit zboží z košíku												x																				
UC014 – Změnit objednávané množství zboží													x																			
UC015 – Zobrazit detail knihy														x																		
UC016 – Vyhledat knihu															x																	
RQ017 – Zobrazit kategorii knih																x																
RQ018 – Použít kontaktní formulář																	x															
UC019 – Vytvořit objednávku																															x	
UC020 – Zrušit vlastní objednávku	x																															x
UC021 – Zrušit objednávku zákazníka	x																						x		x							x
UC022 – Zobrazit zprávy z kontaktního formuláře																						x										
UC023 – Odpovědět na zprávu z kontaktního formuláře																						x										
UC024 – Zobrazit objednávku																							x	x								
UC025 – Zpracovat objednávku zákazníka																							x		x							
UC026 – Zpracovat platbu																												x				
UC027 – Zobrazit hlášení																													x			
UC028 – Vypracovat hlášení																													x			
UC029 – Vytvořit reklamaci																										x						
UC030 – Vyřešit reklamaci																										x						
UC031 – Přidat knihu																			x													
UC032 – Upravit detail knihy																			x			x										
UC033 – Odstranit knihu																			x		x											
UC034 – Přidat kategorii																												x				
UC035 – Přidat autora																												x				
UC036 – Zobrazit podrobnosti o návštěvnosti																														x		

Obrázek 10 – Matice požadavků

Realizace konkrétních funkčních požadavků uvedenými případy užití lze pozorovat v matici požadavků vyobrazené na Obrázku 10. Jak lze z matice vyčíst, mnoho funkčních požadavků

je realizováno právě jedním případem užití. Některé požadavky však může řešit více případů užití. Například požadavek RQ 029 – Hlášení, který reprezentuje požadavek na práci s administrátorskými nástroji pro reporting stavu webu, skladových zásob a návštěvnosti webových stránek knihkupectví, je realizován dvěma případy užití, těmi jsou UC 027 – Zobrazit hlášení, kdy systém knihkupectví zobrazuje vyhotovené hlášení administrátorovi pro kontrolu, a RQ 028 – Vypracovat hlášení, kde externí software pro reporting (reprezentovaný neživým aktérem *Reporting SW*, který je popsán na předchozí straně) generuje hlášení na bázi plánovaného jobu (například CRON jobu) nebo po žádosti o report od administrátora.

Dále jsou více případy užití reprezentovány nadřazené požadavky skupin požadavků, jako je například RQ 018 – Správa produktů, jež je nadřazeným požadavkem pro požadavky RQ 019, 020 a 021 pro operace s produkty. Tento nadřazený požadavek je realizován všemi případy užití, kterými jsou realizovány podřízené požadavky.

4.1.4 Scénáře případů užití

Scénáře případu užití prezentované v tabulkách 2, 3, 4 a 5 zachycují realizaci případu užití UC 001 – Přihlásit se do systému. Existuje jeden hlavní scénář, ze kterého mohou v závislosti na interakci uživatele se systémem alternativní scénáře. Ty zachycují případy, kdy nastala chyba ve vyplnění přihlašovacího formuláře na straně uživatele.

Případ užití: Přihlásit se do systému
ID: 1
Stručný popis: Uživatel se přihlašuje do systému
Hlavní aktéři: Uživatel
Vedlejší aktéři: Žádný
Vstupní podmínky: Uživatel je zaregistrován v systému internetového knihkupectví Uživateli nebyl pozastaven účet administrátorem Uživatel není momentálně přihlášen
Hlavní scénář: 1. Případ užití začíná, když si Uživatel zobrazí přihlašovací formulář 2. Uživatel vyplní pole formuláře 3. Uživatel odešle formulář 4. Systém kontroluje, zda jsou všechny požadované údaje zadány

5. Systém kontroluje správnost údajů 6. Systém uživatele přihlásí 7. Uživatel přistoupí ke svému uživatelskému účtu
Výstupní podmínky: Systém vytvoří relaci Uživatele Systém zobrazí Uživateli podrobnosti o jeho účtu
Alternativní scénáře: 1/a – Nebyly zadány požadované údaje 1/b – Uživatel neexistuje 1/c – Zadané heslo je nesprávné

Tabulka 2 – Scénář případu užití pro přihlášení uživatele do systému

Alternativní scénář: Přihlásit se do systému: Nebyly zadány požadované údaje
ID: 1/a
Stručný popis: Systém zobrazí chybové hlášení
Hlavní aktéři: Uživatel
Vedlejší aktéři: Žádný
Vstupní podmínky: Uživatel nevyplnil jedno nebo více polí v přihlašovacím formuláři
Alternativní scénář: 1. Alternativní scénář začíná krokem 4 hlavního scénáře 2. Systém zobrazí Uživateli chybové hlášení o chybném vyplnění přihlašovacího formuláře
Výstupní podmínky: Žádné

Tabulka 3 – Alternativní scénář A případu užití pro přihlášení uživatele do systému

Alternativní scénář: Přihlásit se do systému: Uživatel neexistuje
ID: 1/b
Stručný popis: Systém zobrazí chybové hlášení
Hlavní aktéři: Uživatel
Vedlejší aktéři: Žádný
Vstupní podmínky: Uživatel není registrován v systému internetového knihkupectví
Alternativní scénář: 1. Alternativní scénář začíná krokem 5 hlavního scénáře

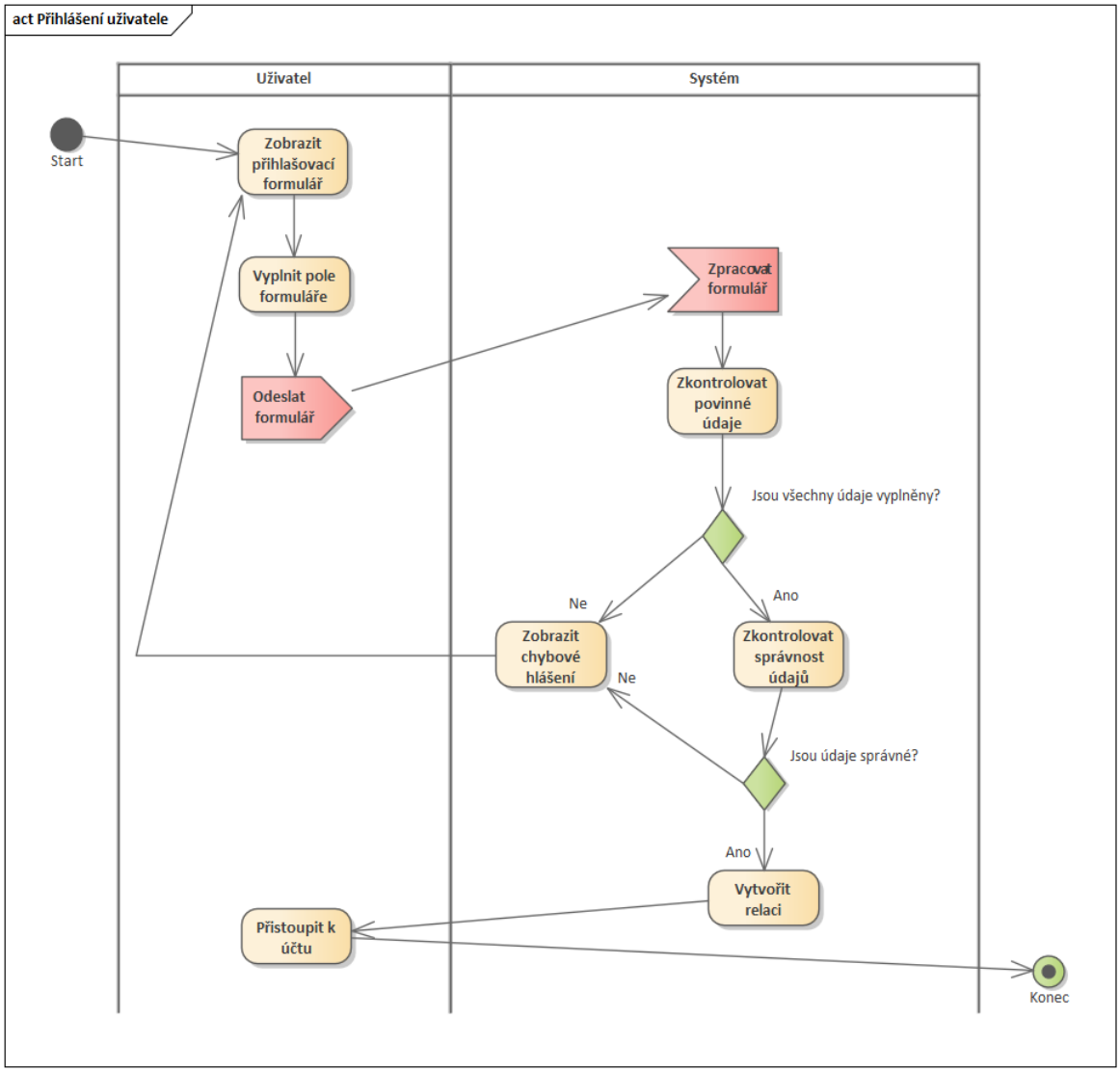
2. Systém zobrazí Uživateli chybové hlášení o neexistenci účtu odpovídajícího zadaným údajům 3. Systém vyzve Uživatele k registraci
Výstupní podmínky: Žádné

Tabulka 4 – Alternativní scénář B případu užití pro přihlášení uživatele do systému

Alternativní scénář: Přihlásit se do systému: Zadané heslo je nesprávné
ID: 1/c
Stručný popis: Systém zobrazí chybové hlášení
Hlavní aktéři: Zákazník
Vedlejší aktéři: Žádný
Vstupní podmínky: Uživatel zadal při přihlašování chybné heslo
Alternativní scénář: 1. Alternativní scénář začíná krokem 5 hlavního scénáře 2. Systém zobrazí Zákazníkovi chybové hlášení o zadání chybného hesla 3. Systém nabídne Zákazníkovi možnost resetovat heslo
Výstupní podmínky: Žádné

Tabulka 5 – Alternativní scénář C případu užití pro přihlášení uživatele do systému

Na aktivitním diagramu reprezentovaném Obrázkem 11 je možné pozorovat průběh přihlášení uživatele. Diagram odpovídá struktuře případu užití UC 001 a jeho implementaci ve scénáři s ID 1 popsaném v Tabulce 2. Spuštění alternativních scénářů je zachyceno pomocí rozhodovacích uzlů (na obrázku zelené diamanty).



