FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Produktový katalog administrace e-shopu
Student:	Bc. Alois Kouba
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je ve spolupráci s Vojtou Benešem, Bc. Vítem Urbanem a Filipem Dubjakem realizovat a nasadit do použitelné formy více doménovou e-shop administraci navázanou na současnou databázi tak, aby byla paralelně provozovatelná s již existující administrací, kterou následně nahradí. Z důvodu velkého rozsahu a také aby byla zajištěna forma izolace od ostatních prací, je předmětem této konkrétní práce produktový katalog a jeho návazné procesy. Práce klade důraz na budoucí snadnou rozšiřitelnost, údržbu a provoz včetně efektivního využití technologií.
Postupujte v těchto krocích:
1. Navažte na analýzy řešené e-shop administrace a předešlých prací věnující se vývoji dané části projektu v backend i frontend oblasti. Zanalyzujte i doposud využívané technologie včetně možnosti aplikovat jazyk C#. Neopomeňte důkladnou analýzu zaměřenou na produktovou část administrace včetně jejích doprovodných procesů.
2. Na základě analýzy společně se zbytkem týmu navrhněte a zvolte vhodné technologie a společné postupy pro následnou realizaci.
3. Navrhněte vhodný celek frontend a backend produktové části administrace.
4. Návrhy řádně konzultujte se zadavatelem.
5. Implementujte kompletní a použitelnou produktovou část administrace.
6. Navrhněte a realizujte vhodné formy testů frontend a backend části vaší realizace.
7. Otestujte výslednou realizaci buď v ostrém provozu, nebo alespoň s reálnými uživateli.
8. Zhodnoťte výsledné řešení, navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 31. ledna 2024 v Praze.

Diplomová práce

PRODUKTOVÝ KATALOG ADMINISTRACE E-SHOPU

Bc. Alois Kouba

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
9. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především panu Ing. Jiřímu Hunkovi, a to jak za možnost si toto téma zvolit, tak za cenné rady a čas, které mi v průběhu tvorby práce poskytl. Poděkování také patří kolegům Vojtěchovi Benešovi, Bc. Vítu Urbanovi a Filipovi Dubjákovi za spolupráci při implementaci. Dále bych chtěl poděkovat Bc. Martinovi Dvořákovi za technické konzultace. V poslední řadě bych chtěl poděkovat také svojí rodině za podporu během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 9. května 2024

Abstrakt

Tato práce se zabývá vývojem webové aplikace administrace e-shopu. Tento vývoj probíhá v týmu společně se třemi dalšími kolegy. Práce je rozděleni do 4 hlavních částí. Nejprve probíhá analýza problémové domény s důkladným přihlédnutím k předchozím závěrečným pracem, které se touto problémovou doménou zabývaly. Dále je na základě této analýzy proveden návrh výsledné aplikace, která je následně implementována. Výsledná aplikace je otestována několika různými druhy testů.

Klíčová slova eshop, API, backend, .NET, ASP.NET, tým, frontend

Abstract

This thesis focuses on the development of a web application of e-shop administration. This development is conducted within a team made of myself and three other colleagues. The thesis is divided into four main parts. Firstly, this thesis deals with an analysis of the problem domain, with careful consideration given to previous theses that dealt with this problem domain. Based on this analysis, a design for the resulting application is created and subsequently implemented. The resulting application is tested using various types of tests.

Keywords eshop, API, backend, .NET, ASP.NET, team, frontend

Seznam zkratek

API	Application Programming Interface
MVC	Model View Controller
JSON	JavaScript Object Notation
REST	Representational state transfer
SQL	Structured query language
ORM	Object Relational Mapper
BE	Backend
FE	Frontend
HTTP	Hyper Text Transfer Protocol

Úvod

V digitálním věku se e-shopy stávají nezbytným prvkem ekonomického prostředí, nabízejícím obrovský potenciál pro růst a rozvoj podnikání. E-shopy vznikají nejen jako nezávislé entity, ale také jako doplňková služba již zaběhlým kamenným obchodům. S nárůstem objemu internetového obchodování se však zvyšuje i potřeba efektivní správy a administrace těchto elektronických obchodů. S těmito výzvami se potýkají podniky různých velikostí a odvětví.

Jedním z takových řešení je e-commerce platforma od společnosti Jagu s.r.o (dále jen Jagu). založená na systému Opencart, která obsahuje kompletní řešení pro více doménové e-shopy. Tento produkt je v provozu již od roku 2009 s několika e-shopy, a vyvstala tak potřeba nahradit jej za novější řešení.

Paralelně se mnou pracují na novém systému také kolegové Bc. Vít Urban, Vojtěch Beneš a Filip Dubjak, kde ačkoliv každý máme na starosti víceméně oddělené části systému, tak se bude potřeba zaměřit na zvolení vhodných postupů pro práci v týmu pro realizaci projektu.

Hlavním cílem této práce je tedy vytvoření moderního a efektivního systému pro administraci e-shopu, který bude schopen zvládat komplexní požadavky na správu zboží, objednávek, zákaznických účtů a dalších klíčových aspektů provozu e-commerce platformy. Konkrétně tato práce se zabývá především částí administrace, která se zabývá správou produktů. Jedním z hlavních požadavků je vzájemná kompatibilita s případně paralelně běžícím starým systémem. Nakonec je nutné výsledné řešení otestovat, a to nejen pomocí automatizovaných testů, ale také s reálnými administrátory.

Tato práce navazuje na několik prací, které se v posledních letech snažily o stejný hlavní cíl. Mimo jiné jsem se tímto zabýval i já ve své bakalářské práci *„Backend administrace e-shopu“*[1]. V podstatě žádnou z nich se však nepodařilo dotáhnout do podoby, ve které by se ji dalo nasadit pro reálné uživatele.

Diplomová práce je rozdělena na 4 hlavní části, které reprezentují jednotlivé etapy vývoje softwaru - analýza, návrh, implementace a testování.

Kapitola 1
Analýza

Analýza představuje klíčovou fázi v procesu vývoje nejen administrace e-shopu, ale obecně jakéhokoliv softwaru. Bez kvalitně provedené analýzy hrozí nejen nesplnění vytyčených požadavků, ale také vznik software, který sice splňuje zadané požadavky, ale není kvalitní a další práce na něm je nesnadná.

V této kapitole se tedy nejprve budu zabývat analýzou současného stavu upravené platformy OpenCart od Jagu. Dále shrnu dosavadní pokusy o její náhradu v režii několika sérií závěrečných prací. Následně budu pokračovat zavedením vhodných postupů vývoje v rámci našeho čtyřčlenného týmu. Nakonec vhodně navážu na a rozšířím analýzu produktové části z předchozích prací.

Celkově bude tato kapitola sloužit jako základní kámen pro návrh a implementaci nového administrativního systému e-shopu, poskytujícího efektivní řešení pro stávající potřeby a připravené na budoucí výzvy v oblasti e-commerce.

1.1 Existující řešení – OpenCart

Jak jsem již zmínil výše, současné řešení e-shopové platformy běží na systému OpenCart.

„OpenCart je zdarma dostupná open source e-commerce platforma pro online obchodníky. OpenCart poskytuje profesionální a spolehlivý základ, ze kterého lze postavit úspěšný internetový obchod. Tento základ oslovuje širokou škálu uživatelů; od zkušených webových vývojářů hledajících uživatelsky přívětivé rozhraní, až po majitele obchodů, kteří poprvé spouštějí svůj podnik online. OpenCart nabízí rozsáhlé množství funkcí, které vám umožňují pevně ovládat přizpůsobení vašeho obchodu. S nástroji OpenCartu můžete pomoci vašemu internetovému obchodu dosáhnout jeho plného potenciálu.“[2]

Konkrétně systém, který vyvíjí Jagu, běží již od roku 2009. Původně byl založen na verzi Opencart 1.1, což je dokonce první stabilní verze tohoto

systému psaná v jazyce PHP, vydaná v roce 2009.[3]

Ačkoliv je systém založen na prastarém Opencartu 1.1, tak od této verze prošel mnoha rozšířeními a úpravami. Nicméně forma, kterou byly některé úpravy provedeny, způsobila velký nárůst technologického dluhu.

„Technický dluh (také známý jako technický dluh nebo kódový dluh) popisuje situaci, kdy vývojové týmy přijímají opatření k urychlení dodání určité funkcionality nebo projektu, které později potřebují být refaktorovány. Jinými slovy, jedná se o výsledek prioritizace rychlého dodání před dokonalým kódem.“[4]

Toto v kombinaci s tím, že aplikace je monolitické architektury způsobilo, že údržba i další rozšiřování tohoto systému v současné podobě je obtížná. Tyto problémy nezasáhly jen aplikaci jako takovou, ale i databázi, která není v ideálním stavu. Konkrétně neobsahuje z historických důvodů na většině míst cizí klíče, některé tabulky a sloupce v tabulkách jsou nepoužívané a některé věci nejsou navržené správně. Nicméně, přes to všechno je systém funkční a obsahuje velké množství důležitých funkcí. Především tato rozsáhlost v kombinaci s nutností použít stejnou databázi působí problémy při pokusech o náhradu.

1.1.1 Komplikovanost existujícího řešení

V této podsekci seznámím čtenáře s některými konkrétními řešeními v existujícím systému, konkrétně těch, které se vážou k produktům. Toto seznámení má za cíl ilustrovat komplexitu systému, zvýraznit některé problémy, které bude potřeba vyřešit v rámci návrhu a realizace a je nutné pro pochopení návrhu aplikace v kapitole 2.

1.1.1.1 Multidoménovost

Celý systém spravuje ne jen jednu, ale větší množinu domén. Správa konfigurace domén je mimo rámec této práce, avšak nový systém bude umět pracovat na základě už uložené konfigurace v databázi. Každá doména je identifikována pomocí svého názvu a z pohledu produktového katalogu k ní lze přiřadit 3 hlavní informace, které ji charakterizují –

- Jazyk

  Jazyk je jeden ze dvou hlavních způsobů, kterým lze danou doménu asociovat s konkrétními daty. Druhým způsobem je přímo odkaz na konkrétní doménu, který se v případě existence používá prioritně.

- Měna

  Měna je atribut, který se v dosavadním systému administrace víceméně nepoužíval, byl důležitý především pro zákaznickou část e-shopu, která na jeho základě přepočítala ceny na základě definované měny a kurzu vůči měně výchozí.

- Stát

  Identifikátor státu, ve kterém doména existuje. Je důležitý pro asociaci s některými entitami, jako například s daňovými třídami.

V databázi se pak příslušnost k různým doménám definuje v rámci jednoho sloupečku konkrétní hlavní entity. Jinými slovy, M:N vztah mezi doménou a entitou je realizován jakožto řetězec obsahující jména jednotlivých domén oddělených speciálním oddělovačem. V případě produktů se pak k této vazbě váže i aplikační logika, která v případě nedefinovaných domén v entitě produktu aplikuje domény definované v entitě výrobce, který je k produktu přiřazen.

1.1.1.2 Skupiny domén

Multidoménovost systému také způsobila problémy, které se ukázaly postupně při implementaci požadavků od uživatelů systému. OpenCart od začátku počítal s tím, že domény spravované jednou instancí spolu budou úzce souviset. Konkrétně to znamenalo, že definice hlavní kategorie k produktu znamenala implicitní předpoklad, že je definována i pro všechny domény, na kterých se daný produkt nabízí. V opačném případě neměl na dané doméně produkt definovanou hlavní kategorii.

V rámci vývoje systému nicméně vyvstal požadavek provozovat několik speciálně zaměřených e-shopů, které prodávají pouze produkty jedné značky, a paralelně jednoho centrálního, který prodává produkty většiny výrobců. Tyto e-shopy mají definované různé kategorie ve stromové struktuře. Při provozování aplikace byl kladen důraz na dodržení vzájemné disjunktivity těchto stromů kategorií.

V důsledku toho došlo na implicitní rozdělení spravovaných domén na skupiny domén, které se vyznačují právě disjunktním stromem kategorii. Protože bylo požadováno produkt prodávat na doménách jedné nebo dvou skupin domén, byl do databáze přidán sloupec, který realizoval identifikátor „hlavní kategorie na druhé skupině domén“. Jinými slovy, došlo k realizaci M:N vazby pomocí dvou sloupců v hlavní tabulce, ne pomocí dekompoziční tabulky. Tento návrhový vzor pak komplikuje práci s produkty a validaci dat pro ně.

1.1.1.3 Doménové overridy

Vlivem multidoménovosti systému se požaduje definice různých dat pro různé domény. To můžou být například jazykové mutace, které se s danou doménou spojují na základě výše zmíněného atributu jazyka dané domény. Pro jazykové mutace obsahuje administrace relativně robustní podporu na databázové úrovni. Nicméně, ne všechna data jsou jazykovou mutací, a ne všechny domény se stejným jazykem by měly mít stejná jazykově závislá data.

Tento problém je řešen právě systémem overridů, nebo-li přepisovačů. Override je definován jako entita v databázi a obsahuje informace o tom,

kterým typem entity se zabývá, její identifikátor, identifikátor parametru, který přepisuje a hodnotu, kterou přepisuje. Tento způsob ukládání dat umožňuje teoreticky snížit objem uložených dat. Nicméně, jeho použití komplikuje business logiku v rámci aplikace a aplikaci zpomaluje.

1.1.1.4 Struktura databáze

Celý původní OpenCart funguje nad databází MySQL. Vznikl v době, kdy výchozím databázovým strojem pro MySQL byl MyISAM, a všechny jeho tabulky tak původně tento databázový stroj používaly. V roce 2009 byl MyISAM nahrazen databázovým strojem InnoDB jakožto výchozí stroj pro MySQL. Tato změna přinesla spoustu výhod –[5]

- Transakční zpracování

  InnoDB umožňuje použít transakce. Díky tomu lze manipulovat s více různými daty v rámci jedné transakce, příčemž v případě selhání dané transakce se změny neuloží. V případě MyISAM je pak v případě selhání některé části aplikační logiky změny ručně opravit zpět, jinak hrozí nekonzistentní stav databáze.

- Podpora cizích klíčů

  InnoDB podporuje na rozdíl od MyISAM cizí klíče.

  „Cizí klíč se používá k zabránění akcím, které by zničily spojení mezi tabulkami. Cizí klíč je sloupec (nebo soubor sloupců) v jedné tabulce, které odkazuje na primární klíč v jiné tabulce.“[6]

  Na cizí klíče bývají navázány i akce, které má databázový stroj provést v případě zneplatnění cizího klíče, vlivem například smazání dané entity. Konkrétně lze tento záznam smazat, lze cizí klíč nastavit na *null* hodnotu nebo provedení takové operace zakázat. V praxi to pak znamená, že databáze si sama hlídá konzistenci dat, usnadňuje tak práci programátorům a zefektivňuje vykonávání aplikační logiky.

Databázová struktura verze OpenCartu od Jagu již obsahuje aktualizaci většiny tabulek na stroj InnoDB, což je skvělá zpráva kvůli podpoře transakčního zpracování. Bohužel ale kvůli pracnosti této operace nebylo provedeno přidání cizích klíčů do schématu, vlivem čehož se v databázích můžou nacházet nevalidní data a aplikační logika s tímto stavem musí počítat. Je ale vhodné dodat, že nově přidané tabulky již cizí klíče obsahují.

1.1.1.5 Obrázky a soubory

Administrace umožňuje nahrávat soubory a dokumenty a následně je přiřazovat k entitám. Každá taková operace nahrání souboru ho uloží na server a k dané entitě přidá textový řetězec, který definuje cestu k danému souboru.

Tím dochází k denormalizaci schématu, protože při úpravě umístění, názvu nebo i jakékoliv složky v cestě k tomuto souboru je potřeba nový stav reflektovat na všech místech v databázi, kterých se tato změna dotkla.

1.2 Pokusy o náhradu OpenCartu

V této sekci popíšu postupně všechny pokusy o náhradu systému OpenCart, které proběhly v minulosti.

1.2.1 Migrace customizovaných e-shopů OpenCart 1.1

Historicky prvním pokusem o náhradu je diplomová práce Ing. Tomáše Nováčka z roku 2019. V rámci této práce proběhl pokus o přesun na novější existující řešení. Autor zvolil konkrétně platformu PrestaShop a provedl migraci části funkcionalit. Celé řešení bylo navrženo tak, aby bylo možné aktualizovat platformu automaticky. [7]

Na výslednou práci však nenavázal nikdo další – ani žádný student v rámci své závěřečné práci, ani nikdo z firmy Jagu.

1.2.2 Druhý pokus o náhradu systému

Dalším pokusem byly závěrečné práce Bc. Radomíra Koudely a Ing. Iuliie Evsenko. Tyto práce byly vypracovány v roce 2021 a oproti předchozímu pokusu se autoři rozhodli vydat cestou nové, nezávislé aplikace. Závěrečná práce Radomíra Koudely se zabývala návrhem API a přípravou backendu. Práce Iuliie Evsenko se měla zabývat pouze frontendovou částí, nicméně vlivem nedokončeného stavu backendu se zabývala i opravou potřebných míst na backendu. [8][9]

Ve výsledku se finální implementace ukázala jako nevhodná pro další vývoj, a tak se na tento pokus nikdo nepokusil navázat.

1.3 Projekt nové administrace – třetí pokus o náhradu systému

Třetím a v době psaní posledním pokusem o náhradu byly závěrečné práce Bc. Martina Dvořáka, Bc. Jana Babáka, Ing. Tomáše Hojka a moje vlastní bakalářská práce. V tomto týmu jsme zvolili relativně podobný přístup jako práce předchozí. Opět vznikala nová, nezávislá aplikace a opět bylo zvoleno rozdělení na frontendovou a backendovou část. Naděje byla, že zdvojnásobením pracovní síly se podaří projekt dotáhnout do podoby, na kterou bude možné navázat. [1][10][11][12]

Na tuto snahu bylo i navázáno v rámci dalších závěrečných prací i specializovaných předmětů na fakultě. V této sekci se podívám na současný stav projektu a provedu analýzu v rámci těchto prací vzniklých implementací. V ideálním případě bude možné na tyto implementace navázat. Budu se tedy věnovat postupně oběma částem – backendu i frontendu – a postupně zhodnotím jejich stav.

1.3.1 Projekt Porcupine – backend

Ze zmíněných 4 prací se implementací backendu zabývala práce má a Ing. Tomáše Hojka.[1][10]

Výsledkem byla implementace používající jazyk PHP a framework Symfony. Výsledek z hlediska funkčnosti dosáhl definovaných požadavků. Musím však sportovně přiznat, že tohoto stavu dosáhl až příliš pozdě pro hladkou spolupráci s frontendovým týmem. Navíc byl kód zatížen významným technologickým dluhem.

Nicméně, stav projektu byl dostatečně dobrý, aby na jeho implementaci navázali Bc. David Mareš a Bc. Nikita Golmgren. První jmenovaný se zabýval modulem nákupního košíku a druhý jmenovaný se zabýval automatizaci některých podpůrných procesů.[13][14]

Obě práce dospěly do úspěšného cíle, nicméně odhalily některé problémy existujícího řešení. Například Nikita Golmgren v závěru zhodnotil –

„Tuto práci považuji za hodně náročnou. Hlavní překážkou pro mě bylo za prvé pochopení celého systému – kvůli absenci dokumentace a těžce pochopitelným kódu, a za druhé technické problémy ve vývojovém prostředé (jak bylo popsáno v 2.10).“[13]

V rámci prvotní analýzy bylo vedoucím za firmu Jagu navrhnuto použití jazyka C#. Jagu tento jazyk začala relativně nedávno využívat pro své novější projekty a tento jazyk byl označen za preferovaný. To nicméně automaticky nediskvalifikuje rozšíření projektu Porcupine – jazyk PHP firma stále používá u více projektů a má s ním bohaté zkušenosti. Refactoring problematických částí kódu je tak stále validní možností. Pojďme se tedy podívat na porovnání těchto dvou jazyků.

1.3.1.1 Jazyk PHP

„PHP (rekurzivní zkratka pro PHP: Hypertextový Preprocesor) je široce používaný open source skriptovací jazyk obecného určení, který je vhodný zejména pro vývoj webových aplikací a lze jej vkládat přímo do HTML. To, co PHP odlišuje například od klientem prováděného JavaScriptu je skutečnost, že kód je prováděn na serveru – generuje HTML, které je pak odesláno klientovi. Klient tedy obdrží stránku, ale nebude vědět, jaký je podkladový kód.“[15]

Projekt konkrétně využívá verzi PHP 8.1 a základní funkčnost je zajištěna frameworkem Symfony. Tento framework je složen z přenositelných komponent a je open source. Z hlediska architektury je založený na modelu MVC.[16]

1.3.1.2 Jazyk C#

„C# (vyslovovující se ”See Sharp”) je moderní, objektově orientovaný a typově bezpečný programovací jazyk. C# umožňuje vývojářům vytvářet mnoho typů bezpečných a robustních aplikací, které beží v rámci platformy .NET. C# má svoje kořeny v rodině jazyků C a bude okamžitě povědomý programátorům C, C++, Java a JavaScript.“[17]

V době psaní je nejnovější verzí platformy .NET 8. Tím, že je celý ekosystém okolo jazyka C# a platformy .NET podporován firmou Microsoft, nabízí vysokou míru standardizace. Nicméně, stále se jedná o open source.

Z hlediska implementace MVC aplikace nabízí framework ASP.NET Core. Ten je redesignem a reimplementací staršího ASP.NET. Mezi jeho výhody patří například dobrá integrovatelnost s dalšími produkty od Microsoftu, hlavně co se týče nasazení v cloudu, což je v dnešní době velmi často používaný model. Také nabízí vlastní in-house framework Blazor pro implementaci frontendu. Více k Blazoru v sekci 1.3.2. [18]

1.3.1.3 Závěr

Z hlediska srovnání jazyků jako takových je relativně složité vybrat vítěze. Jednotlivé jazyky mají vůči sobě nějaké výhody a nevýhody. Výhodou C# je například silné typování, které je bezpečnější a zabrání mnoha nedefinovaným chováním aplikace[19] a rychlejší runtime, nabízející vyšší rychlost provádění kódu. Oproti tomu PHP je jednodušší na použití a obsahuje pestřejší paletu knihoven.[20]

Ve finále je však mnohem důležitější stav frameworků, které pro implementaci použijeme. PHP má sice mnoho možností, třeba zmiňované Symfony, ale i frameworky jako Laravel nebo CakePHP, ale ve finále se zdá, že ASP.NET Core stejně pokryje veškeré požadavky kladené na projekt. Zároveň se mi líbí jeho silnější standardizace a jednotná dokumentace. Plusové body získává C# i díky preferenci ze strany zadavatele.