Obrázek 11 – Aktivitní diagram pro UC001

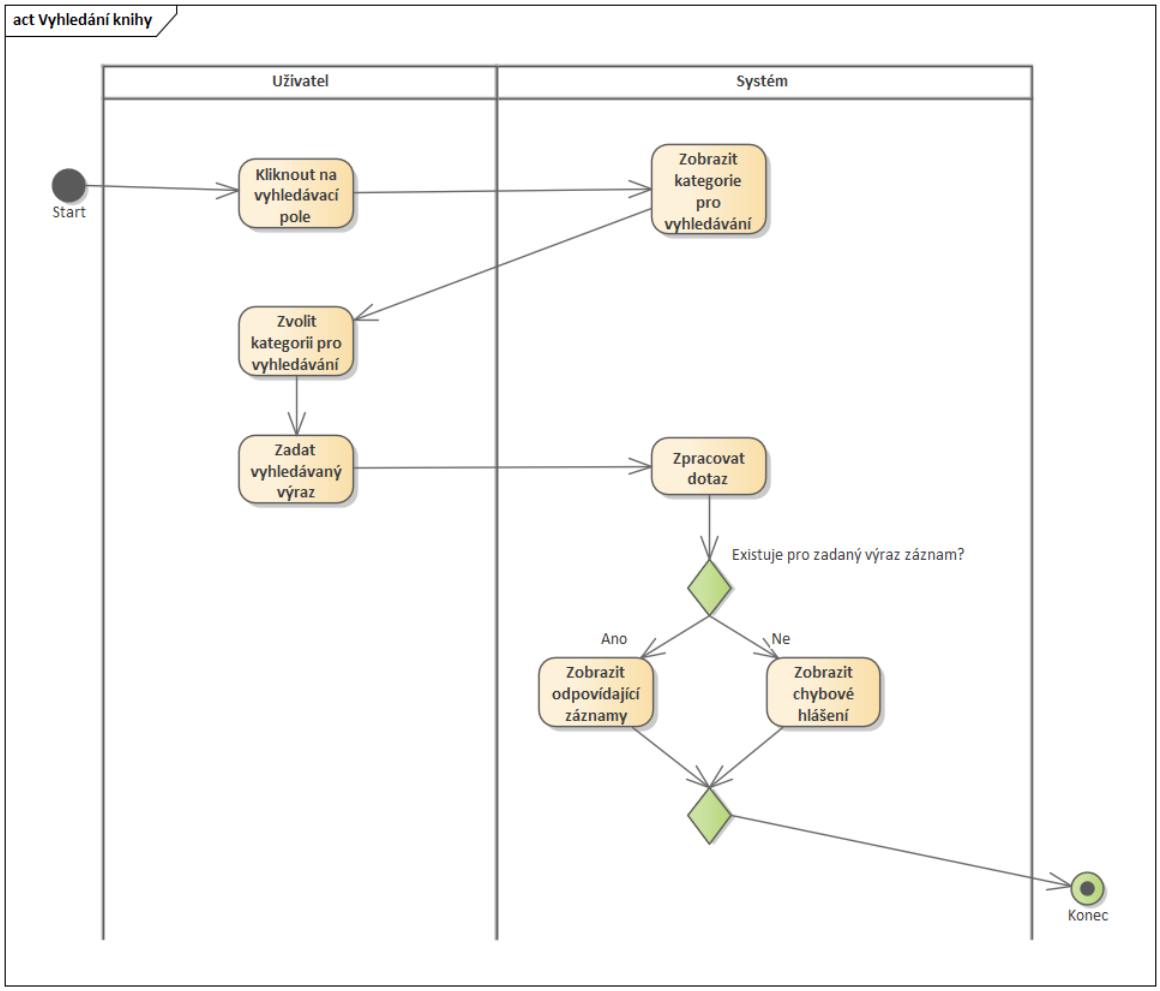
Tabulka 6 zachycuje případ užití UC 016 – Vyhledat knihu. Tomuto scénáři nepřísluší žádné alternativní scénáře, protože i když uživatel (zákazník) vyhledá výraz, který v databázi knihkupectví neexistuje, výsledek scénáře bude vždy stejný, což lze pozorovat i na diagramu aktivit zobrazeném na Obrázku 12.

Případ užití: Vyhledat knihu
ID: 2
Stručný popis: Zákazník vyhledává knihu
Hlavní aktéři: Zákazník
Vedlejší aktéři: Žádný
Vstupní podmínky: V katalogu knihkupectví existují knihy
Hlavní scénář:

1. Případ užití začíná, když Zákazník klikne na formulář vyhledávání knih 2. Zákazník zvolí kategorii pro vyhledávání 3. Zákazník zadá klíčové slovo k vyhledání do formuláře 4. Zákazník odešle formulář 5. Systém zobrazí záznamy odpovídající zadaným parametrům
Výstupní podmínky: Systém zobrazí záznamy na samostatné stránce
Alternativní scénáře: Žádné

Tabulka 6 – Scénář případu užití pro vyhledání knihy

Na Obrázku 12 je vyobrazen aktivitní diagram pro scénář případu užití popsaný v Tabulce 6. Tento diagram popisuje chování aplikace při pokusu o využití vyhledávacího formuláře návštěvníkem webu elektronického knihkupectví. Jako u předchozího aktivitního diagramu, i zde jsou využity rozhodovací uzly, v tomto případě však nespouštějí alternativní scénáře, nýbrž rozhodují o zobrazeném výsledku vyhledávání v aplikaci.



Obrázek 12 – Aktivitní diagram pro UC016

Poslední ukázkový scénář případu užití popisují tabulky 7 a 8. Tento scénář popisuje případ užití UC 014 – Změnit objednávané množství zboží. Tento případ užití se vztahuje k změně množství jednotlivých položek košíku, případně objednávky, toto by dále záleželo na implementaci přímo v aplikaci. Případ užití má jeden alternativní scénář, který se spustí v případě, že žádané množství knih v košíku či připravené objednávce je větší, než je aktuální skladová dostupnost tohoto zboží.

Případ užití: Změnit objednávané množství zboží
ID: 3
Stručný popis: Zákazník upravuje množství jednotlivých knih v košíku
Hlavní aktéři: Zákazník
Vedlejší aktéři: Žádný
Vstupní podmínky: Zákazník má v košíku alespoň jednu knihu
Hlavní scénář:

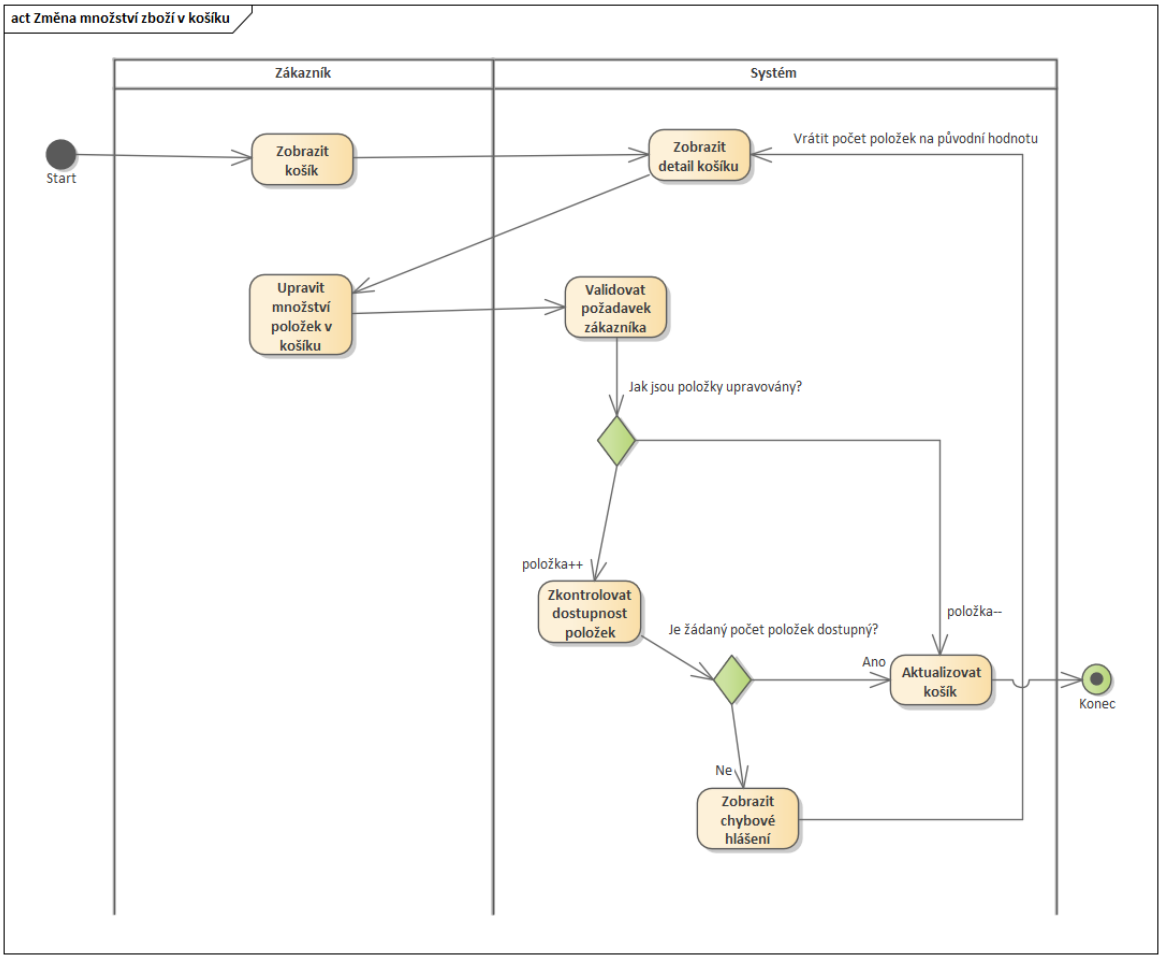
1. Případ užití začíná, když Zákazník zobrazí košík 2. Systém zobrazí produkty, které se aktuálně v košíku nacházejí 3. Zákazník změní množství objednávaných položek 4. Systém aktualizuje košík
Výstupní podmínky: Systém zobrazí nový stav košíku
Alternativní scénáře: 3/a – Požadované množství zboží není k dispozici

Tabulka 7 – Scénář případu užití pro změnu množství zboží v košíku

Alternativní scénář: Změnit objednávané množství zboží: Požadované množství zboží není k dispozici
ID: 3/a
Stručný popis: Systém zobrazí hlášení o nedostupnosti zboží a neaktualizuje košík
Hlavní aktéři: Zákazník
Vedlejší aktéři: Žádný
Vstupní podmínky: Zákazník přidá množství objednávaných položek Požadované množství položek není k dispozici
Alternativní scénář: 1. Alternativní scénář začíná krokem 3 hlavního scénáře 2. Systém zobrazí Zákazníkovi chybové hlášení o nedostupnosti žádaného množství zboží 3. Systém vrátí množství objednávaného zboží na původní hodnotu
Výstupní podmínky: Žádné

Tabulka 8 – Alternativní scénář A případu užití pro změnu množství zboží v košíku

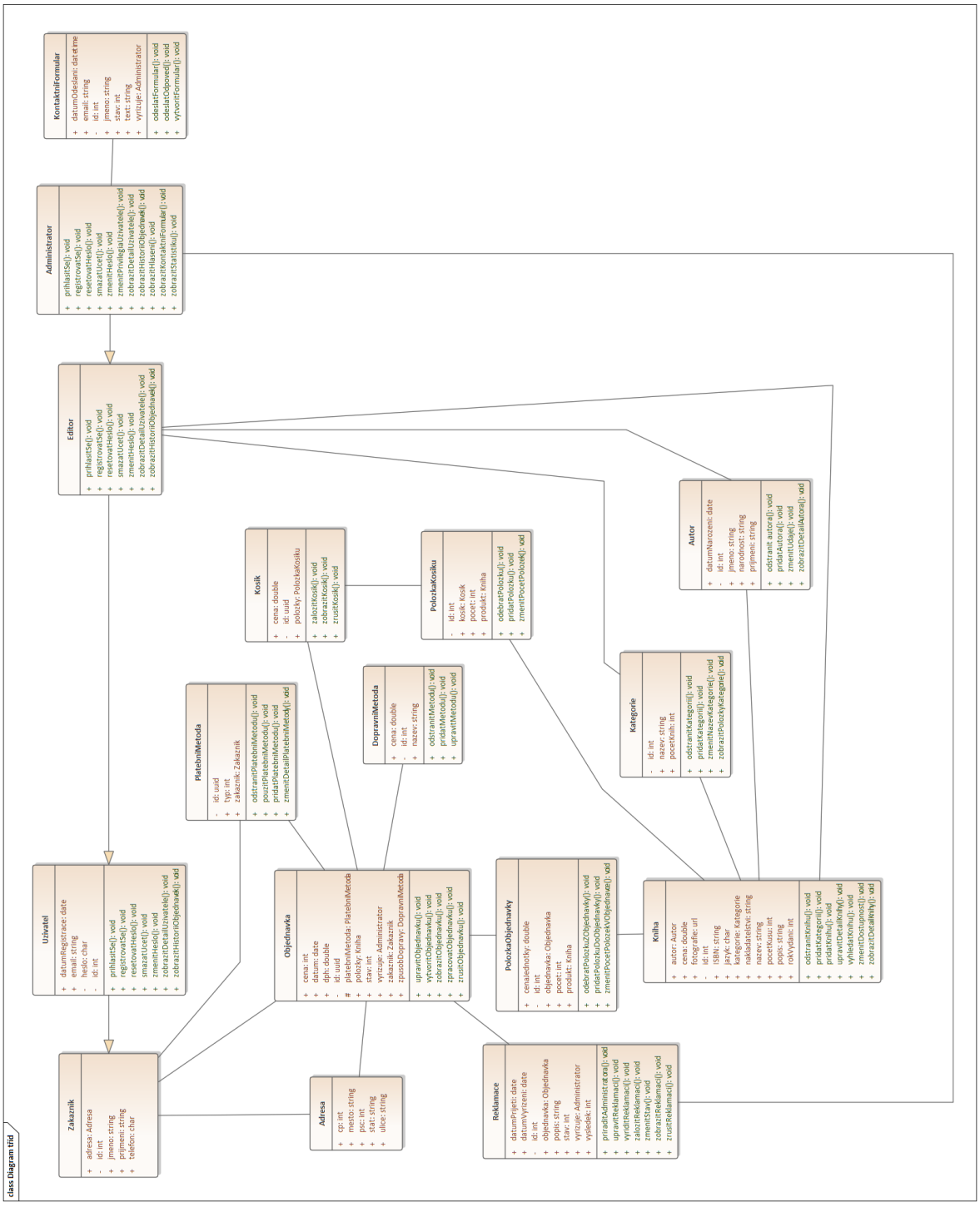
Chování systému dle výše uvedených scénářů zobrazuje aktivitní diagram na Obrázku 13. I tento diagram obsahuje aktivity uživatele (zákazníka) a systému, stejně jako rozhodovací uzly. První uzel rozhoduje o zvýšení či snížení počtu u položky košíku o jeden, druhý uzel pak kontroluje dostupnost položek. Tento uzel taktéž koresponduje s podmínkou spuštění alternativního scénáře vyjádřeného Tabulkou 8.