Posledním dílkem skládanky je zhodnocení stavu kódu. Projekt Porcupine již zpracovávaly 4 závěrečné práce a jedná se tedy o relativně rozsáhlý projekt.

Ukázalo se, že objem práce potřebný k refaktoru by byl příliš velký.

Pro implementaci backendu jsme tak zvolili implementaci nového řešení v jazyce C#. Byl také zvolen název projektu, který budu dále v práci používat – Pangolin.

1.3.2 Projekt Kangaroo – frontend

Na projektu Kangaroo pracovali tedy zmínění Bc. Martin Dvořák a Bc. Jan Babák.[12][11]

Vývoj probíhal paralelně s projektem Porcupine. Napsaný je v typescriptu[21], což je v podstatě vylepšení jazyka JavaScript o silné typování, a používá framework Vue[22]. Vzhledem k výše zmíněným prodlevám při vývoji backendu, které musím jakožto jeho spoluautor sportovně přiznat jako reálné a pro kolegy Dvořáka a Babáka dosti problematické, vývoj frontendové části neprobíhal optimálním způsobem. Bylo například potřeba dělat změny v již dohodnutém rozhraní. Nicméně, navzdory tomu se podařilo vytvořit kvalitní návrhy jednotlivých částí aplikace a díky namockovaným datům i její část otestovat s reálnými uživateli.

Podobně jako u projektu Porcupine na implementaci bylo navázáno, a to v režii Bc. Martina Salaje[23], který zpracoval správu objednávek v tomto projektu. Jeho výsledek je vlastně velmi podobný, jako výsledek prvotní implementace – s uživateli otestovaný návrh, který však narazil na problémy u vývoje backendové části. Dále v projektu provedl aktualizaci z Vue 2 na Vue 3. To znamená použití tzv. Composition API, oproti staršímu Options API.

Options API je způsob psaní kódu, který se ve frameworku Vue.js používal od jeho počátku, až do verze 2.6. Spočívá v použití množiny atributů jako *data*, *metody* a *computed properties* ve strukturovaném formátu pro definici chování a stavu komponenty.[24]

Composition API je náhradou Options API od verze Vue 2.7. Velkou změnou je uvolnění této struktury, které umožňuje mnohem snažší a flexibilnější sdílení částí kódu mezi jednotlivými komponentami za použití funkcionálního a reaktivního programování.[24]

Bez zabíhání do zbytečných detailů jen konstatuji, že tento krok je správným směrem a usnadňuje nám další případný vývoj. Vzhledem k tomu, že pro BE byl zvolen jazyk C# a rozhodli jsme se implementovat jej od začátku, je na místě provést podobnou úvahu i u projektu Kangaroo. V návaznosti na to je nutné rozhodnout, zda budeme pokračovat ve Vue projektu, nebo použijeme framework Blazor pro implementaci nového frontendového řešení.

1.3.2.1 Vue.js

„Vue (vyslovuje se jako view) je JavaScriptový framework pro tvorbu uživatelských rozhraní. Staví na standardním HTML, CSS a JavaScriptu a

poskytuje deklarativní, komponentově orientovaný programovací model, který pomáhá efektivně vyvíjet uživatelská rozhraní libovolné složitosti.“[25]

Hlavním znakem tohoto frameworku je tzv. deklarativní rendering, který vlastně znamená, že se jednotlivé komponenty popisují pomocí HTML v závislosti na Javascriptovém stavu a na toto navazující zpracování reaktivity, kde Vue automaticky trackuje změny v onom Javascriptovém stavu a propisuje je do renderovaných komponent. Také je uživatelsky přívětivý.

1.3.2.2 Blazor

Blazor je frontendový webový framework, který umožňuje vývoj aplikací v prostředí .NET. Snaží se tak nabourat monopol, který mají v současnosti JavaScriptové aplikace. Umožňuje jak server-side rendering, tak čistě vytvoření frontendové aplikace napojené na nějaké rozhraní.

Podobně jako Vue je i Blazor založený na komponentách, ze kterých se postupně skládá celá aplikace. Hlavním rozdílem je pak právě použití programovacího jazyka C# pro správu logiky. Umožňuje však také volat JavaScriptový kód z .NET metod a naopak. Tím se stává relativně flexibilní a lákavou volbou.[26]

1.3.2.3 Závěr

Ve výsledku, podobně jako v případě srovnání PHP a C# je složité najít reálné hmatatelné výhody jedné technologie nad druhou. Ve výsledku je tedy hlavním faktorem opět preference vývojářů, případně zadavatele.[27] Jednou takovou výhodou Blazoru nad Vue je, že jeho použitím získáme jednotnou technologii pro vývoj, protože jsme se již rozhodli použít C# pro vývoj backendu. Snížíme tak náročnost implementace, protože nebude nutné aplikovat dvě různé technologie.

Nicméně, narozdíl od projektu Porcupine je Kangaroo v lepším stavu. Některé části jsou již i otestované s uživateli a bude možné je pouze přepojit na nový backend a použít znovu. Dalším faktorem k zhodnocení je, že zadavatel v tomto směru preferuje spíše použití existujícího Vue řešení, protože se shoduje s technologickým řešením u dalších projektů firmy.

Rozhodli jsme se tedy použít existující projekt pro implementaci problémové domény.

1.4 Práce v týmu

Diplomové (a případně bakalářské) práce velmi často studenti zpracovávají jako jednotlivci. Důvodů je pro to spoustu, zmíním třeba obtížnost rozdělení práce na vhodné části nebo také riziko, že se některá navazující práce nepovede a ovlivní tak vypracování závěrečné práce.

Nicméně, v případě tohoto projektu je zpracování pouze jednotlivcem vcelku nepraktické, protože rozsah problémové domény neumožňuje zpracovat dostatečně velkou část. Je tedy nutné pracovat ve více lidech, čímž se objem zvládnuté práce zvýší a utvoří tak celek, který bude už sám o sobě funkční a umožní někomu dalšímu, ať už studentovi v rámci svojí závěrečné práce, nebo reálnému zaměstnanci Jagu, na toto navázat a dokončit.

Touto problematikou jsem se již zabýval ve svojí bakalářské práci a zabývali se jí i všichni tehdejší členové týmu. Nebudu tedy znovu zbytečně opakovat jednotlivé provedené kroky, pokusím se ale na provedenou analýzu navázat, identifikovat problémy které nastaly a v reakci na ně navrhnout lepší postup. Nejprve se tedy podívám na to, jak vybraný způsob rozdělení práce v týmu ovlivnil finální výsledek projektu. Na základě této analýzy a analýz provedených v předešlých pracech budu rovnou prezentovat zvolené rozdělení práce a její postupy.

1.4.1 Analýza rozdělení práce v týmu v předešlých pracích

Jak jsem zmiňoval výše, jak kolegové Koudela s Iulii Evsenko[9][8], tak já s kolegy Dvořákem, Babákem a Hojkem[1][11][12][10], tak další navazující práce[14][23][13] vždy zpracovávaly buď frontendovou část, nebo backendovou část. A víceméně ve všech pracech zabývajících se frontendem se dá dočíst o stejném problému – zpoždení backendu a problémy v komunikaci mezi jednotlivými týmy. Konkrétně kolegové Dvořák a Babák napsali –

- *„FE byl oproti BE velmi napřed. I přes to, že jsme data mockovali, nebyla situace ideální a vedlo to ke zvýšení požadavků na nás oba. Já jsem navrhoval téměř celé API kategorií a správce souborů, protože jsem FE nechtěl výrazněji přepracovávat po dodání BE. S vývojem BE mám pouze malé zkušenosti, ani podrobněji neznám aktuální stav databáze. Návrh jsem prováděl z pohledu FE, jaká data potřebuji a nebral jsem ohledy na to, zda a v jakém formátu je BE schopný je poskytnout. Z těchto důvodů při implementaci BE došlo k výraznějším změnám v navrženém rozhraní a bylo nutné provést i některé změny ne FE. Například u kategorií došlo k zavedení skupiny domén.“*[11]

- *„Produkty jsou nejkomplikovanější stránkou, takže návrhy zabírají podstatně více času. Návrhy trvají výrazně déle než by měly – alespoň*

3 týdny. Toto zpoždění má vliv na celkový objem zvládnuté práce. Jako vývojář frontendu nemohu připojit svůj prototyp aplikace ani na mock-server, protože kvůli neznámé dokumentaci serveru nevím, jaké metody mám mockovat.“[12]

Z toho jasně vyplývá, že zvolené postupy nefungovaly správně. Jako hlavní problém lze vcelku jednoznačně identifikovat závislost jednotlivých týmů na sobě, v tomto případě závislost frontendu na backendu. Nicméně, v těchto závěrečných pracech se také nacházejí sekce zabývající se týmovou spoluprácí. Problematická komunikace je identifikována víceméně v každé této práci a v návrhu je pro tento problém navrženo řešení, které by ho mělo zvládnout. Třeba kolega Babák napsal –

„V našem projektu jsou vývojáři frontendu závislí na vývojářích backendu. Frontend komunikuje s backendem – získává od něj všechna data a posílá mu je zpět. Vývojáři frontendu by tak museli s vývojem funkcionality počkat do té doby, než pro danou funkcionalitu bude vytvořená podpora od backendu. To je velice nepříjemné a časově neefektivní. Chceme, aby oba dva týmy mohly vyvíjet současně. Řešení tohoto problému spočívá v mockování backendového serveru.“[12]

Praktickým řešením to však, jak napovídá i úryvek o kousek výše, nebylo. Ačkoliv byly pro vývoj definované postupy, které měly závislosti jednotlivých týmů na sobě řešit, tak ve finále selhaly, a podobný problém nastal i při spolupráci kolegů Koudely a Evsenko. Nezbývá tedy, než se z chyb minulých poučit. Nejefektivnější způsob zvládání závislostí mezi týmy je prostě je odstranit. To často nejde, nicméně v tomto případě se nabízí elegantní řešení – každý člen týmu zpracuje určitou část aplikace jako celku, tedy backendu i frontendu.

1.4.2 Zvolené technologie a postupy pro práci v týmu

1.4.2.1 Rozdělení práce

Jak jsem popsal v předchozí sekci, hlavním poučením z předchozích prací je minimalizace vzájemných závislostí jednotlivých členů týmu na sobě. V podstatě budeme tedy pracovat jako tzv. fullstack vývojáři.

„Fullstack webový vývojář je osoba, která dokáže vyvíjet jak frontend, tak backend software.“[28]

Aplikací tohoto přístupu můžeme podle[29] získat některé výhody –

- Identifikace problémů

  Fullstack vývojář má větší vhled do celkového fungování projektu a je v dobré pozici pro brzkou identifikaci problémů.

- Rychlá aplikace nových informací

  Protože se fullstack vývojáři zabývají širší problematikou, je pro ně snažší začít s prací na něčem pro ně do té doby neznámém.

- Méně závislostí mezi členy týmu

  Fullstack vývojáři vyžadují méně synchronizačních schůzek. Také můžou zpracovávat svoje úkoly jak na straně frontendu, tak backendu, což snižuje vzájemnou závislost mezi členy týmu.

  Samozřejmě, tento přístup má i svoje nevýhody[29] –

- Nedostatečná specializace

  Fullstack vývojář má široký záběr, avšak pravděpodobně nebude expertem ve všech oblastech frontendu a backendu.

- Problematické trackování postupu

  Snížením objemu delegované práce se stává složitějším sledovat, kdo dělá co.

1.4.2.2 Role v týmu

Navazujícím tématem na rozdělení práce je rozdělení rolí v týmu. Přirozeně, všichni v rámci týmu jsme jako vývojáři. Nicméně, byly definovány ještě role týmového manažera a seniorního vývojáře.

- Seniorní vývojář

  „Seniorní vývojáři mají více času a zkušeností v dané problémové doméně. Jsou schopni plánovat a provádět i složité projekty. Také jsou schopni pomáhat svým kolegům.“[30]

- Týmový manažer

  „Týmový manažer vede a řídí tým softwarových inženýrů a vývojářů.“[31]

V případě našeho týmu se týmovým manažerem stal Vojtěch Beneš, a povinnosti z toho plynoucí navíc jsou především organizace schůzek a komunikace v rámci týmu. Pozici seniorního vývojáře jsem zastal já a to hlavně z důvodu, že již do problémové domény mám určitý vhled a můžu tento vhled předat kolegům v týmu. Jelikož jsem ale před začátkem vývoje neměl žádné zkušenosti s vývojem v C# a ani s frameworkem Vue, tak jsem nezajišťoval technickou pomoc kolegům. Toto zajistil za firmu Jagu Bc. Martin Dvořák, který poskytoval případné technologické konzultace a prováděl code review. Code review je proces kontroly kódu jiným vývojářem, provedený jakožto základ schválení těchto změn.[32]

1.4.2.3 Metodika procesu vývoje software

Finálním kouskem skládanky jsou samotné postupy pro vývoj. Zde jsem neshledal nějaké problémy v předchozích pracech, a bez dalšího zdržení tedy jen zmíním výsledek, který jsme v týmu zvolili.

„Cyklus života vývoje softwaru (SDLC) je proces, který se využívá k návrhu, vývoji a testování softwaru. SDLC je sledován v rámci softwarového projektu v softwarové organizaci. Sestává z podrobného plánu popisujícího, jak vyvíjet, udržovat, nahrazovat a upravovat nebo vylepšovat konkrétní software. Cyklus definuje metodologii pro zlepšení kvality softwaru a celkový vývojový proces.“[33]

V našem případě jsme se rozhodli pro iterativní metodiku.

„Iterativní model životního cyklu se nesnaží začít s kompletní specifikací požadavků. Místo toho vývoj začíná specifikací a implementací pouze části softwaru, která je následně přezkoumána s cílem identifikovat další požadavky. Tento proces se pak opakuje, přičemž na konci každé iterace modelu vzniká nová verze softwaru.“[34]

Tento způsob jsme zvolili proto, že se hodí pro zpracování většího software, a také proto že se osvědčil v předešlých pracích. Kvalita návrhu a vhled do systému se totiž při každé další iteraci rapidně zlepšovaly.

1.4.2.4 Použité technologie

Podobně jako u předchozí sekce zde nebyly s žádným postupem zvoleným v minulých pracech shledány nedostatky. Bez dalšího zdržení budu tedy jen prezentovat konkrétní zvolené technologie.

- Sdílení kódu

  Pro sdílení kódu byl zvolen Gitlab. Všichni s ním máme zkušenosti, protože se používá v rámci fakulty. Konkrétně jsme pracovali na privátním gitlabu společnosti Jagu.

- Komunikace

  Hlavním komunikačním médiem byl text a kombinace online a offline schůzek. Zpočátku jsme se scházeli společně se zadavatelem, od letního semestru jsme však schůzky s týmem a s vedoucím rozdělili. Pro textovou komunikaci je použit platforma Slack. Dokonce se jedná o stejnou Slack doménu, kterou jsem použil v rámci své bakalářské práce. Ačkoliv Slack podporuje také funkci videohovorů, pro online schůzky jsme zvolili Google Meets. Tyto schůzky probíhají každý týden.

- Team management

  Zvoleným nástrojem je opět Redmine. Od letního semestru jsem také začal používat integraci se službou Toggl od firmy Jagu Timer2ticket. Zvolený

týmový vedoucí – Vojtěch Beneš – měl tak přehled nejen o úkolech zadaných jednotlivým členům, tak o čase na nich stráveném.

1.5 Sběr požadavků

Při vývoji softwaru je sběr požadavků kritickým prvkem, který má zásadní vliv na úspěch celého projektu. Důkladně definované a správně komunikované požadavky slouží jako základ pro následnou implementaci softwaru. Přestože tyto požadavky byly již sesbírány v rámci předešlých prací včetně té mojí, bez jejich opětovného sesbírání, definici a analýzy by tato práce nebyla kompletní.

Jako první je však potřeba definovat, co to vlastně budeme sbírat.

„Podle standardu IEEE 729 je požadavek definován následovně – “[35]

- *„Podmínka nebo schopnost, kterou uživatel potřebuje k řešení problému nebo dosažení cíle.“*
- *„Podmínka nebo schopnost, která musí být splněna nebo vlastněna systémem nebo jeho komponentou, aby vyhověla smlouvě, standardu, specifikaci nebo jiným formálně ustanoveným dokumentům.“*
- *„Dokumentované zobrazení podmínky nebo schopnosti jako v bodech 1 a 2.“*

V softwarovém inženýrství dále existují způsoby, jak požadavky dále dělit. To je užitečné, protože požadavky můžou být různého charakteru – některé můžou reprezentovat třeba nějakou věc, co má výsledek umět, jiné zase například formu, kterou je výsledek vytvořen. Obecně, to nejhrubší dělení je na požadavky funkční a nefunkční –

- Funkční požadavky

  „Požadavky, které zákazník specificky požaduje jako funkce, které by systém měl nabídnout. Všechny tyto požadavky by měly být ve výsledném systému zahrnuty. Reprezentovány jsou formou vstupu, který systém dostane, operace kterou vykoná a očekávaného výsledku. Jinými slovy, jsou to požadavky artikulované uživatelem, které přímo vidí ve výsledném produktu, na rozdíl od požadavků nefunkčních.“[36]

  Dobrým příkladem pro funkční požadavek může být třeba „Systém umožní vytvoření produktu pomocí formuláře.“

- Nefunkční požadavky

  „Jedná se o kvalitativní omezení, které systém musí splňovat na základě dohody. Jejich priorita se liší od projektu k projektu. Zabývají se problémy jako přenositelnost, bezpečnost, výkon a podobně.“[36]

  Konkrétním příkladem takového požadavku může být kupříkladu „Systém bude fungovat jako webová aplikace“.

Pro základní roztřídění požadavků je dělení na funkční a nefunkční požadavky dostatečné. Nicméně, pro konkretizaci daných požadavků je vhodné provést ještě detailnější rozdělení. Takovým rozdělením může být například

FURPS+. Tato zkratka dále dělí požadavky na funkční, použitelnostní, spolehlivostní, výkonostní a rozšiřitelnostní. Primárně cílí na rozdělení nefunkčních požadavků na podkategorie. Konkrétně –[37]

- F – Funkční

  „Zahrnuje hlavní funkce, které jsou známé v oblasti řešení, které se vyvíjí. Funkční požadavky mohou být také velmi technicky orientované. Mezi funkční požadavky, které by se mohly považovat za také architektonicky významné požadavky na celý systém, mohou patřit auditování, licencování, lokalizace, e-mail, online nápověda, tisk, reporting, bezpečnost, správa systému nebo pracovní postupy.“

  Víceméně se tak jedná o množinu funkčních požadavků podle klasického dělení na požadavky funkční a nefunkční.

- U – Použitelnost

  „Použitelnost zahrnuje zkoumání, zachycování a stanovení požadavků založených na otázkách uživatelského rozhraní jako je dostupnost, estetika rozhraní a konzistence.“

- R – Spolehlivost

  „Spolehlivost zahrnuje aspekty jako dostupnost, přesnost a obnovitelnost - například výpočty nebo obnovení systému po selhání.“

- P – Výkon

  „Výkon zahrnuje věci jako propustnost informací skrze systém, čas odezvy systému (což souvisí také s použitelností), doba obnovení a čas spuštění.“

- S – Rozšiřitelnost

  „Zde jsou specifikovány dalších požadavky jako je testovatelnost, adaptabilita, udržovatelnost, kompatibilita, konfigurovatelnost, instalovatelnost, škálovatelnost, lokalizovatelnost a tak dále.“

- \+ – Různá omezení

  „Umožňuje specifikovat omezení, včetně návrhových, implementačních, rozhraní a fyzických omezení.“

  Příkladem návrhového omezení může být například požadavek na relační databázi, implementační omezení může artikulovat například implementační jazyk.

Při sběru požadavků tedy požadavky rozdělím na funkční a nefunkční. Ke každému dále přípíši jeho klasifikaci v modelu FURPS+. Konkrétně se budu zabývat požadavky kladenými na část systému, kterou se zabývám, tedy produktový katalog.

1.5.1 Funkční požadavky

Funkční požadavky jsou z velké části dané systémem OpenCart. Několik požadavků však funkčnost původního systému rozšiřuje. Takové požadavky jsou označeny popisem.

- **FR1 – Zobrazení přehledu produktů [F]**

  Aplikace bude obsahovat stránku, která zobrazí seznam všech produktů.

- **FR2 – Filtrování v přehledu produktů [F]**

  V rámci přehledu produktů bude uživateli umožněno vyhledávat podle různých pokročilých filtrů.

- **FR3 – Smazání produktu [F]**

  Systém umožní smazání produktu z přehledu produktů.

- **FR4 – Úprava viditelnosti v rámci přehledové tabulky [F]**

  V rámci přehledové tabulky produktů bude uživateli umožněno upravovat, která data jsou mu zobrazena.

- **FR5 – Zobrazení detailu produktu [F]**

  Uživatel si může zobrazit všechna relevantní data k danému produktu. Tato stránka se zároveň očekává ve formě formuláře, v němž může daná data upravovat.

- **FR6 – Vytvoření produktu [F]**

  Systém umožní vytvořit nový produkt.

- **FR7 – Úprava produktu [F]**

  Systém umožní upravit existující produkt.

- **FR8 – Úprava produktových parametrů [F]**

  Systém umožní uživateli upravit parametry, které se k produktu vážou. Úprava těchto dat se nachází v produktové sekci, ale její zpracování má na starosti Filip Dubjak ve své paralelně vznikající bakalářské práci.