Obrázek 13 – Aktivitní diagram pro UC014

4.2 Diagram tříd

Diagram tříd vyobrazený na Obrázku 14 reprezentuje strukturu softwarového řešení a odráží se i na struktuře databáze. Jak již bylo popsáno i v teoretické části, tento diagram kromě popisu jednotlivých tříd zobrazuje i jejich závislosti a propojení, na zmíněném obrázku je například k vidění generalizace mezi třídami odrážejícími již dříve zmíněné živé aktéry systému, tedy zákazníka, uživatele, editora a administrátora, kteří po sobě dědí atributy i operace. V rámci MVC (Model View Controller) struktury navrhovaného systému mohou být atributy jednotlivých tříd reprezentovány v modelech systému a operace reprezentovány v jeho controllerech. Diagram v plné velikosti bude k dispozici jako elektronická příloha práce.



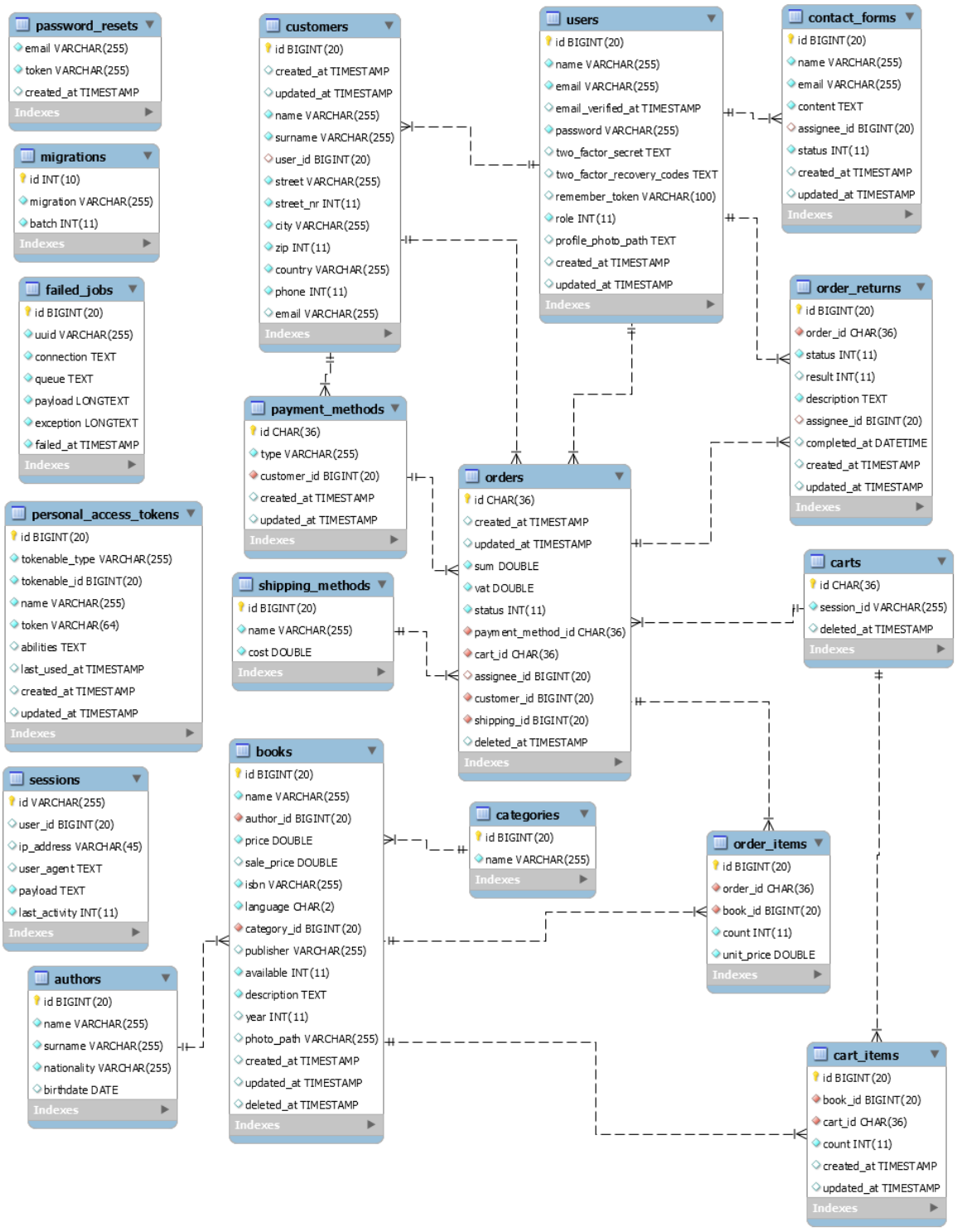
Obrázek 14 – Diagram tříd

5 PROTOTYP NAVRŽENÉHO ŘEŠENÍ

Prototyp navrženého řešení byl vytvořen v jazyce PHP, k účelu zjednodušení práce byl využit framework Laravel. Jako databázový backend byl zvolen RDBMS MySQL, který je, jakožto relační databáze, pro tento typ projektu vhodný. Pro interaktivní prvky front-endu, jako je například košík, byl použit Laravel Livewire, což je knihovna, která do určité míry nahrazuje nutnost psát větší množství JavaScriptu (JS) či používat JS frameworky, jako jsou React či Vue. Většinu kódu Livewire komponent je možné psát v PHP a šablonovacím jazyku Blade.[24] Pro potřebu interaktivity v jednodušších komponentách, jako jsou například informační okna, byla použita JS knihovna AlpineJS.

5.1 Struktura databáze

Struktura databáze odráží strukturu modelu tříd (Obrázek 14), atributy třídy by měly povětšinou odpovídat sloupcům relační databáze. Odkazy na jiné třídy, jako je tomu kupříkladu u položky košíku, která odkazuje na nadřazenou třídu Košík a třídu Kniha, reprezentující detail položky košíku, jsou řešeny pomocí cizích klíčů. Na Obrázku 15 je vyobrazena struktura této databáze pomocí relačního schématu vytvořeného programem MySQL Workbench. Tato konkrétní databázová struktura byla předtím vytvořena pomocí nástrojů dostupných ve frameworku Laravel, tedy vytvořením migrací, které po definování požadovaných sloupců a jejich vlastností pomocí PHP kódu uceleně vytvořily strukturu prezentovanou na Obrázku 15, a to včetně vztahů mezi tabulkami, které jsou mimo jiné také definovány v těchto migracích. Jak lze na schématu pozorovat, každá kniha může mít pouze jednoho autora, což by mohlo způsobovat problémy, pakliže by se v knihkupectví využívajícím navržený systém prodávaly například odborné knihy, které mívají zpravidla vícero autorů. Takový problém by mohl být řešen prostou úpravou migrace a modelu v Laravelu tak, aby vztah mezi zmíněnými entitami odpovídal typu many-to-many. Takovýto postup by měl za následek vytvoření propojovací tabulky, například s názvem *book_authors*, která by obsahovala pouze identifikátory knihy a jejích autorů jakožto cizí klíče.



Obrázek 15 – Relační schéma navržené databáze

5.2 Tvorba MVC struktury

Základní modely a controllery odpovídající tabulkám uvedeným ve struktuře databáze, společně s přidruženými databázovými migracemi a seedery pro reálná i testovací data, byly vytvořeny pomocí Artisan skriptu příkazem *php artisan make:model $MODEL -msfc*. Zprvu

byly vytvořeny migrace dle databázového schématu uvedeného výše, načež byla databáze migrována, opět pomocí příkazové řádky a skriptu Artisan.

5.2.1 Modely

Dle migrací byly následně upraveny modely tak, aby bylo možné do databáze vkládat záznamy. K tomuto účelu bylo nutné vyplnit proměnnou *$fillable* každého modelu. Tato proměnná reguluje tzv. *mass assignment* hodnot, tj., možnost vkládání hodnot více sloupců v rámci jednoho příkazu pro vytvoření nového databázového záznamu. Eloquent z bezpečnostních důvodů v defaultní konfiguraci modelu *mass assignment* pro nové hodnoty nepovoluje. Dále bylo nutné definovat relace mezi modely, v rámci navrhované aplikace se nevyužívají polymorfní vztahy, je tedy nutné definovat pouze one-to-many či one-to-one relace. Toho lze dosáhnout funkcemi Eloquentu, kterými jsou definovány vztahy přímo na příslušných modelech pomocí metod. Tyto metody lze dále volat například z Controllerů pro dotazy na tyto vztahy. Není proto nutné řešit jakýkoliv JOIN clause, jako v klasických SQL dotazech, Eloquent toto řeší sám vlastní funkcionalitou.[18]

Například vztah mezi knihou a kategorií jsou řešeny pomocí one-to-many relace, kde kategorie může mít mnoho knih, ale jedna kniha patří právě do jedné kategorie. Pomocí Eloquentu je vztah řešen metodami *hasMany()* a *belongsTo()*. Pokud by tedy vyvstal dotaz na knihy patřící do určité kategorie, je možné tento vztah dotazovat pomocí takovéto funkce:

```
$category = Category::where('name', $name)->first()->books;
```

Tento dotaz vyhledá model kategorie, jejíž název odpovídá zadané proměnné *$name* a pomocí metody *first()* vybere první záznam, na který narazí. Díky propojení s modelem *Book*, tedy modelem zodpovědným za zpracování dat o dostupných knihách, uloží do proměnné *$category* všechny knihy náležící této kategorii. K tomu využívá metodu *books()*, která je v navrhovaném řešení součástí modelu kategorie a má vztah k modelu *Book* pomocí vztahové metody *hasMany()*. Kniha je naopak propojená s modelem *Category* pomocí vztahu *belongsTo()*, což znamená, že zápis, podobný tomu výše, je možné využít i pokud by bylo žádáno tento vztah dotazovat z pohledu modelu *Book*.

5.2.2 Controllery

Controllery obsahují většinu logiky aplikace a zpravidla bývají napojeny na určitou cestu definovanou v jednom ze souboru ve složce *routes*, jak již bylo uvedeno v teoretické části práce. Controllery navrhované aplikace odpovídají CRUD (Create Read Update Delete)

koncepci, pročež většina controllerů obsahuje metody pro všechny základní operace přiřazeného modelu, tedy tvorbu záznamu, čtení (zobrazení všech záznamů z databázové tabulky určené modelem i zobrazení jednotlivých záznamů samostatně), aktualizaci určitých dat v záznamu a smazání záznamu. V rámci navrhovaného řešení jsou součástí většiny controllerů dále metody pro validaci formulářových dat odesílaných do controlleru pro zpracování a jiné metody zajišťující dodatečnou funkcionalitu pro tyto CRUD metody onoho controlleru. Validace je zpracována pomocí metody *validateInput()* využívající vestavěného validátoru, který je součástí Laravelu. Tento validátor poskytuje škálu nástrojů pro validaci vstupů uživatele, je také možné definovat vlastní validační pravidla, jak bylo nutné učinit například u validace ISBN knih.

5.2.2.1 Tvorba ISBN validátoru

Jelikož Laravel ve své základní konfiguraci nemá implementován žádný nástroj pro validaci kódů ISBN, bylo potřeba implementovat vlastní validační pravidlo pro tyto položky. K tomuto účelu byla využita knihovna *biblys/isbn*.[26]

Tato knihovna zajišťuje ověření platnosti kódu ISBN, a také případné převody mezi jednotlivými standardy ISBN (ISBN10/ISBN13).

```php
<?php

namespace App\Rules;

use Illuminate\Contracts\Validation\Rule;
use Biblys\Isbn\Isbn as IsbnValidator;

class Isbn implements Rule
{
    /**
     * Determine if the validation rule passes.
     *
     * @param  string  $attribute
     * @param  mixed  $value
     * @return bool
     */
    public function passes($attribute, $value)
    {
        return IsbnValidator::isParsable($value);
    }

    /**
     * Get the validation error message.
     *
     * @return string
     */
    public function message()
    {
        return ':attribute field value does not match any of the ISBN standards';
    }
}
```

Obrázek 16 – Implementace validačního pravidla pro kódy ISBN

Jak lze pozorovat na Obrázku 16, validátor je velmi jednoduchý. Pokud je vložená hodnota validní ISBN, metoda *isParsable()* z knihovny *biblys/isbn* vrátí booleovskou pravdu a validace je úspěšná (passes), v opačném případě validace vrátí chybovou hlášku (message).

5.2.2.2 Příklad controlleru

Na dalším obrázku je možné vidět příklad implementace základního controlleru pro administraci autorů v rámci navrhovaného řešení. Controller obsahuje metody pro CRUD operace a validaci. Veškerá validace byla napsána pod metodou *validateInput()*, aby bylo zabráněno opakování kódu v rámci controlleru, jelikož stejná validace probíhá v metodách *store()* i *update()*.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Author;
use Illuminate\Http\Request;

class AuthorController extends Controller
{
    public function index()
    {
        $authors = Author::withCount('books')->orderBy('surname')->get();

        return view('admin.authors.index', compact('authors'));
    }

    public function show($id)
    {
        $author = Author::with('books')->where('id', $id)->first();

        return view('admin.authors.detail', compact('author'));
    }

    public function edit(Author $author)
    {
        return view('admin.authors.edit', compact('author'));
    }

    public function update(Request $request, Author $author)
    {
        $author->update($this->validateInput($request));

        return redirect('/admin/authors')->with('status', 'Success');
    }

    public function create()
    {
        return view('admin.authors.create');
    }

    public function store(Request $request)
    {
        Author::create($this->validateInput($request));

        return redirect('/admin/authors')->with('status', 'Success');
    }

    public function destroy($id)
    {
        Author::where('id', $id)->delete();

        return redirect('/admin/authors')->with('status', 'Success');
    }

    private function validateInput($input)
    {
        return $input->validate([
            'name' => ['required', 'min:2', 'max:100'],
            'surname' => ['required', 'min:2', 'max:100'],
            'nationality' => 'nullable',
            'birthdate' => ['nullable', 'date']
        ]);
    }
}
```

Obrázek 17 – Controller pro správu autorů

5.3 Komponenty šablon

Jak již bylo zmíněno v teoretické části práce, Blade šablony lze rozšířit o znovupoužitelné komponenty, což zabraňuje excesivnímu opakování kódu v rámci stavby front-endu aplikace. Stejně jako je možné do view renderovaného jakožto Blade šablonu injektovat data z Controlleru, lze do Blade komponentu posílat data jakožto proměnné v poli, které je v rámci daného komponentu označovaného jako *@props()*. Tento direktiv může obsahovat jednu nebo více proměnných, které lze dále využívat v onom komponentu. Alternativně je možné tvořit komponenty jakožto PHP třídy, tato metoda však v případě navrhované aplikace použita nebyla, jelikož komponenty nebyly natolik složité, aby bylo nutné užít tohoto postupu.

```
@props(['book'])

<div class="flex-shrink-0 mx-2 my-2 relative overflow-hidden bg-blue-300 rounded-lg max-w-md shadow-
lg">
    <div class="mr-0">
    </div>
    <div class="relative pt-10 px-10 flex items-center justify-center">
        <a href="/books/detail/{{ $book->id }}">
            <img class="relative w-40 border-2 border-transparent rounded-lg hover:border-blue-800
transform-all duration-300"
                src="/storage/{{ $book->photo_path ??
config('filesystems.cover_photo_placeholder_path') }}"
                alt="book" />
        </a>
    </div>
    <div class="relative px-6 pb-6 mt-6 text-gray-800">
        <span class="block opacity-75 -mb-1">
            <x-hoverable-link link="{{ '/category/' . $book->category->id }}"><i class="ri-book-mark-
line mr-1"></i>
                {{ $book->category->name }}</x-hoverable-link>
        </span>
        <div class="flex justify-between">
            <span class="block font-semibold text-xl">
                {{ $book->name }}
            </span>
            <span
                class="bg-white rounded-full text-blue-800 text-xs font-bold px-3 py-2 leading-none
flex items-center">
                <i class="ri-price-tag-3-line mr-2"></i>
                {{ $book->sale_price ?? $book->price }} Kč
            </span>
        </div>
        <span class="block opacity-75 text-sm -mb-1">
            by <x-hoverable-link link="{{ '/authors/detail/' . $book->author->id }}">
                {{ $book->author->name . ' ' . $book->author->surname }}</x-hoverable-link>
        </span>
    </div>
</div>
```

Obrázek 18 – Příklad komponentu pro zobrazení karty knihy

Na Obrázku 18 je vyobrazen kód jednoho z těchto komponentů. Jedná se o komponentu karty nabídky knih na hlavní straně. Do komponentu vstupuje proměnná *book*, která

reprezentuje kolekci dat o jedné knize pocházející z původní Blade šablony, které komponent náleží. Z kolekce jsou pak zpracovány informace, které se mají na kartě zobrazit, jako je například název knihy, kategorie, autor či prodejní cena. Jak lze na obrázku dále pozorovat, i v rámci tohoto komponentu se užívá další komponent uvozený elementem <*x-hoverable-link*>, což značí, že je možné využít metodu tzv. *nested* komponentů, tedy komponentu uvnitř komponentu.

Laravel v základním nastavení hledá všechny komponenty v adresáři *resources/views/components*, tyto komponenty musí být při použití vždy uvozeny jako <*x-komponenta*>, kde komponenta je název blade souboru v adresáři s komponenty. V případě již zmiňované komponenty <*x-hoverable-link*> je soubor s takovýmto komponentem pojmenován *hoverable-link.blade.php*.

5.4 Autentizace a autorizace

Pro potřeby autentizace a autorizace byla využita first-party knihovna Laravel Jetstream, která využívá knihoven Laravel Fortify a Laravel Sanctum pro operace s uživatelskými účty v rámci aplikace. Vzhledem k faktu, že navrhovaná aplikace funguje pouze na bázi webového rozhraní a nemá k dispozici API endpointy, je v ní ve pro autentizaci využíván pouze Laravel Fortify. Pokud by do aplikace měly být později doplněna funkcionalita pro REST API, Laravel Sanctum by byl použit například pro operace s API tokeny a regulaci přístupu uživatelů ke zmiňovaným API endpointům aplikace.

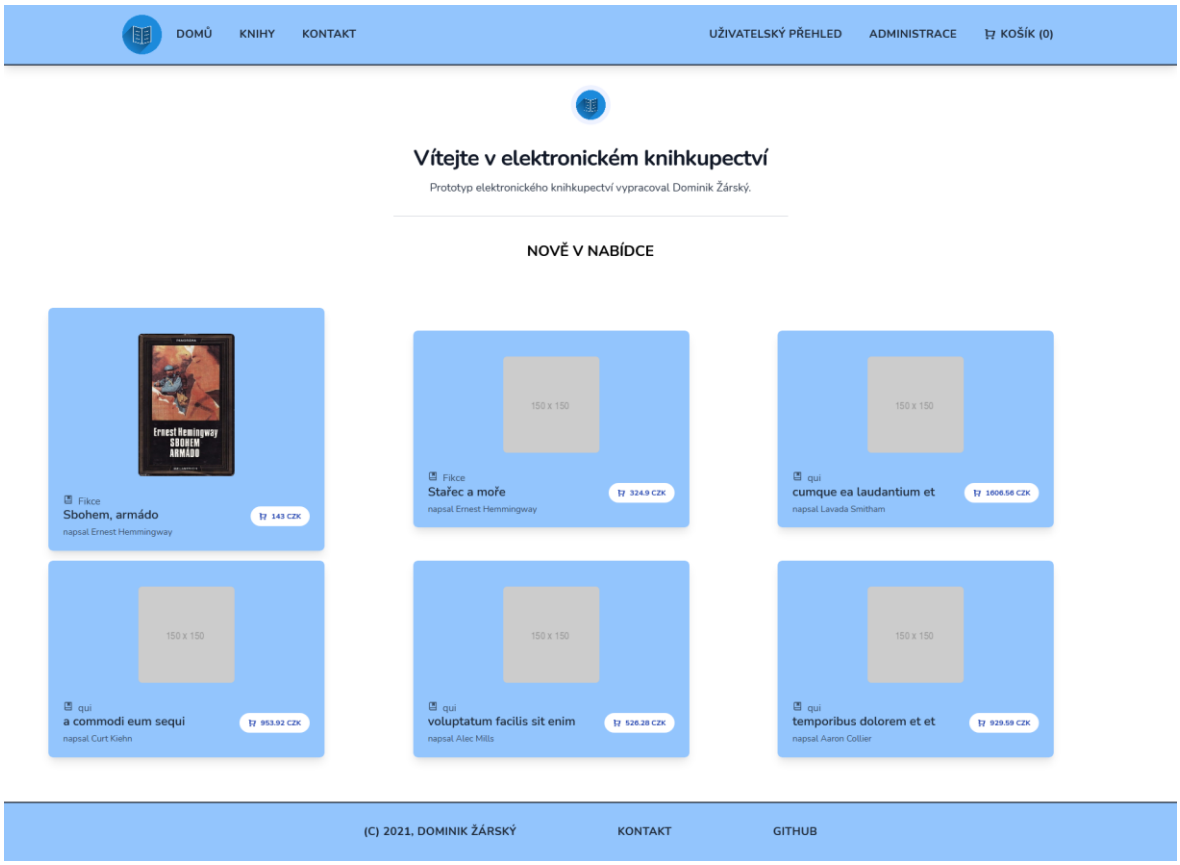
Uživatel se přihlašuje pomocí uživatelského emailu a hesla, toto heslo je uchováno v databázové tabulce *users* jako *bcrypt* hash. Heslo je taktéž možné hashovat pomocí algoritmu *Argon2*. Jak *bcrypt*, tak i *Argon2*, jakožto funkce patřící do skupiny KDF (Key Deravation Function), jsou v dnešní době považovány za velmi bezpečné funkce pro ukládání hesel.[27] Z bezpečnostního hlediska dále může uživatel využít možnosti zapnutí dvoufaktorové autentizace, která je součástí knihovny Laravel Jetstream.

Z hlediska autorizace je aplikace rozdělena na tři samostatné vrstvy. První vrstvou je hlavní strana knihkupectví, která je přístupná všem registrovaným i neregistrovaným návštěvníkům stránky. Další vrstvou je uživatelská sekce, do které mají přístup registrovaní uživatelé. V rámci této sekce stránky může uživatel zobrazit své předchozí objednávky, upravovat své fakturační údaje nebo sledovat stav svých nedokončených objednávek. Uživatel má právo zobrazovat pouze svůj profil, což je řešeno vázáním zobrazeného profilu na aktuálně přihlášeného uživatele pomocí fasády *Auth*. Profily ostatních uživatelů nejsou přihlášenému

uživateli k dispozici. Poslední sekcí je administrátorská sekce, ke které mají přístup pouze uživatelé s administrátorskými či editorskými privilegii. Úroveň privilegií je definovaná v databázové tabulce *users* jako sloupec *role*. V rámci administrátorské sekce je možné spravovat obsah stránky, jak bude dále popsáno v kapitole 5.6. Další funkcionalita obnáší správu uživatelů, objednávek, reklamací a kontaktních formulářů. Pro administrátora jsou k dispozici i hlášení o stavu skladu a návštěvnosti webu knihkupectví.

5.5 Hlavní strana

Hlavní stranu tvoří tři jednoduché komponenty, jsou jimi menu nacházející se ve vrchní části stránky, dále to je sekce vystavující nově přidané knihy, které byly postaveny do gridu o třech řadách a třech sloupcích. O jejich zobrazení se stará *FrontPageController*, který posílá do šablony 6 nejnovějších záznamů v databázové tabulce s knihami. V rámci hlavní strany jsou zobrazeny pouze základní informace o knihách, kliknutím na obrázek obalu knihy je však možné zobrazit detail knihy, kliknutím na cenovku je možné přidat knihu do košíku. Poslední částí strany je patička, kde je uveden copyright a několik odkazů. Hlavní strana je vyobrazena na Obrázku 19.

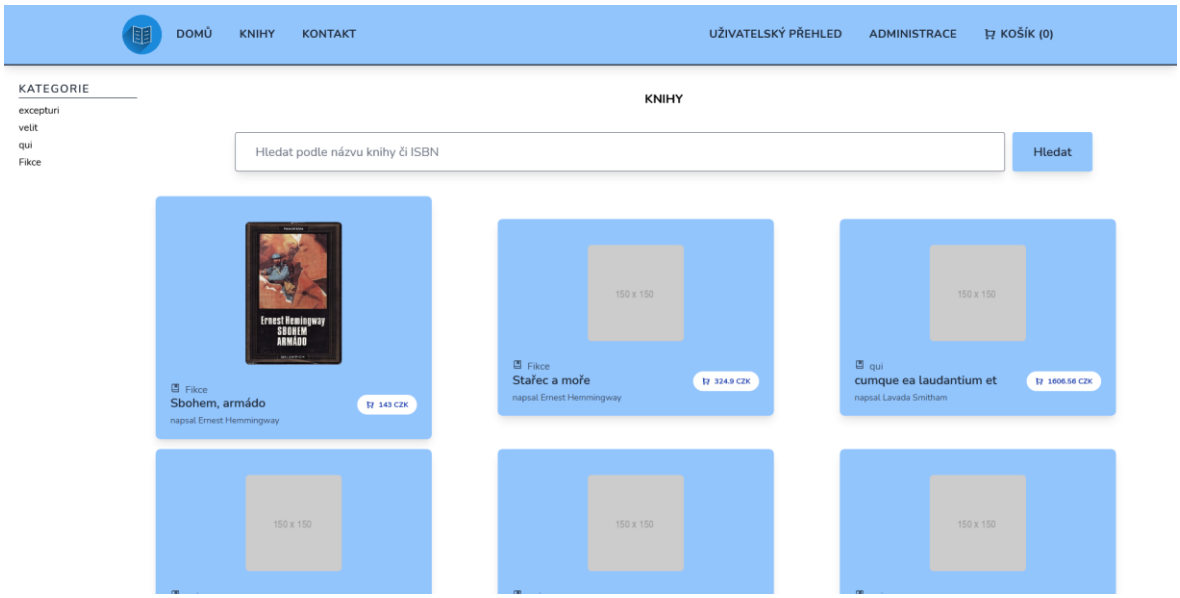


Obrázek 19 – Hlavní strana aplikace

Při tvorbě front-endu aplikace byla použita CSS knihovna TailwindCSS, některé komponenty byly převzaty z knihoven MambaUI, Kometa UI Kit a Tail-Kit, tyto komponenty byly následně přepracovány pro využití v prototypu knihkupectví. Pro práci s interaktivními prvky stránky byly využity knihovny Laravel Livewire a AlpineJS. Pro potřebu zobrazování ikon na stránce byla použita knihovna Remix Icons.

5.5.1 Nabídka knih

V nabídce knih jsou návštěvníkovi zobrazeny všechny knihy, které jsou stránkovány. Aplikace zobrazuje 18 záznamů na stránku, které jsou opět seřazeny podle data přidání. Dále má návštěvník na této stránce možnost řadit knihy podle kategorie, které jsou zobrazeny ve vertikálním menu na levé straně stránky.



Obrázek 20 – Obrazovka nabídky knih

5.5.2 Vyhledávání

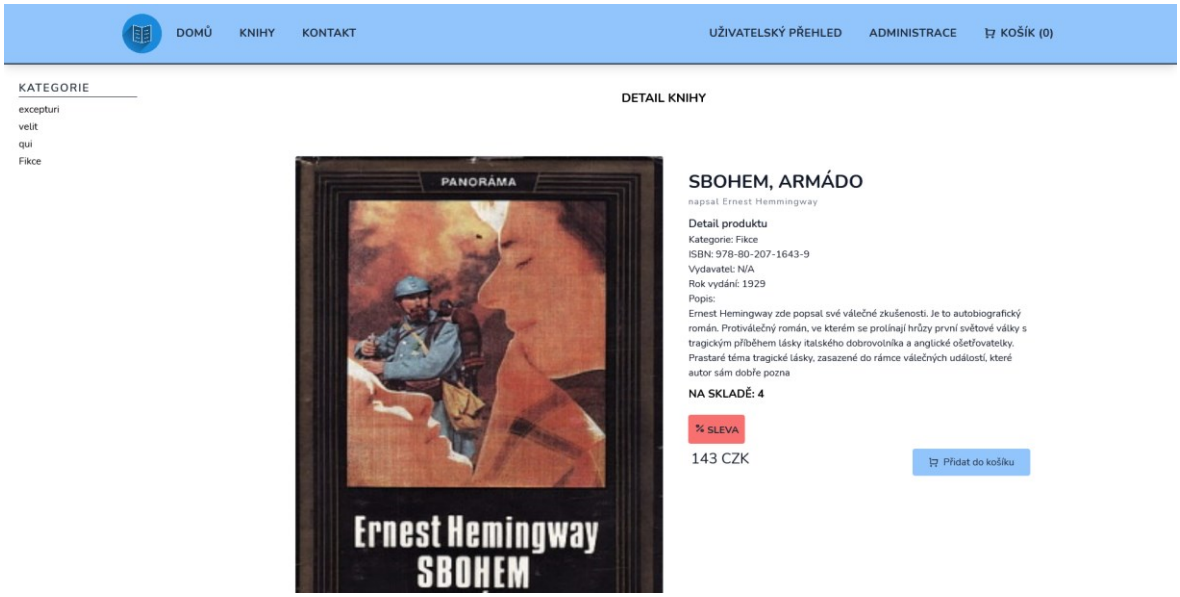
Vyhledávání je součástí stránky nabídky knih, zákazník zde může vyhledávat knihu pomocí názvu knihy či ISBN. Funkcionalita vyhledávání je řešena pomocí formuláře, který posílá GET request do controlleru zodpovědného za renderování elementů hlavní strany. Podle údajů zadaných ve vyhledávacím poli následně provede databázový dotaz na název nebo ISBN knihy pomocí Eloquentu a modelu *Book*. Zápis zmiňovaného postupu v rámci controlleru je zobrazen na dalším obrázku.