- **FR9 – Úprava slev produktů [F]**

  Systém umožní upravit cenu pro konkrétní zákaznické skupiny, doménu a produkt.

- **FR10 – Úprava variant produktu [F]**

  Systém umožní uživateli vytvářet, upravovat a mazat varianty produktu. Variantou se myslí například velikost bot.

- **FR11 – Jednotné tlačítko pro uložení [F]**

  Systém bude mít v produktové sekci tlačítko, které při použití upraví data pro produkt ve všech místech – obecná data, parametry, slevy a varianty.

- **FR12 – Primární doména [F]**

  Systém umožní definovat jednu z domén definovaných pro daný produkt jako výchozí. Funkce výchozí domény spočívá v tom, že data definovaná pro tuto doménu jsou využita jako základ, který nahradí případně neexistující data pro jinou doménu. Tento požadavek rozšiřuje funkčnost starého systému administrace.

- **FR13 – Zadávání cen v měně definované pro danou doménu [F]**

  Cena produktu se bude zadávat v měně, která je nastavena pro danou doménu. V dosavadním systému se měna zadávala pouze v korunách a administrátor tak musel přepočítávat tuto cenu ručně. Tento požadavek rozšiřuje funkčnost starého systému administrace.

- **FR14 – Správa souborového systému [F]**

  Systém umožní spravovat souborový systém spojený s daným e-shopem. Umožní do něj nahrávat obrázky a dokumenty, a ty následně přidávat k výrobcům, kategoriím, produktům a všem dalším entitám, které tyto soubory definují.

- **FR15 – Validita dat [F]**

  Aplikace neumožní uložení nevalidních dat do databáze a adekvátně upozorní uživatele, že nastal problém.

- **FR16 – Bezpečnost [F]**

  Systém bude přístupný pouze přihlášeným uživatelům. Backend bude podporovat autorizaci na základě rolí přihlášených uživatelů.

- **FR17 – Testování [F]**

  Systém bude otestován. Ideálně se bude jednat o automatické testy spouštěné v rámci pipeline Gitlab CI. Uživatelské rozhraní bude otestováno při testování s uživateli.

1.5.2 Nefunkční požadavky

- **NR1 – Architektura aplikace [S]**

  Aplikace bude rozdělena přinejmenším na backendovou a frontendovou část.

- **NR2 – Programovací jazyk [S, +]**

  Frontendová část systému bude využívat javascriptový framework Vue, backendová část bude využívat programovací jazyk C# a framework ASP.NET Core.

- **NR3 – Zpětná kompatibilita [S, +]**

  Systém bude pracovat s již existující databází MySQL. Dále bude potřeba zajistit, aby oba systémy byly schopné paralelního běhu po minimálně přechodnou dobu.

- **NR4 – Dokumentace [U, S]**

  Systém bude mít řádně zdokumentované rozhraní.

- **NR5 – Rozšiřitelnost [S]**

  Obsah práce neobsáhne celý existující systém. Implementaci bude tedy potřeba provést tak, aby byl výsledný kód v dobrém stavu a snadno rozšiřitelný.

Kapitola 2
Návrh

V rámci kapitoly Návrh se zaměřím na návrh systému administrace eshopu podle specifikované množiny funkčních a nefunkčních požadavků z předchozí kapitoly. Návrh bude rozdělen postupně na celkovou architekturu, backendovou část a frontendovou část. Toto pořadí není náhodné, odpovídá totiž tomu, jak bude systém postupně vznikat.

2.1 Celková architektura aplikace

Nejprve je tedy potřeba podívat se na systém jako na celek. Požadavky, které zasahují do architektury systému jsou –

- **NR1 — Architektura aplikace**

  Tento požadavek přímo udává, že systém se bude dělit na frontendovou a backendovou část. Vznikají tím tedy alespoň 2 komponenty systému.

- **FR16 — Bezpečnost**

  Tento požadavek specifikuje nutnost zabezpečení systému pomocí nějakého systému pro autentizaci a autorizaci na základě rolí. Tento požadavek ovlivňuje celkovou architekturu systému, protože jsem zvolil řešení pomocí samostatného autorizačního serveru. Tímto tedy vzniká třetí komponenta. Více v sekci 2.2.

- **NR3 – Zpětná kompatibilita**

  Tento požadavek pro celkovou architekturu systému definuje databázi. Bude použita stejná databáze MySQL, kterou používá stará administrace.

- **NR2 -– Programovací jazyk**

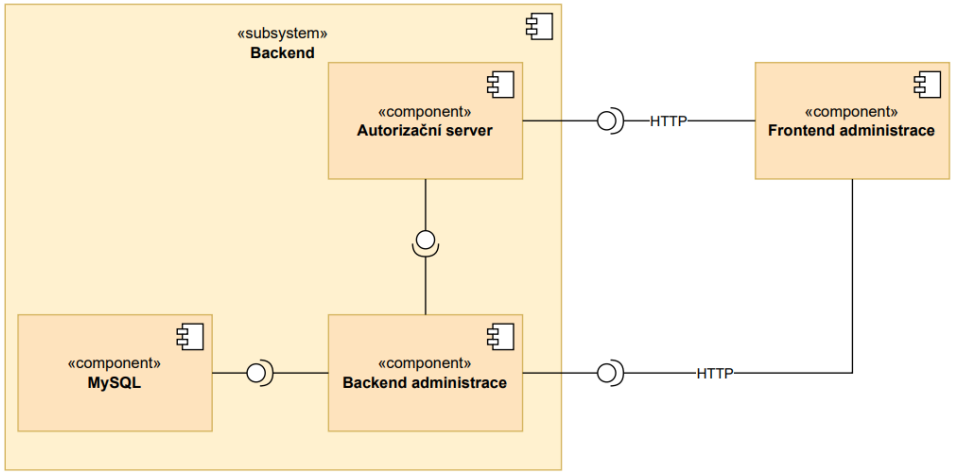
  Tento požadavek vyplývající z analýzy specifikuje, že pro backendovou část systému bude použit programovací jazyk C# a pro frontendovou část javascriptový framework Vue 3.

- **NR5 – Rozšiřitelnost**

  Tento poslední požadavek víceméně zasahuje do celého projektu, z hlediska architektury nicméně klade důraz na použití standardních postupů při jejím návrhu, které budou dobře pochopitelné pro další navazující programátory a budou dobře rozšiřitelné.

Výsledkem je, že aplikace se bude skládat ze 3 hlavních komponent – frontendové části, backendové části a autorizačního serveru. Celková struktura systému je zachycena na obrázku 2.1.

Obrázek 2.1 Architektura systému



2.2 Autorizační server

Jak jsem zmínil výše, pro realizaci autentizace a autorizace na základě uživatelských rolí bylo zvoleno řešení pomocí samostatného autorizačního serveru. Toto však nebyla jediná možnost, a rád bych alespoň krátce zmínil onu druhou možnost – zajištění autentizace a autorizace přímo v backendu, a proč jsem se rozhodl použít spíše variantu samostatného autorizačního serveru.

Nejprve je potřeba v rychlosti definovat, co vlastně autorizace a autentizace je a rozdíl mezi nimi. Často se totiž zaměňují.

„Autentizace je proces určení identity uživatele. Autorizace je proces určení, zda má uživatel přístup k danému zdroji.“[38]

Jinými slovy, autentizace zajišťuje pouze přihlášení uživatele a autorizace je až navazující proces, který určije co daný uživatel může provádět za operace.

Jediným požadavkem, který se týká autorizačního serveru je pak **FR16 — Bezpečnost**, který specifikuje nutnost použití rolí pro zajištění autorizace. To je víceméně základní variantou autorizace, tudíž tento požadavek nediskvalifikuje žádné možné řešení.

2.2.1 Varianta přímo v backendu

ASP.NET Core jakožto obsáhlý framework obsahuje mnoho různých způsobů, jak tuto problematiku řešit. Starý systém OpenCart autorizaci a autentizaci také řešil přímo, a prvním nápadem tak bylo použít řešení přímo v rámci backendu. To by mělo výhodu v tom, že je to z hlediska architektury stejné řešení jako OpenCart a uživatelské informace by byly sdílené mezi oběma dočasně paralelně provozovanými systémy na úrovni databáze.

Z hlediska reálného použití by to pak znamenalo použít ASP.NET Core Identity.

„ASP.NET Core Identity je API, které podporuje login pomocí UI a spravuje uživatele, hesla, uživatelská data, role, uživatelská tvrzení, tokeny, emaily a více.“[39]

Ten by vyžadoval vytvořit v databázi několik nových tabulek případně pohledů v místech, kde chceme sdílet data na úrovni databáze se starým systémem.

Velmi rychle se však ukázalo, že agregovat data z existujících tabulek pro OpenCart do těch pro ASP.NET Core by bylo problematické, protože některé potřebné sloupečky se v dané tabulce pro OpenCart nenacházejí. Víceméně od začátku jsem tak vyloučil možnost sdílení uživatelů mezi starým a novým systémem.

V důsledku tím zmizel hlavní důvod, proč toto řešení implementovat, a to mě společně s následujícími důvody přesvědčilo k použití separátního autorizačního serveru –

- Zkušenosti společnosti Jagu

Řešení s separátním autorizačním serverem již společnost Jagu nějakou dobu využívá u některých dalších projektů.

- Projekt nové administrace

  Projekty Kangaroo a Porcupine také využily separátního autorizačního serveru. Vzhledem k tomu, že Kangaroo budeme rozvíjet, bude možné využít již nakonfigurovaného řešení používajícího autorizační server Keycloak, které nám ušetří práci.

2.2.2 Autorizační server Keycloak

Nyní je tedy čtenář seznámen s důvody pro použití separátního autorizačního serveru. Ještě je však třeba definovat, co vlastně tento server dělá.

„Keycloak je jednotné řešení pro přihlašování pro webové aplikace a RESTful webové služby. Cíl Keycloaku je zabezpečit aplikace snadno, aby vývojáři mohli zabezpečit své nasazené služby a aplikace. Funkce, které vývojáři často musí implementovat sami jsou okamžitě dostupné a snadno přizpůsobitelné.“[40]

Konkrétně pro zajištění této funkcionality použijeme protokol OpenID Connect.

„OpenID Connect (OIDC) je protokol pro ověření identity, který je rozšířením otevřené autorizace (OAuth) 2.0 s cílem standardizovat proces ověřování a autorizace uživatelů při jejich přihlašování k digitálním službám.“[41]

„OAuth 2.0, což je zkratka pro ”Otevřená autorizace”, je standard umožňující webové aplikaci přístup k zdrojům hostovaným jinými webovými aplikacemi jménem uživatele. Nahradil OAuth 1.0 v roce 2012 a je nyní de facto průmyslovým standardem pro online autorizaci. OAuth 2.0 poskytuje souhlas k přístupu a omezuje akce, které klientská aplikace může provádět se zdroji jménem uživatele, bez nutnosti sdílet uživatelské přihlašovací údaje.“[42]

Tím, že delegujeme funkcionalitu ohledně autentizace na již hotové řešení, zbyde nám v rámci implementovaných řešení pouze správně napojit rozhraní poskytované Keycloakem. Interakce s autorizačním serverem pak probíhá následovně –

1. Uživatel přistoupí v klientské aplikaci na stránku vyžadující přihlášení, nebo se pokusí přihlásit ručně. Uživatel je pak přesměrován na přihlašovací stránku, kde se přihlásí. Konkrétní způsob přihlášení viz dále.