```php
public function booksPage(Request $request)
{
    if (! $request->book)
    {
        $books = Book::with(['author', 'category'])->latest('created_at')->paginate(18);
    }
    else
    {
        $books = Book::with(['author', 'category'])
          ->where('name', 'like', '%' . $request->book . '%')
          ->orWhere('isbn', $request->book)->paginate(18);
    }

    $categories = Category::all();

    return view('books', compact(['books', 'categories']));
}
```

Obrázek 21 – Metoda zajišťující renderování nabídky knih

5.5.3 Detail knihy

Na detail knihy může zákazník přistoupit z nabídky knih kliknutím na nabízený odkaz. Detail knihy obsahuje všechny informace o knize, na rozdíl od stránky nabídky, která obsahuje pouze základní údaje, jako je název, cena, autor a kategorie. Stejně jako z nabídky, i z detailu knihy lze knihu přidat do košíku.



Obrázek 22 – Obrazovka detailu knihy

5.5.4 Košík

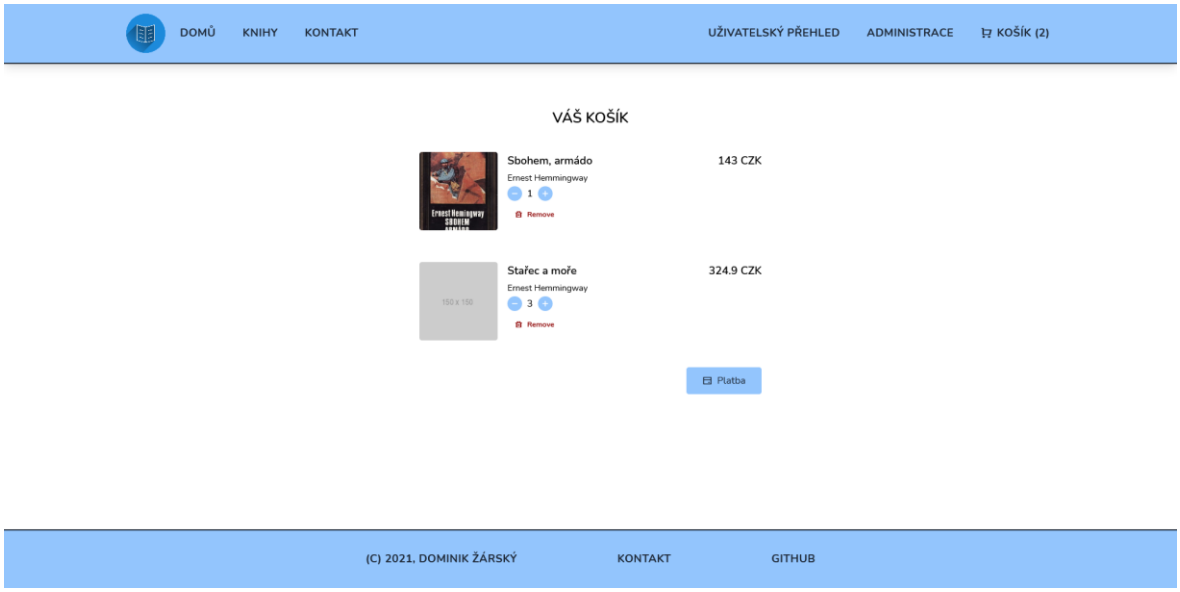
Vzhledem k dynamické povaze komponenty košíku, tedy, nutnosti dynamicky měnit jeho obsah, ať už při operacích, jako je přidání knihy do košíku, její odstranění nebo změna počtu objednávaných kusů knih, je vhodné zajistit, aby se po každé změně stavu košíku nemusela strana v prohlížeči „refreshovat“. Za tímto účelem byl pro vytvoření košíku použit Laravel Livewire, jež byl již popsán v teoretické části práce. Jakožto celek je košík rozdělen do čtyřech hlavních komponent, těmi jsou *Status košíku*, *Tlačítko košíku*, *Položka košíku,* a nakonec i samotný *Košík*, ve kterém je většina těchto komponentů koncentrována. Komponenta *Status košíku* (v navrhovaném řešení označená jako *Cart Status*), je přítomna v hlavním navigačním menu stránky, tudíž je dostupná ve všech součástech hlavní strany aplikace. Tato komponenta slouží jako klikatelný odkaz na zobrazení košíku, taktéž ale zobrazuje aktuální počet položek v košíku, čehož je docíleno pomocí *emitování eventů* z jiných komponentů, v tomto případě z komponentu tlačítka košíku. *Tlačítko košíku* je dostupné u jednotlivých knih v nabídce i v košíku samotném. Zatímco u nabídky knih toto tlačítko slouží k přidání této knihy do košíku, v košíku je možné z něj pomocí tohoto tlačítka knihy odstraňovat, případně měnit jejich počet. Rozličné funkcionality tohoto tlačítka je docíleno aplikováním několika metod v ovladači komponenty, jako příklad lze uvést metodu *addItem()*, tato metoda zpracuje vstupní parametr z tlačítka, který reprezentuje ID knihy při jejím volání z Livewire komponenty na front-endu. Jako výstup této metody je provedena databázová operace, která vytvoří položku právě aktivního košíku. Nakonec tato metoda *emituje event*, který oznamuje ostatním komponentám, které na takový *event* vyčkávají, že se mají znovu vyrenderovat, neboť byly položky košíku aktualizovány. Kód uvedeného příkladu je k dispozici na Obrázku 23. Jak lze na tomto obrázku také pozorovat, v rámci Livewire komponentu bývají často definovány základní proměnné třídy, v tomto případě například *$buttonType* nebo *$price*. Tyto proměnné je možné nastavit přímo v tomto Livewire ovladači nebo jako proměnnou vstupující do ovladače z view příslušícího Livewire komponentu. Rovněž je možné přímo z view volat jednotlivé funkce daného Livewire ovladače, a to kupříkladu pomocí direktivu *wire:click*, který zajišťuje volání oné funkce po kliknutí na tlačítko.

```php
<?php

namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Cart as CartModel;
use App\Models\CartItem;

class AddToCart extends Component
{

    public $buttonType; // Can be listing or detail, for book listings and book detail pages
respectively
    public $price;
    public $bookId;
    public $cart;
    public $itemId;

    public function addItem($bookId)
    {
        $cart = CartModel::where('session_id', session()->getId())
          ->firstOrCreate(['session_id' => session()->getId()]);
        if ($cart->items->contains('book_id', $bookId))
        {
            $cart->items->where('book_id', $bookId)->first()->increment('count');
        }
        else
        {
            CartItem::create([
                'cart_id' => $cart->id,
                'book_id' => $bookId,
                'count' => 1
            ]);
        }
        $this->emit('addItem');
    }

    public function removeItem($itemId)
    {
        CartItem::find($itemId)->delete();
        $this->emit('removeItem');
    }

    public function countPlus($itemId)
    {
        CartItem::find($itemId)->increment('count');
        $this->emit('qtyChange');
    }

    public function countMinus($itemId)
    {
        CartItem::find($itemId)->decrement('count');
        $this->emit('qtyChange');
    }

    public function render()
    {
        return view('livewire.add-to-cart');
    }
}
```

Obrázek 23 – Kód Livewire komponentu zajišťující operace s produkty v košíku

Posledními součástmi této skupiny komponent jsou komponenty *Košík* (v navrhovaném řešení jako *Cart*) a *Položka košíku* (v navrhovaném řešení jako *Cart Item*). Tyto komponenty jsou vždy renderovány společně v rámci obrazovky košíku a zobrazují informace o obsahu košíku. Komponenta košík, jakožto nadřazená komponenta, je opět interaktivní a vždy

vyčkává na *Livewire event* smazání položky z košíku, na který reaguje opětovným vyrenderováním sebe sama tak, aby reflektovala nový stav položek košíku. Z obrazovky košíku je možné dále pokračovat na obrazovku založení objednávky.

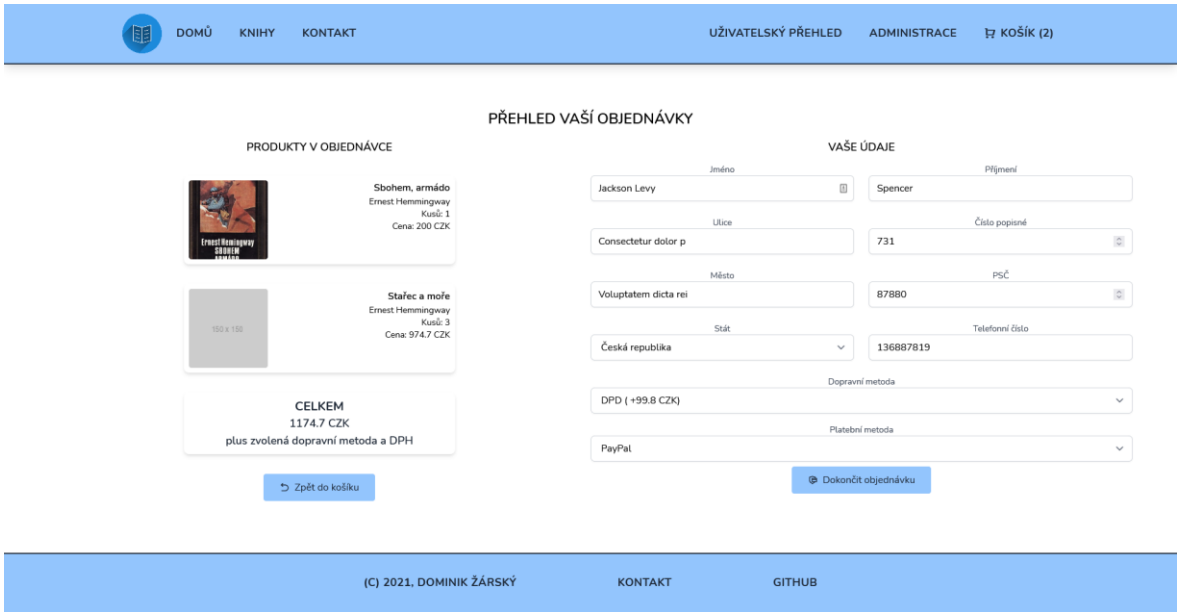


Obrázek 24 – Obrazovka košíku

5.5.5 Objednávka

Obrazovka založení objednávky je rozdělena na dvě sekce, sekce na levé straně obrazovky indikuje zákazníkovi stav jeho košíku, tedy všechny položky košíku i s jejich jednotlivými cenami s i bez DPH. Taktéž je nakonec zobrazena celková cena. V pravé části obrazovky pak zákazník zadává doručovací údaje, tedy své jméno a příjmení, adresu, telefonní číslo či email, taktéž zde vybírá způsob platby i dopravy. Dále aplikace nabízí možnost registrovat se do systému knihkupectví, což zákazníkovi umožní například i přístup k jeho minulým objednávkám, reklamacím, a podobně. Jakmile zákazník vyplní tento formulář, data putují do controlleru, kde je provedena řádná validace vstupů zákazníka a následně je objednávka uložena. Pokud zákazník vybral možnost, že si chce v aplikaci založit účet, je dále přesměrován na stránku registrace, kde bude propojen jeho zákaznický a uživatelský účet. Toto propojení probíhá na základě propojení databázových tabulek *User* a *Customer*, kde v tabulce *Customer* je na ID vytvořeného uživatele odkázáno pomocí cizího klíče. Na konci procesu objednávky by měl být zákazník odkázán na externí platební bránu, jak bylo koncipováno v objektovém návrhu aplikace, jímž se zabývá kapitola 4 této práce. Vzhledem k tomu, že tento prototyp nezpracovává reálné objednávky, a tedy ani platby, nebyl zde důvod snažit se do něj zakomponovat externí poskytovatele, jako je PayPal či

Stripe. Krok kontaktování platební brány je tedy v prototypu přeskočen a zákazník je v posledním kroku objednávky přesměrován na obrazovku stavu objednávky.



Obrázek 25 – Obrazovka založení objednávky

5.6 Administrace

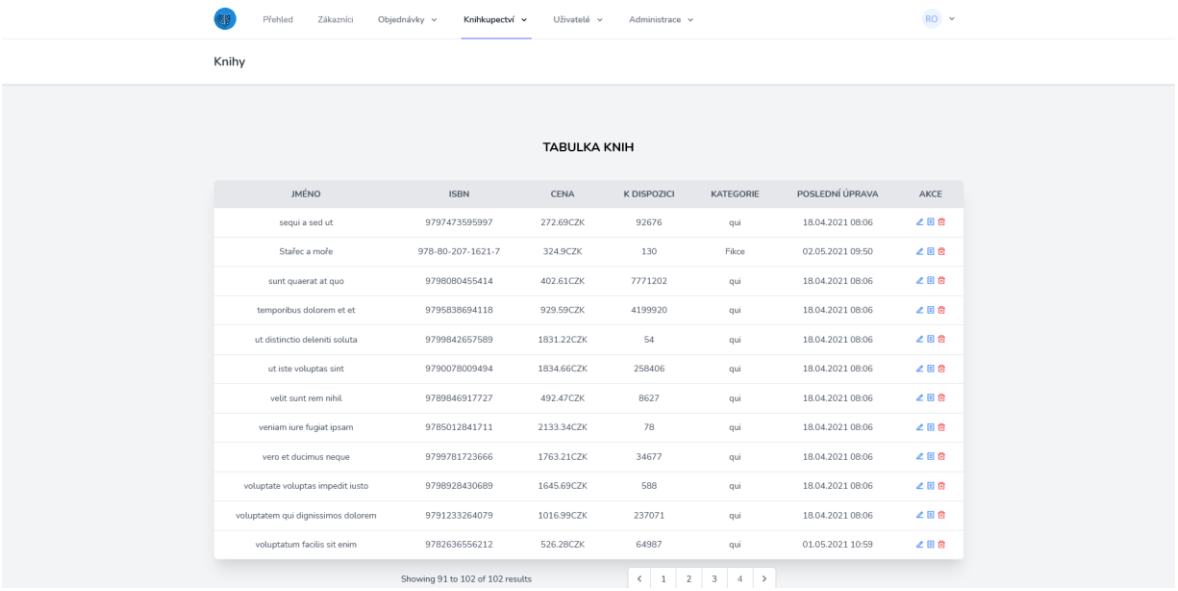
K administrační sekci aplikace mohou přistupovat pouze uživatelé s právy *Editor* nebo *Admin*. Tito uživatelé mají v rámci navrhovaného řešení ta největší privilegia. Omezení přístupu k této sekci je definováno pomocí *middleware*, který je přítomen na každé cestě, která má, co dočinění s administrací stránek. Skupina těchto cest je definována v souboru *routes/web.php* jako *Route group* s předponou cesty *admin*. Middleware zabraňující neoprávněnému přístupu k této skupině cest byl nazván *IsAdmin* a v rámci aplikace je registrován jako *auth.admin* middleware. Pro zajištění bezpečnosti administrativní sekce ověřuje, zda v databázovém záznamu přihlášeného uživatele existuje role typu 1 či 2 (tedy editor či admin), pokud ano, uživatel je vpuštěn do administrátorské sekce, v opačném případě je přesměrován zpět na hlavní stranu, totéž platí i v případě, kdy uživatel není přihlášen.

Administrátorská sekce obsahuje širokou škálu nástrojů pro zobrazování a správu různých součástí aplikace, jako jsou například operaci s uživateli, zákazníky, objednávkami, kontaktními formuláři, dostupnými knihami, a podobně. Front-endové zpracování administrátorské sekce je odlišné od hlavní strany prototypu, jelikož využívá základní šablonu knihovny Laravel Jetstream, využívající TailwindCSS a AlpineJS, obohacenou o vlastní komponenty. Na hlavní straně administrátorské sekce má administrátor k dispozici

přehled posledních deseti objednávek a seznam jemu přiřazených objednávek, reklamací či kontaktních formulářů k vyřízení.

5.6.1 Správa obsahu

Správa obsahu je dostupná pro uživatele administrátorského i editorského typu. Správa obsahu elektronického knihkupectví se váže především ke správě produktů, tedy knih, a dále autorů a kategorií. Všechen uvedený obsah je možno libovolně vytvářet, upravovat i mazat, všechny controllery starající se o tyto sekce jsou tedy opět CRUD controllery. Na Obrázku 26 lze pozorovat jednu z obrazovek správy obsahu zobrazující seznam aktuálně dostupných knih. Pro všechny strany pod administrátorskou sekcí aplikace obsahují obdobný seznam jim odpovídajících položek. Tento seznam vždy obsahuje navigační ikony, pomocí kterých je možné prokliknout na detail položky či možnost úpravy položky. Některé tabulky mají k dispozici i možnost výmazu záznamu, podobně jako ta na Obrázku 26.



Obrázek 26 – Obrazovka správy knih

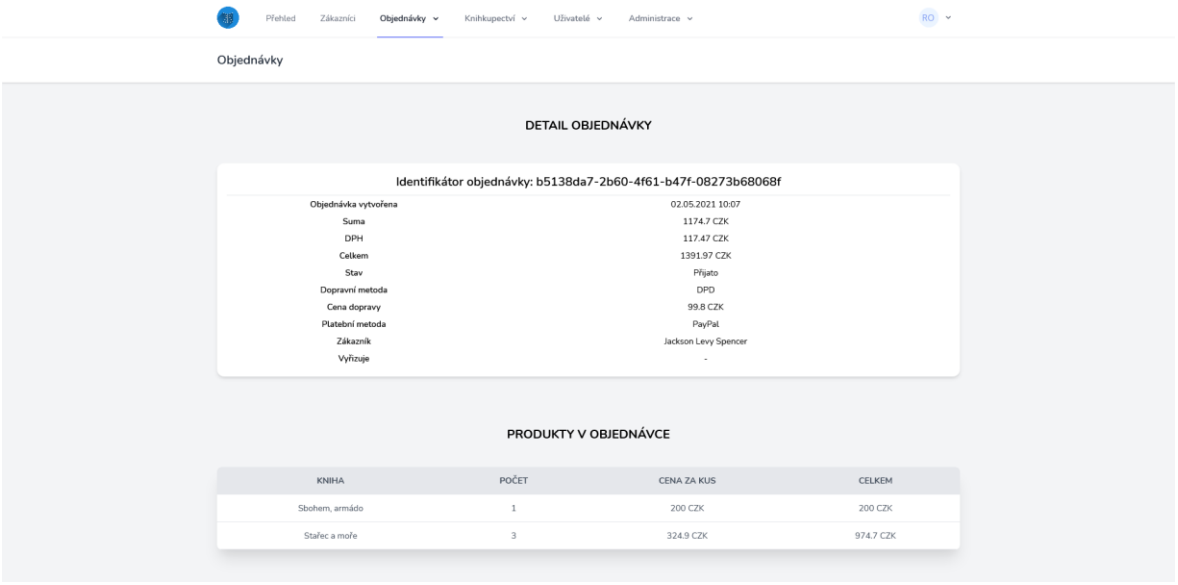
5.6.2 Reklamace

V rámci navrhovaného prototypu existují dva způsoby založení reklamací, prvním způsobem je založení vlastní reklamace zákazníkem v uživatelské sekci aplikace, podmínkou však je, že je zákazník registrován v systému knihkupectví. Zákazník může reklamaci či žádost o zrušení objednávky založit v nabídce vlastních objednávek, ve formuláři založení reklamace pak musí popsat důvod reklamace, která se následně odešle pro zpracování administrátorem. Druhá možnost založení reklamace je z administrátorské

sekce, kde přímo administrátor může založit tuto reklamaci. V této sekci může taktéž měnit stav reklamací, zobrazovat tento stav nebo reklamaci zrušit.

5.6.3 Objednávky

Po zadání objednávky na hlavní straně aplikace, jak bylo uvedeno v kapitole 5.5.5, je možné tyto objednávky zobrazit z administrátorské i uživatelské sekce aplikace. Uživatelé mají přístup pouze k informacím své objednávky, čehož je docíleno využitím UUID jakožto identifikátoru objednávky, stejně jako validací oprávněnosti přístupu uživatele k dané objednávce v příslušném controlleru. Naopak administrátor má přístup ke všem objednávkám, může měnit jejich stav, tyto objednávky vyřizovat nebo kontrolovat pomocí administrátorských nástrojů. Na Obrázku 27 je k dispozici náhled detailu objednávky, který obsahuje všechny informace o objednávce, včetně objednaných produktů.



Obrázek 27 – Obrazovka detailu objednávky

5.6.4 Kontaktní formulář

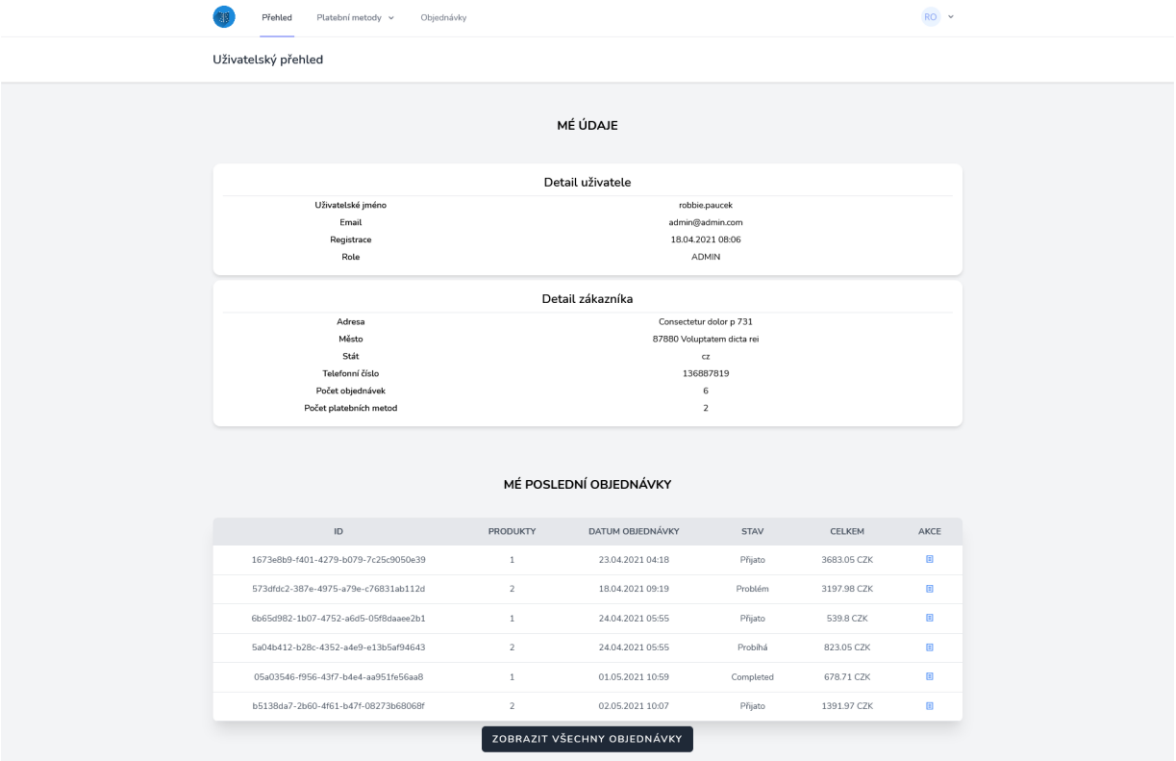
Kontaktní formulář má zákazník možnost vyplnit na hlavní straně aplikace v sekci *Kontakt*. Formulář se uloží do databáze a zobrazí se administrátorům v sekci administrace stránek. Kontaktní formulář obsahuje jméno a email osoby, která jej vyplnila, a zprávu. Pokud osoba zanechá email, může administrátor tuto osobu kontaktovat dále pomocí emailové komunikace. V administrátorské sekci je také možnost přiřazovat vyřízení kontaktních formulářů jednotlivým administrátorům a měnit stav kontaktního formuláře.

5.6.5 Reporting