2. Po přihlášení je uživatel přesměrován zpět. Klientská aplikace obdrží alespoň přístupový token a id token, volitelně refresh token. Rozdíly mezi nimi popíši dále.

3. Klientská aplikace nyní ke každému požadavku poslanému na backend připojuje získaný access token v HTTP hlavičce Authorize.

4. Backend server ověří access token z hlavičky Authorize. Způsob ověření je různý v závislosti na konfiguraci, více dále.

5. V případě úspěšného ověření je provedena autorizace, na základě jejíhož výsledku jsou uživateli vrácena data, nebo HTTP kód 401 Unauthorized.

2.2.2.1 Způsob získání přístupového tokenu

OAuth 2.0 standard definuje různé způsoby přihlášení – [43]

- **Autorizační kód (Authorization code grant)**

  Tento způsob přesměruje uživatele na přihlašovací stránku poskytovanou autorizačním serverem. Spočívá v tom, že po přihlášení uživatelem získá klientská aplikace autorizační kód. Tento kód následně společně s tajným klíčem odešle pro získání access tokenu. Je to standardní způsob získání access tokenu.

- **Implicitní (Implicit grant)**

  Tento způsob je zjednodušením způsobu s autorizačním kódem. Je vynechán krok, kde autorizační server vrátí autorizační kód, a klient tak rovnou obdrží přístupový token. Tento krok je vynechán, aby klientská aplikace nemusela znát tajný klíč. Je nicméně náchylný na odposlechnutí tokenu útočníkem.

- **Klíč pro výměnů kódů (Proof key for code exchange grant)**

  Tento způsob je pokusem o lepší zabezpečení Implicitního způsobu. Je velmi podobný způsobu Autorizační kód, ale tajný klíč je generován dynamicky klientem. Tím se stává bezpečnějším než Implicitní grant, protože znemožňuje útočníkovi odposlechnout autorizační kód.

- **Kód zařízení (Device code grant)**

  Tento způsob se používá pro věci jako IoT, které mají omezené možnosti vstupu.

- **Autorizace klientu (Client credentials grant)**

  Tento způsob nevyžaduje zapojení uživatele. Slouží pak k autorizaci klientské aplikace, která pro tento účel použije na pozadí identifikátor klienta a tajný kód.

- **Autorizace uživatele (Resource owner password credentials grant)**

  Tento způsob spočívá v tom, že uživatel zadá přihlašovací údaje přímo do klientské aplikace, která pak tyto údaje použije k získání přístupového tokenu. Tento způsob je relativně málo bezpečný a není doporučován.

- **Obnovovací token (Refresh token grant)**

  Pro úplnost uvedu i tento způsob. Tento způsob se často používá k doplnění předchozích způsobů. Není vždy vhodné vyžadovat po uživateli, aby se přihlašoval neustále dokola. Platnost přihlašovacího tokenu bývá v řádu minut, nutnost přihlášení každých pár minut by značně snížila uživatelský komfort. Tento způsob tedy spočívá v tom, že pro získání přístupového tokenu je použit tzv. refresh token, bez nutnosti opětovného přihlášení. Ten se typicky vydává společně s access tokenem.

Projekt nové administrace bude používat standardní Autorizační kód (Access code grant).

2.2.2.2 Různé tokeny

Jak jsem zmínil, autorizační server bude vracet access a id token a volitelně také refresh token. Token jako takový je vlastně jen text, který je vygenerován serverem a používám klientem. Typicky se používá právě k zajištění autorizace a autentizace a nejpoužívanějším formátem je JWT.

„JSON Web Token (JWT) je otevřený standard (RFC 7519), který definuje kompaktní a samopopisný způsob bezpečného přenosu informací mezi jednotlivými stranami ve formě JSON objektu. Tato informace může být ověřena a je důvěryhodná, protože je digitálně podepsaná.“[44]

Rozdíly mezi nimi jsou následující –[45]

- Access token

  Tento token je základním mechanismem pro umožnění fungování OAuth2 protokolu. Jsou posílány klientem na server se zdroji, kde se tento token použije k přístupu k chráněným zdrojům.

- ID token

  Tento token není zaveden v OAuth2 standardu, ale až v OpenID standardu. ID token obsahuje informace o daných uživatelích, jako je například jméno, email, telefon a podobně. Nejsou použity k samotnému přístupu k chráněným zdrojům, ale používají se v případě, kdy klientská aplikace tyto informace potřebuje k různým úkonům.

- Refresh token

  Tento token je opět zahrnut již v OAuth2 standardu. Používá se pro zvýšení komfortu uživatelů pro snažší opětovné získání přístupového tokenu po jeho expiraci. Je volitelný.

Autorizační server v případě tohoto projektu bude vracet všechny 3 tokeny.

2.2.2.3 Ověření access tokenu

V základní OAuth2 specifikaci není výslovně předepsáno, jakým způsobem se má access token validovat.

Základní metodou je ověření, že daný token nebyl změněn, a že byl opravdu vydán konkrétním autorizačním serverem. Díky tomu, že je JWT digitálně podepsán, stačí tento podpis zkontrolovat pomocí veřejného klíče daného autorizačního serveru. Tento způsob má výhodu v tom, že je rychlejší a snažší na provedení, nedokáže ovšem rozpoznat revoke, tzn. odhlášení uživatelem.[46]

Revoke lze rozpoznat pomocí tzv. introspekce, což je specifikace RFC 7662 rozšiřující OAuth2 standard. Pro ověření daného tokenu je pak použit specifikací daný endpoint autorizačního serveru, který vrátí aktuální informace o tokenu. Díky tomu může poznat i zmiňovaný revoke. Je nicméně náročnější na zdroje a pomalejší.[47]

Pro validaci tokenů v našem případě bude použit základní způsob bez introspekce.

2.3 Backendová část

Backendová část je vlastně první částí, kterou začneme v týmu implementovat. Jak jsme zjistili v první kapitole, bude se jednat o novou implementaci za použití jazyka C#. Pro název této komponenty byl zvolen Pangolin. Nicméně, to že se jedná o novou aplikaci neznamená, že návrh provedený v rámci projektu Porcupine je nevhodný. Při návrhu datového modelu tedy budu vycházet právě z návrhu pro projekt Porcupine. Konkrétně se lze s tímto návrhem seznámit v mojí bakalářské práci[1], případně v práci Ing. Tomáše Hojka[10].

Začněme nicméně od píky a definujme, které požadavky bude tato část řešit. Protože se bude jednat vlastně o velkou většinu požadavků, nevidím důvod je všechny zopakovat a zmíním tak jen ty, ke kterým je vhodný rozšiřující komentář. –

- **NR3 – Zpětná kompatibilita**

  Tento požadavek říká, že systém nesmí pracovat s existující databází tak, aby rozbil funkčnost administrace a eshopu OpenCart.

- **NR5 — Rozšiřitelnost**

  Tento požadavek víceméně zasahuje do celého projektu, z hlediska backendové komponenty nicméně klade důraz na kvalitu kódu a návrhu, kde především na kvalitu kódu doplatil předchozí projekt Porcupine.

- **FR12 — Primární doména**

  Tento požadavek rozšiřuje funkčnost starého systému a je pro něj tedy nutné provést úpravy v databázi. Tyto úpravy musí splňovat **NR3**

- **FR17 — Testování**

  Z hlediska backendové části je nutné výsledek otestovat automatizovaně. Na rozdíl od frontendové části zde totiž nelze testovat věci jako uživatelské rozhraní.

- **NR4 – Dokumentace**

  Zaznamenat dokoumentaci je nutné především právě v backendové části. Bez kvalitní dokumentace nelze rozumně napojit jakýkoliv frontend.

2.3.1 Architektura backendu

V rámci architektury nebyly provedeny žádné změny od projektu Porcupine. Stále se jedná o RESTové rozhraní a bude implementovat model MVC. Backend bude konkrétně realizovat vrstvy M a C, neboli Model a Controller.

- **Model**

  „Model obsahuje logiku a vše, co do ní spadá. Mohou to být výpočty, databázové dotazy, validace a podobně. Model vůbec neví o výstupu. Jeho funkce spočívá v přijetí parametrů zvenku a vydání dat ven.“[48]

  V rámci projektu Pangolin se bude jednat o relativně nejsložitější část, protože musí zajistit překlad dat ze struktury databáze do objektů, se kterými chceme reálně pracovat, a zpět.

- **Controller**

  „Controller je nyní onen chybějící prvek, který osvětlí funkčnost celého vzoru. Jedná se o jakéhosi prostředníka, se kterým komunikuje uživatel, model i view. Drží tedy celý systém pohromadě a komponenty propojuje.“[48]

2.3.2 Produktový katalog

Nyní se již můžeme pustit do návrhu samotného rozhraní backendu. Jak jsem zmiňoval výše, návrh vychází z návrhu pro projekt Porcupine, nicméně byly zde provedeny změny. Pro jejich pochopení bych se rád odkázal ještě jednou do první kapitoly, do sekce 1.1.1, kde byly analyzovány některé problémové části systému. Produktového katalogu se týkají především první 4 podsekce. Datový model části databáze zabývající se produkty lze pozorovat na obrázku 2.2. Protože vycházím ze svého návrhu provedeného v rámci mojí bakalářské práce[1], nepovažuji za vhodné vypisovat přímo do textu práce celý výsledek, který bude ve velké míře shodný s původním návrhem. Budu tedy v rámci textu prezentovat pouze relevantní změny od původního návrhu. Detailní specifikace lze pak nalézt na přiloženém médiu.

2.3.2.1 Problém s multidoménovostí

Pojďme se tedy podívat na změny. V tabulce 2.1 lze pozorovat relevantní část návrhu pro detail produktu, který byl proveden v rámci projektu Porcupine.

Tabulka 2.1 ProductDetail – Porcupine

název	typ	popis
id	integer	Identifikátor produktu
defaultDomain	DomainProduct	Atributy specifické pro danou doménu
otherDomainNames	string	Všechny domény produktu kromě výchozí
... další atributy nejsou relevantní pro tuto část		

Většina návrhu je provedena v souladu s požadavky na novou administraci, a není tedy záhodno ji více měnit. Chtěl bych nicméně upozornit na návrhový vzor, který byl v projektu Porcupine použit v souvislosti s multidoménovostí platformy administrace a eshopu. Detailní specifikaci lze najít v

mojí bakalářské práci[1]. V tabulce 2.1 lze pozorovat, že detail produktu vrací data, která jsou společná pro všechny domény, dále data pro výchozí doménu a pro ostatní domény pouze její název. Pro získání všech dat pro daný produkt je pak nutné zavolat další endpointy, tentokráte se specifikovanou doménou, které pro tuto kombinaci produktu a domény vrátí výsledný objekt „DomainProduct“, který obsahuje data specifická pro konkrétní doménu. Tento přístup je pak zrcadlen i ve zbytku rozhraní, kde se data pro produkt ukládají a upravují separátně pro jednotlivé domény a také se separátně ukládají data společná pro jednotlivé domény.