Reporting by měl být obstaráván 3rd party softwarovým nástrojem, je však možné vyvinout vlastní softwarový nástroj v rámci navrhovaného systému knihkupectví, jakožto rozšiřovací modul. To by obnášelo úpravy na databázi a vytvoření přídavných modelů a controllerů, které by obstarávaly logiku reportingu. Jak již bylo zmíněno v předchozích částech práce, reporting by měl administrátorovi aplikace knihkupectví dodávat potřebné informace o návštěvnosti, stavu zboží, objednávkách a podobně, a to na denním, týdenním či měsíčním principu. Alternativně by měla být k dispozici možnost vygenerování reportu mimo tyto intervaly po manuální intervenci administrátora. Reporty by měly být vyjádřeny v číselné i grafické podobě a měly by upozorňovat na výkyvy, například výkyvy v návštěvnosti či možným skutečnostem, při kterých může například stav jednotlivých knih ve skladu dosahovat kriticky nízkého množství. Na tato upozornění by měl administrátor reagovat.

5.7 Uživatelská sekce

Uživatelská sekce nabízí registrovaným uživatelům možnost zobrazení vlastních objednávek, a to současných i minulých, zobrazení uchovávaných údajů o svém uživatelském i zákaznickém účtu, nebo například možnost založit reklamaci, případně žádost o zrušení objednávky. Tato sekce taktéž umožňuje uživateli smazat svůj uživatelský účet, měnit své heslo i osobní údaje, spravovat své platební metody, a podobně. Front-end této sekce odpovídá designu administrátorské sekce, založené na Laravel Jetstream. Na Obrázku 28 je vyobrazena uvítací obrazovka uživatelské sekce aplikace. Uživatel zde má k dispozici rychlý přehled posledních několika objednávek a informace o svých uživatelských i zákaznických údajích. Z této obrazovky může uživatel také zobrazit své další objednávky či platební metody. Z obrazovky objednávek může také založit reklamaci.



Obrázek 28 – Úvodní obrazovka uživatelské sekce

5.8 Lokalizace a další vlastnosti systému

Většina textových elementů, které jsou součástí navrženého prototypu, je tvořena tzv. *lokalizačními stringy*. Ty jsou součástí šablonovacího jazyku Blade a zajišťují jednoduchou možnost lokalizace celé aplikace. Součástí navrženého řešení jsou dvě použitelné jazykové mutace aplikačních prvků, tedy původní anglická lokalizace a následná česká lokalizace. Mezi nimi lze přepínat změnou hlavního jazyka aplikace v konfiguraci. Lokalizační soubory jsou k dispozici v adresáři *resources/lang*, zde je možné najít dvě metody lokalizace. Jednou z nich je překlad jednotlivých *lokalizačních stringů* v JSON (JavaScript Object Notation) souboru umístěném přímo v adresáři *lang*. Název tohoto souboru nese zkratku překládaného jazyka. Dále je možné lokalizovat některé prvky frameworku, jako jsou validační zprávy, stránkování záznamů, a podobně. Soubory zabývající se tímto typem lokalizace je možné nalézt v separátních složkách označených překládaným jazykem v adresáři *lang*.

Systém dále obsahuje některé parametry týkající se nabídky knih, prodejů a obalů nabízených knih. Jelikož údaje o měně, hodnotě DPH a podobné údaje jsou využívány napříč celou aplikací, bylo nutné snížit redundanci těchto údajů a případně zabránit inkonzistenci dat. K tomuto byla použita parametrizace pomocí systémových konfiguračních proměnných. Ty byly definovány v konfiguračním souboru */config/app.php*. Tato konfigurace obnáší již

zmíněnou měnu a hodnotu DPH, které byly v konfiguraci zapsány jako proměnné *currency* a *vat_pct*. Tyto proměnné je možné získat z jakékoliv části aplikace zavoláním funkce *config()*, která byla hojně využita. Z podobných důvodů, jako u měny a DPH, byla parametrizována i cesta k náhradnímu obrázku knihy, který je u reference knihy doplněn, pokud nebyl ke knize přidán obrázek. Cesta k tomuto náhradnímu obrázku byla definována v konfiguračním souboru */config/filesystems.php* jako *cover_photo_placeholder_path* a využívá se všude, kde probíhá kontrola dostupnosti obrázku knihy.

6 MOŽNOSTI ROZŠÍŘENÍ SYSTÉMU

Jelikož je navržený systém díky využití kompetentního webového frameworku do velké míry škálovatelný, rozšířit jej by nemělo činit veliký problém. Jednotlivá rozšíření by se mohly týkat hlavně interakce se zákazníkem, větší míru automatizace či propojení s jinými systémy.

6.1 Chat zákazníka s uživatelskou podporou

Mnoho knihkupectví v České republice neposkytuje ve svých internetových obchodech mnoho možností pro kontakt zákazníka s ním. Většinou bývá zákazník odkázán na emailovou adresu knihkupectví nebo na telefonní číslo, málokdy se na těchto stránkách objevuje funkční kontaktní formulář. Toto lze pozorovat i na stránkách příkladových českých knihkupectví uvedených v kapitole 1.3 teoretické části práce. Prototyp knihkupectví, navržený v kapitole 4 a vytvořený v kapitole 5 praktické části práce, obsahuje především onen kontaktní formulář jako preferovaný způsob komunikace zákazníka s knihkupectvím. Je však možné se inspirovat dalšími elektronickými obchody u nás i v zahraničí a zavést „živý chat“ zákazníka s podporou knihkupectví. Toto by ušetřilo zákazníkovi čekání na odpověď na email i peníze za telefonní hovor, pokud by byl nucen podporu knihkupectví kontaktovat telefonicky. Problémem by mohly být zvýšené výdaje na operátory tohoto live chatu ze strany provozovatele knihkupectví, které by jej zavedlo. Realizace by mohla proběhnou například pomocí použití JS knihovny *socket.io* [28] či jiné JavaScriptové knihovny obstarávající podobnou funkcionalitu – tedy real-time výměnu zpráv.

6.2 Generování faktur a jiných administrativních listin

Ačkoliv navržený systém eviduje všechny potřebné informace o objednávce i zákazníkovi, který ji založil, neumí k těmto objednávkám vygenerovat žádný report či fakturu. Toto je chyba, která by musela být v produkční verzi opravena. Nejjednodušším způsobem realizace této funkcionality by mohlo opět být využití 3rd party nástrojů pro generování faktur, jako jsou v případě tuzemských produktů například *Vyfakturuj*, *Superfaktura* či *IDoklad*.[29] Tyto aplikace by měly být pro systém knihkupectví dostačující, avšak je nutné si za ně připlatit. Alternativou je vytvoření vlastního systému na generaci faktur, opět v rámci rozšíření již vytvořené aplikace knihkupectví. Tento systém by využíval data o objednávkách uložených v databázi knihkupectví, ze kterých by následně generoval faktury.

Export těchto faktur do formátů, jako je PDF, či případně CSV a dalších používaných formátů, by mohlo být dosaženo instalací dalších Composer balíků zajišťujících kýženou funkcionalitu. Na stejném principu by mělo fungovat i generování jiných dokumentů, jako mohou být například reklamační listy a jiná administrativní dokumentace. Faktury a další dokumenty by měly po zavedení tohoto rozšíření být dostupné zákazníkovi i administrátorovi v příslušných sekcích aplikace, případně u příslušných objednávek.

6.3 API

Vytvoření privátní API pro potřeby interního systému knihkupectví je další vhodnou možností rozšíření hlavní aplikace. Vytvoření API v Laravelu je velmi snadné, jelikož tento framework obsahuje nástroje pro tvorbu API endpointů již v základní konfiguraci, tedy bez nutnosti instalace dalších balíků. Jak již bylo uvedeno dříve, autentizace a autorizace v rámci API by byly řešeny pomocí balíku Laravel Sanctum, který byl pro tento účel vytvořen. Jelikož by tato API nebyla primárně určena zákazníkům, ale administrátorům systému knihkupectví, dodavatelům či dalším subjektům, které s knihkupectvím spolupracují, obnášela by i zahrnutí endpointů pro CRUD operace. To by mohlo sloužit například administrátorům systému pro jednoduchý vklad nových produktů či úpravu produktů starých bez nutnosti vždy otevírat webový prohlížeč a přihlašovat se do systému knihkupectví k provedení změn na produktech. Zavedení této API by mohlo mít za důsledek například i větší míru automatizace procesů v rámci správy knihkupectví.

6.4 Prodej elektronických knih a audioknih

Prodej elektronických knih a audioknih je v současné době velmi rozšířený zejména v zahraničí. Z příkladových českých knihkupectví rozebíraných v první části práce prodává tyto produkty pouze jedno. Proto by bylo dobré, aby navrhované knihkupectví bylo rozšířeno právě o e-knihy a audioknihy. Toto by obnášelo nutnost zavedení systému pro evidenci nákupu takovýchto produktů zákazníky, aby bylo zajištěno bezproblémové vyzvedávání těchto elektronických zdrojů zákazníkem. Při zavedení tohoto rozšíření by také bylo nutné rozšířit úložnou kapacitu systému knihkupectví a posílit robustnost systému, protože by se dal očekávat nárůst požadavků na servery při stahování těchto produktů ze serverů knihkupectví zákazníky.

ZÁVĚR