Tento přístup pak vede na některé problémy –

- **Rychlost zpracování dat a složitost kvůli ztrátě kontextu**

  Jak jsem zmínil v podsekci 1.1.1.3, data pro jednotlivé domény se získávají na základě jazyka dané domény. Takto získaná data lze dále přepsat pomocí doménového overridu. Znamená to tedy, že když mají být data pro nějakou doménu upravena, je potřeba upravit buď data přímo v entitě pro jazykovou mutaci, nebo použít doménový override. Upravit jazykovou mutaci je však v kontextu doménově specifických úprav nebezpečné, protože tato data jsou základem i pro potenciální druhou doménu se stejným jazykem. Zároveň, upravovat vždy jen doménový přepisovač ve výsledku vede na to, že budou takto přepsána všechna data. Tím, že se v rámci zpracování takového požadavku potenciálně upravují i data spojená s jinou doménou, není možné tyto požadavky provádět paralelně. Konzument takového rozhraní pak musí požadavky posílat sériově, čímž se prodlouží čas ztracený přenosem dat, a celkový čas zpracování bude delší i na straně serveru, protože se data pro některou doménu potenciálně upraví několikrát.

  Jediný způsob, jak tuto situaci napravit je nějakým způsobem dodat každému požadavku kontext o všech úpravách na všech doménách.

- **Ukládání dat při chybě**

  Dalším problémem souvisejícím s rozdělením požadavků na jednotlivé domény je způsob ukládání. Server nemůže držet kontext mezi jednotlivými požadavky na jednotlivé domény. Uvažme situaci, kdy klient pošle sériově požadavky na úpravu dat pro dvě domény – první požadavek se správnými daty a druhý se špatnými. V momentě, kdy druhý požadavek selže je již první požadavek vyřízený a nelze jej snadno vrátit zpět. Uložila se tedy jen polovina dat, což je problém v rámci klienta správně komunikovat administrátorovi.

  Tento problém vyřešit je velmi složité, znamenalo by to nutnost držet si na pozadí v klientu původní data, a v případě selhání je poslat jako další požadavek na server. Postup, který by byl ještě více problematický v případě vytváření nového produktu.

- **Složitost rozhraní**

Menším, nicméně stále důležitým problémem, je pak složitost použití takového rozhraní. Vzhledem k tomu, že požadavkem na novou administraci je zavedení výchozí domény, je pak nutné, aby klient posílal nejprve data pro výchozí doménu, a až následně data pro zbytek domén. Zároveň, aby nedocházelo k tzv. race conditions[1] mezi požadavky pro jednotlivé domény, backend server by musel implementovat zámky na úrovni produktu, nebo by klient musel posílat data sériově všechna.

Tento problém je tedy nejlépe řešitelný ze všech identifikovaných problémů, ale stále vyžaduje zbytečnou práci navíc.

Změnou pro systém Pangolin je tak změna tohoto návrhového vzoru. Data se již budou posílat najednou pro celý produkt, tedy pro všechny domény najednou. Backend server tím získá potřebný kontext, bude možné definovat transakční zpracování na úrovni úpravy celého produktu a rozhraní bude mnohem snažší pro použití pro jeho konzumenta. Jsou tím řešeny všechny 3 identifikované problémy. Pro představu čtenáře jsou pak relevantní změny identifikovány v tabulce 2.2.

Tabulka 2.2 ProductDetail – Pangolin

název	typ	popis
id	integer	Identifikátor produktu
domains	DomainProduct array	Atributy specifické pro všechny domény
... zbytek návrhu se liší pouze v detailech		

2.3.2.2 Cenové akce a produktové volby

Další částí rozhraní úzce spojenou s produktem jsou cenové akce. Pro jejich návrh byl v projektu Porcupine použit návrhový vzor, který v rozhraní definuje pouze operace GET a PUT. První zmiňovaná operace má za cíl získat všechny potřebné entity, zatímco operace PUT zajišťuje úpravu, vytváření i mazání v jednom. Tento způsob zpracování byl zvolen proto, že klienta nutně nezajímá, zda se daná entita upravila nebo vytvořila nová. Takovýto endpoint eliminuje logiku hledání nových, upravených a smazaných entit ve frontendové části aplikace a přenese ji na backend, který k tomu je uzpůsoben lépe.

Poslední navrhovanou částí jsou produktové volby. Ty realizují například velikosti bot, nebo barevné varianty a existují v rámci jednoho produktu. Dané rozhraní se od projektu Porcupine opět neliší a na rozdíl od rozhraní pro cenové akce je ve standardním formátu, nebudu ho zde tedy dále rozvíjet.

[1] „Závodní podmínky“ – termín označující situaci, kdy výsledek několika paralelních operací je předem nedefinovaný a je daný pořadím zpracování

2.3.3 Správce souborů

Jedním z požadavků na nový backend byla i možnost spravovat souborový systém na serveru a přiřazovat obrázky a dokumenty entitám, konkrétně se jednalo o požadavek **FR14 – Správa souborového systému**. Způsob, kterým toto řeší systém OpenCart jsem popsal v podsekci 1.1.1.5. Správu souborového systému řešil i projekt Porcupine, pojďme se tedy podívat na způsob, kterým to řeší.

Jak píše ve své diplomové práci[10] Ing. Tomáš Hojek, Porcupine pro správu souborového systému vytvořil v databázi novou tabulku. V té se strom souborového systému ukládal v následujícím formátu –

- **id (int)** uměle vytvořený identifikátor složky nebo souboru
- **parent_id (int)** identifikátor složky, do které složka/soubor patří
- **name (varchar)** název složky/souboru
- **directory (tinyint)** příznak určující, zda se jedná o složku

Idea za vytvořením takové tabulky je vlastně vcelku jednoduchá – eliminace denormalizace dat, kterou jsem zmínil právě v podsekci 1.1.1.5. V momentě, kdy entity obsahují už jen cizí klíč této tabulky místo celé cesty, se musí data upravovat už jen na jednom místě. To je ideální stav, kterého chceme dosáhnout.

Nicméně, i když je tento návrh validní, považuji ho za krok správným směrem ve špatný čas. Jedním z stěžejních požadavků je totiž paralelní souběh se systémem OpenCart, konkrétně **NR3 – Zpětná kompatibilita**. Dokud souborový systém upravuje pouze nový systém, není to problém, protože s tím, že musí držet souborový systém a databáze ve stejném stavu se počítá. Nicméně, stará administrace může tento stav rozbít a nový systém s tímto musí počítat. Tímto se velmi zvyšuje komplexita celé operace. Tudíž celý návrh této komponenty bude potřeba přepracovat tak, aby jsme se tomuto problému vyhnuli, a zároveň umožnili vývoj tímto směrem později. Pojďme se tedy podívat na možnosti tohoto návrhu.

2.3.3.1 MinIO

Od zadavatele vzešel v návrhové fázi návrh použít objektové úložiště MinIO pro reprezentaci souborového systému. Firma Jagu MinIO zvažuje použít v některých dalších projektech a toto řešení by tak mohlo být vhodné i zde.

„MinIO je objektové úložiště, poskytující rozhraní kompatibilní s Amazon Web Services S3 a podporující všechny hlavní funkce S3. MinIO je navržen tak, aby bylo možné jej nasadit kdekoli - ve veřejném nebo soukromém cloudu, na infrastruktuře typu baremetal, v orchestrizovaných prostředích a na edge infrastruktuře.“[49]

Nejprve je však potřeba se podívat, co to vlastně je objektové úložiště a jaký je jeho rozdíl oproti do této chvíle používanému klasickému souborovému úložišti.

„Objektové úložiště je architektura počítačového úložiště, která se uzpůsobena ke zpracování velkého množství nestrukturovaných dat. Na rozdíl od jiných architektur označuje data jako rozdílné jednotky, spojené s metadaty a unikátním identifikátorem, který se používá k nalezení a přístupu k jednotlivým datovým jednotkám.“[50]

Při srovnání jejich vlastností dostáváme vcelku jednoduchý obrázek – objektové úložiště se hodí na ukládání velkého množství nestrukturovaných dat, snadno se škáluje a je efektivní na vyhledávání v něm na základě metadat. Oproti tomu klasický souborový systém má pro jednotlivé soubory se známým umístěním rychlejší přístup.[51]

Pro současnou situaci administrace se bavíme o vyšších desítkách až nižších stovkách tisíc souborů, které je potřeba spravovat. To se již blíží limitu, kde souborové úložiště začíná být neefektivní a mohl by tedy být dobrý nápad toto řešení použít. Nicméně, tento pohled nereflektuje požadavek na paralelní souběh obou systémů. V tomto řešení se skrývá vlastně úplně stejný problém, který jsem identifikoval v řešení pro systém Porcupine. OpenCart bude stále používat souborový systém a bylo by tedy nutné ho synchronizovat s objektovým úložištěm. Potenciál pro toto řešení tak vidím až v momentě, kdy nový systém reálně nahradí OpenCart.

2.3.3.2 Přímé použití souborového systému

Výsledný návrh se tedy pokusí vyhnout nutnosti synchronizace souborového systému mezi oběma systémy tím, že prostě přistoupí k souborovému úložišti přímo. Nový systém pak bude soubory poskytovat přímo na adrese definované jejich cestou v souborovém systému. Tím se zajistí, že administrátor uvidí vždy aktuální souborový systém bez nutnosti synchronizovat ho. Navíc, nahradit tento způsob za řešení pomocí objektového úložiště pak bude vcelku snadné – bude stačit toto objektové úložiště inicializovat pomocí souborového systému, a k jednotlivým souborům se stále bude přistupovat podle názvu. Přístup však bude veden přes klienta objektového úložiště, ne přímo přes souborový systém.

Konkrétní dokumentaci k vytvořenému rozhraní lze pozorovat na přiloženém médiu, přidávat jej sem přímo do textu nepovažuji, podobně jako u produktového katalogu, za stěžejní. Toto rozhraní je ve výsledku velmi podobné jako pro systém Porcupine s tím rozdílem, že jednotlivé zdroje (soubory a složky) nejsou definovány pomocí celočíselných identifikátorů, ale přímo podle jejich cesty v souborovém systému.

2.4 Frontendová část

Na rozdíl od backendové části budeme pracovat nad již existujícím řešením. Změny v rámci návrhu této části jsou tak vcelku limitované, nicméně to nutně neznamená že vývoj frontendové části bude o mnoho snažší. Je totiž nutné upravit velkou část kódu na pozadí tak, aby pracovala s novou backendovou komponentou, je potřeba dotáhnout aktualizaci na verzi Vue 3 a nakonec je potřeba přidat některé části, které nebyly zpracovány v rámci předchozích prací.

Nejprve opět zmíním požadavky, které tato komponenta řeší. Podobně jako u backendové části ale seznam omezím na ty, ke kterým je vhodný doplňující komentář, protože opět nastává sitauce, kde se frontend týká téměř všech požadavků.

- **FR4 — Úprava viditelnosti v rámci přehledové tabulky**

  Tento požadavek se týká čistě frontendu a říká, že tabulka produktů by měla mít možnost být konfigurována uživatelem.

- **FR11 — Jednotné tlačítko pro uložení**

  Tento požadavek říká, že všechna relevantní data k produktu by se měla uložit pomocí pouhého jednoho tlačítka. To znamená nutnost uživatele správně informovat o chybě, která může nastat na pouze některých částech dat.

- **FR13 – Zadávání cen v měně definované pro danou doménu**

  Tento požadavek znamená, že administrátor má mít možnost zadávat cenu v měně specifické pro danou doménu, ačkoliv na pozadí se pracuje pouze s českými korunami.

- **FR15 — Validita dat**

  Tento požadavek říká, že frontend by měl provádět všechny potřebné validace dat už sám, aby mohl uživatele adekvátně upozornit na chybu už před zbytečným pokusem o uložení.

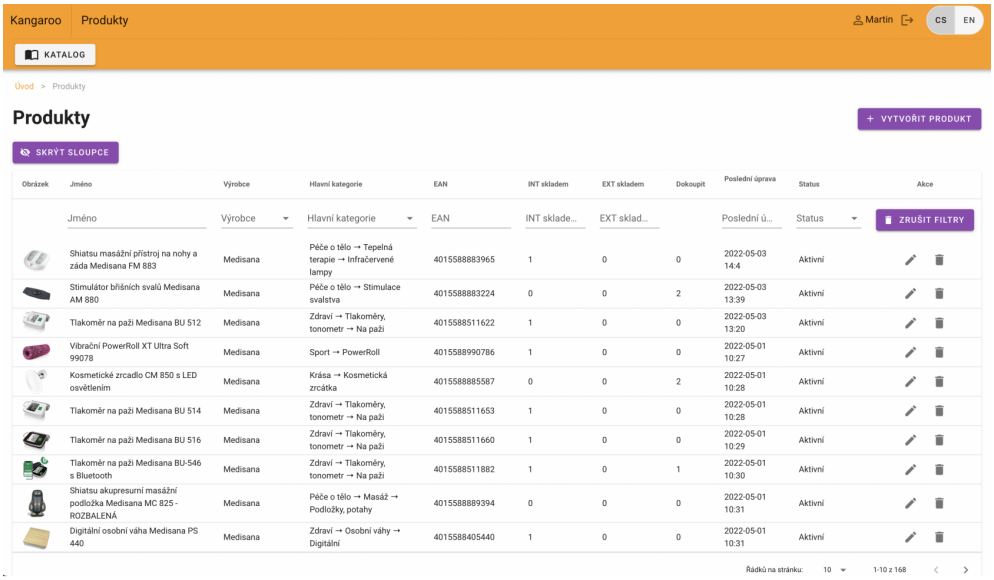
Před tím, než se pustím do samotného návrhu jednotlivých komponent aplikace, musím opět upozornit na to, že navazuji na předchozí práce. V kontextu produktů se jedná především o Bc. Jana Babáka[12], ale zpracovat je potřeba i komponentu správce souborového systému navrženou Bc. Martinem Dvořákem[11]. Některé karty byly také navrženy v rámci předmětů BI-SP. Návrh tedy pojmu podobně jako u produktů, a to úpravami již existujících wireframů. Všechny návrhy provedené v rámci projektu Kangaroo jsou v nástroji Figma.

2.4.1 Přehled produktů

Pro tabulku produktů jsem zjistil, že v původním návrhu ani nevznikl wireframe. Nicméně, vzhledem k tomu že navazuji na již realizovanou komponentu, můžu se podívat na výslednou tabulku už jako implementovanou komponentu a v návrhu vycházet z ní.

Pro představu čtenáře je tato komponenta zachycena na obrázku 2.2. Tato komponenta již prošla jedním kolem uživatelského testování, lze se tudíž podívat na výsledky tohoto testování a na jeho základě navrhnout úpravu. Z výsledků testování provedeného kolegou Babákem vyplývá, že funkčnost tabulky je dobrá a problém činí neznalým uživatelům především možnost skrývat sloupce.[12] Ve výsledku tak není nutné návrh této komponenty měnit a z vnějšku tak zůstane tabulka stejná.

Obrázek 2.2 Výsledek původní implementace – přehled produktů

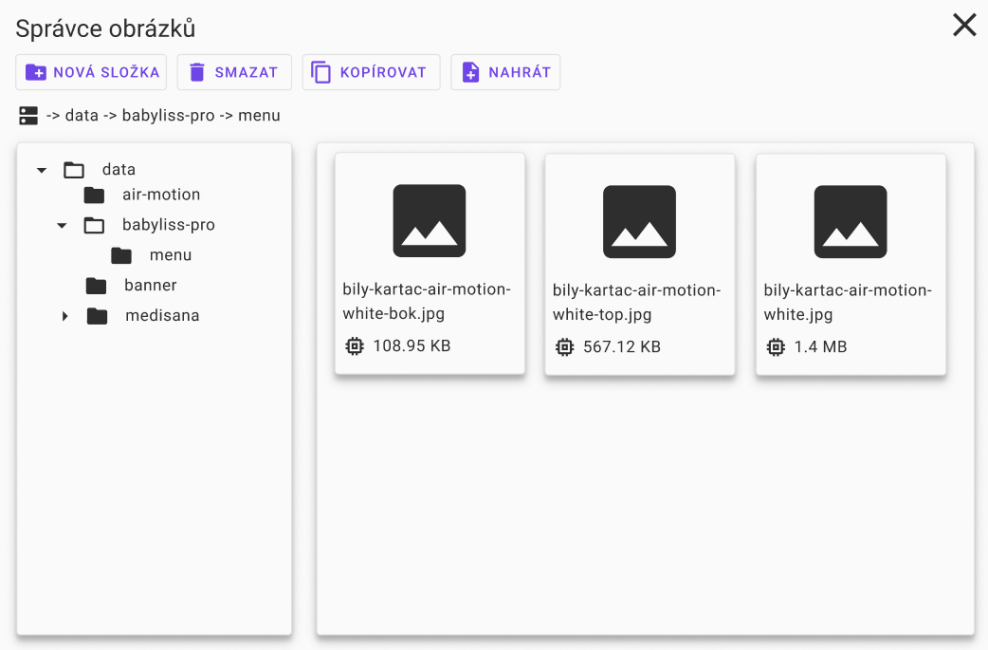


2.4.2 Správce souborů

Návrhem správce souborů se zabýval Bc. Martin Dvořák[11]. Jeho výsledný návrh lze pozorovat na obrázku 2.3. Celý správce souborů se používá jako komponenta, která zajišťuje vybrání obrázků a/nebo souborů pro různé entity. Design této komponenty připomíná správce souborů a já se ho rozhodl tomuto stavu ještě přiblížit. Celá komponenta je rozdělena na navigaci v rámci systému složek napravo a na navigaci v rámci jedné vybrané složky nalevo.

V rámci mého návrhu jsem se rozhodl tyto části sloučit do jedné, uživatel tak uvidí soubory i složky v jednom stromě souborového systému. Dále to umožňuje intuitivní přesouvání souborů a složek přetažením myší. Tato

Obrázek 2.3 Původní návrh – správce souborů



operace také uvolnila místo pro implementaci okna pro nahrání souborů přetažením do pravé části komponenty – původní návrh počítal s přetažením už na úrovni formuláře nebo použitím tlačítka pro nahrání souborů do právě vybrané složky. Také byla přidána chybějící podpora pro výběr více souborů najednou.

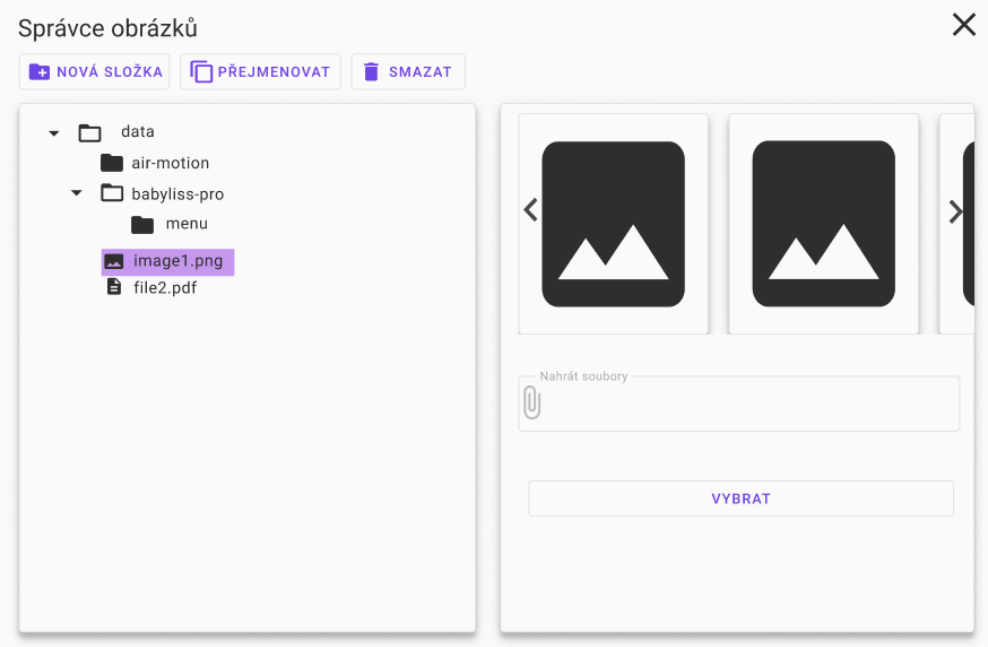
Výsledný návrh lze pozorovat na obrázku 2.4.

2.4.3 Detail produktu – hlavní formuláře

Návrhem těchto formulářů se zabýval opět Bc. Jan Babák.[12] Jeho finální návrh lze pozorovat na obrázcích v následujících podsekcích. Jako hlavní formuláře se rozumí formuláře pro data, která jsou uložena přímo v business entitě Produkt. Vzhledem k jejich množství a struktuře jsou pak navrženy 2 hlavní taby – *Obecné* a *Data*. *Obecné* obsahuje popisky a další textová data, *Data* obsahuje hlavně vazby na další entity jako kategorie produktu, související produkty, dárky k produktu, barvy a další. Každý z těchto tabů pak obsahuje další množinu tabů, kde každý reprezentuje jednu zvolenou doménu. Konkrétně se bavíme o právě jednom tabu pro výchozí doménu, a žádný nebo více tabů pro ostatní domény. Tato část aplikace také prošla uživatelským testováním, takže je možné v návrhu na tuto skutečnost reagovat.

Než se pustím dál, rád bych vysvětlil význam hlavní domény. Jak jsem zmiňoval v sekci 1.1.1 a dále v návrhu backendové části, v databázi vlastně existuje hierarchie dat. Hlavní entita produktu má nějaká data, ty se používají na

Obrázek 2.4 Výsledný návrh – správce souborů



všech doménách. Dále jsou uloženy jazykové mutace, a nakonec ještě doménově individuální overridy. Hlavní doména tedy reprezentuje data, která se v absenci doménových overridů aplikují pro ostatní domény. Tyto taby obsahují tedy i společná, nepřepisovatelná data. Pojďme se tedy podívat na návrh jednotlivých tabů.