Tato práce se zaměřovala především na objektový návrh systému elektronického knihkupectví a jeho implementaci. V úvodu teoretické části práce byl shrnut aktuální stav na trhu elektronických knihkupectví v České republice i v zahraničí, byly popsány technologie, přístupy a softwarové nástroje, které je možné k tvorbě a provozu elektronického knihkupectví využít, taktéž byly stručně srovnány některé náležitosti jednotlivých přístupů k takovému úkonu. Dále byla popsána základní funkcionalita, kterou většina elektronických knihkupectví obsahuje. V rámci analýzy stavu na trhu českých a zahraničních knihkupectví byly rozebrány rozdíly mezi analyzovanými knihkupectvími v každé kategorii. V dalších kapitolách teoretické části byly rozebrány nástroje pro objektovou analýzu aplikace a její následnou realizaci, byly probrány rozličné aspekty jazyka UML, který byl dále v praktické části použit k objektovému návrhu aplikace knihkupectví. V teoretické rovině bylo rozebráno například inženýrství požadavků, vztahy v jazyce UML, specifika diagramu tříd, a podobně. Následně byly shrnuty technologie, které byly použity pro tvorbu prototypu aplikace v praktické části. Byla popsána především funkcionalita PHP frameworku Laravel, jakožto hlavního nástroje pro tvorbu onoho prototypu aplikace elektronického knihkupectví. Tato kapitola se zaměřovala především na základní vlastnosti Laravelu, jako je využití MVC architektury, Eloquent ORM a práci s databází obecně, routing a zmíněna byla i bezpečnost samotného frameworku.

V praktické části byl proveden objektový návrh aplikace za užití jazyka UML a jeho nástrojů rozebraných v teoretické části. Byly navrženy funkční a nefunkční požadavky na systém, z nichž byly vytvořeny případy užití systému. Realizace některých z nich byla později názorně popsána za pomoci scénářů případu užití a aktivitních diagramů. V rámci návrhu byl vytvořen i diagram tříd, od kterého se následně odvíjel návrh databázové struktury databáze a specifika modelů a controllerů v rámci MVC architektury prototypu aplikace. Po vypracování objektového návrhu byl vytvořen prototyp aplikace za použití frameworku Laravel a MySQL databáze na back-endu. Při tvorbě prototypu se podařilo splnit všechny v návrhu vytyčené požadavky a případy užití. Ačkoliv tento prototyp pravděpodobně není v současné době vhodný pro produkční nasazení, je funkční a odpovídá všem východiskům provedeného objektového návrhu aplikace. Poslední část práce se zaměřovala na možnosti rozšíření funkcionality navrženého knihkupectví. V rámci této kapitoly byly zmíněny možnosti zavedení například *živého chatu* zákazníka se zaměstnancem knihkupectví pro jednodušší řešení problémů při nákupu či obecných dotazů.

Vypracování této bakalářské práce bylo pro jejího autora velmi přínosné, jelikož rozšířilo jeho znalosti, a to v oblasti užití softwarové analýzy, rozšíření povědomí o možnostech jazyka UML a vyzkoušení práce na větším projektu za použití frameworku Laravel. V rámci tvorby práce byly splněny všechny cíle, které byly vytyčeny v jejím úvodu.

SEZNAM POUŽITÉ LITERATURY

[1] IndieCommerce. *IndieCommerce and IndieLite: ABA's eCommerce Solutions for Independent Booksellers* [online]. [cit. 2021-03-07]. Dostupné z: https://www.indiecommerce.org/

[2] E-commerce platform comparison: web store software at a glance. *IONOS.com* [online]. [cit. 2021-03-07]. Dostupné z: https://www.ionos.com/digitalguide/online-marketing/online-sales/e-commerce-platforms-a-software-comparison/

[3] Comparing Vendor-Based and Custom eCommerce Solutions: Which One Is Right for You? *OroCommerce* [online]. July 23, 2020 [cit. 2021-05-01]. Dostupné z: https://oroinc.com/b2b-ecommerce/blog/ecommerce-platform-vs-custom-ecommerce/

[4] ECommerce Programming: The Best Language To Develop an ECommerce Website. Fireart [online]. [cit. 2021-03-07]. Dostupné z: https://fireart.studio/blog/ecommerce-programming-the-best-language-to-develop-an-ecommerce-website/

[5] Knihkupectví Kosmas. Kosmas.cz [online]. [cit. 2021-03-07]. Dostupné z: https://www.kosmas.cz

[6] What is A Shopping Cart? BigCommerce.com [online]. [cit. 2021-03-07]. Dostupné z: https://www.bigcommerce.com/ecommerce-answers/whats-shopping-cart/

[7] WILSON, Ralph F. How Does Shopping Cart Software Work? Practical ECommerce [online]. 15.1.2004 [cit. 2021-03-07]. Dostupné z: https://www.practicalecommerce.com/cart_intro

[8] Amazon Books. Amazon Books [online]. [cit. 2021-03-07]. Dostupné z: https://www.amazon.com/amazon-books

[9] The Top 20 Online Bookstores (and all the rest). BookSliced [online]. [cit. 2021-03-07]. Dostupné z: https://booksliced.com/books/here-are-the-20-best-websites-to-use-next-time-you-shop-for-paperback-and-hardcover-books-online/

[10] ARLOW, Jim a NEUSTADT, Ila. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Brno: Computer Press, 2007. s. [1a]. ISBN 978-80-251-1503-9. Dostupné také z: https://ndk.cz/uuid/uuid:79467ad0-b769-11e6-8c54-5ef3fc9ae867

[11] SEIDL, Martina, Marion SCHOLZ, Christian HUEMER a Gerti KAPPEL. UML @ Classroom: An Introduction to Object-Oriented Modeling. 1. New York: Springer International Publishing, 2015. ISBN 978-3-319-12742-2.

[12] PILONE, Dan a Neil PITMAN. UML 2.0 in a Nutshell. 1. Beijing; Sebastopol, CA: O'Reilly Media, 2005. ISBN 978-0596007959.

[13] KRAVAL, Ilja. Analytické modelování informačních systémů pomocí UML v praxi. 1. Brno: Object Consulting, 2010. ISBN 978-80-254-6986-6.

[14] Laravel – Overview. *Tutorialspoint* [online]. [cit. 2021-04-18]. Dostupné z: https://www.tutorialspoint.com/laravel/laravel_overview.htm

[15] MVC Framework – Introduction. *Tutorialspoint* [online]. [cit. 2021-04-18]. Dostupné z: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

[16] Database: Getting Started. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/database

[17] Database: Query Builder. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/queries

[18] Eloquent: Getting Started. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/eloquent

[19] Blade Templates. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/blade

[20] Artisan Console. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/artisan

[21] Testing. Silex – *The PHP micro-framework based on the Symfony Components* [online]. [cit. 2021-04-18]. Dostupné z: https://silex.symfony.com/doc/2.0/testing.html

[22] HTTP Tests. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/http-tests

[23] Routing. *Laravel* [online]. [cit. 2021-04-18]. Dostupné z: https://laravel.com/docs/8.x/routing

[24] MANSURI, Shadid. Why Laravel is the Recommended Framework for Secure, Mission-Critical Applications. *Auth0 Blog* [online]. June 28, 2018 [cit. 2021-04-18].

Dostupné z: https://auth0.com/blog/why-laravel-is-the-recommended-framework-for-secure-mission-critical-applications/

[25] Livewire. *Laravel Jetstream* [online]. [cit. 2021-05-01]. Dostupné z: https://jetstream.laravel.com/2.x/stacks/livewire.html

[26] Biblys/isbn: PHP library to validate and convert ISBNs and EANs. *Github* [online]. [cit. 2021-05-01]. Dostupné z: https://github.com/biblys/isbn

[27] WAGNER, Lane. An Introduction to Key Derivation Functions: Argon2, Scrypt, and PBKDF2. *Hackernoon* [online]. May 4th, 2020 [cit. 2021-05-01]. Dostupné z: https://hackernoon.com/an-introduction-to-key-derivation-functions-argon2-scrypt-and-pbkdf2-g23s32is

[28] LEWINGTON, Glyn. Everything You Need To Know About Socket.IO. *Ably Realtime* [online]. [cit. 2021-5-6]. Dostupné z: https://ably.com/topic/socketio

[29] Faktury online – velké srovnání nástrojů na tvorbu faktur: Nejlepší nástroj pro faktury online 2021. *5Nej* [online]. [cit. 2021-5-6]. Dostupné z: https://www.5nej.cz/srovnani-fakturacnich-nastroju/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API – Application Programming Interface

CLI – Command Line Interface

CMS – Content Management Systém

CRUD – Create Read Update Delete

CSRF – Cross Site Request Forgery

CSS – Cascade Style Sheets

DDoS – Distributed Denial of Service

DPH – Daň z přidané hodnoty

HTML – Hypertext Markup Language

ISBN – Internation Standard Book Number

JS – JavaScript

JSON – JavaScript Object Notation

KDF – Key Derivation Function

MVC – Model View Controller

ORM – Object-Relational Mapper

PDF – Portable Document Format

PDO – PHP Data Objects

PHP – PHP Hypertext Preprocessor

RAM – Random Access Memory

RDBMS – Relational Database Management Systém

REST – Representational State Transfer

RQ – Requirement (požadavek)

SaaS – Software as a Service

SQL – Structured Query Language

SSL – Secure Sockets Layer

TLS – Transport Layer Security

UC – Use Case (případ užití)

UML – Unified Modeling Language

UP – Unified Process

UUID – Universally Unique Identifier

XSS – Cross Site Scripting

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH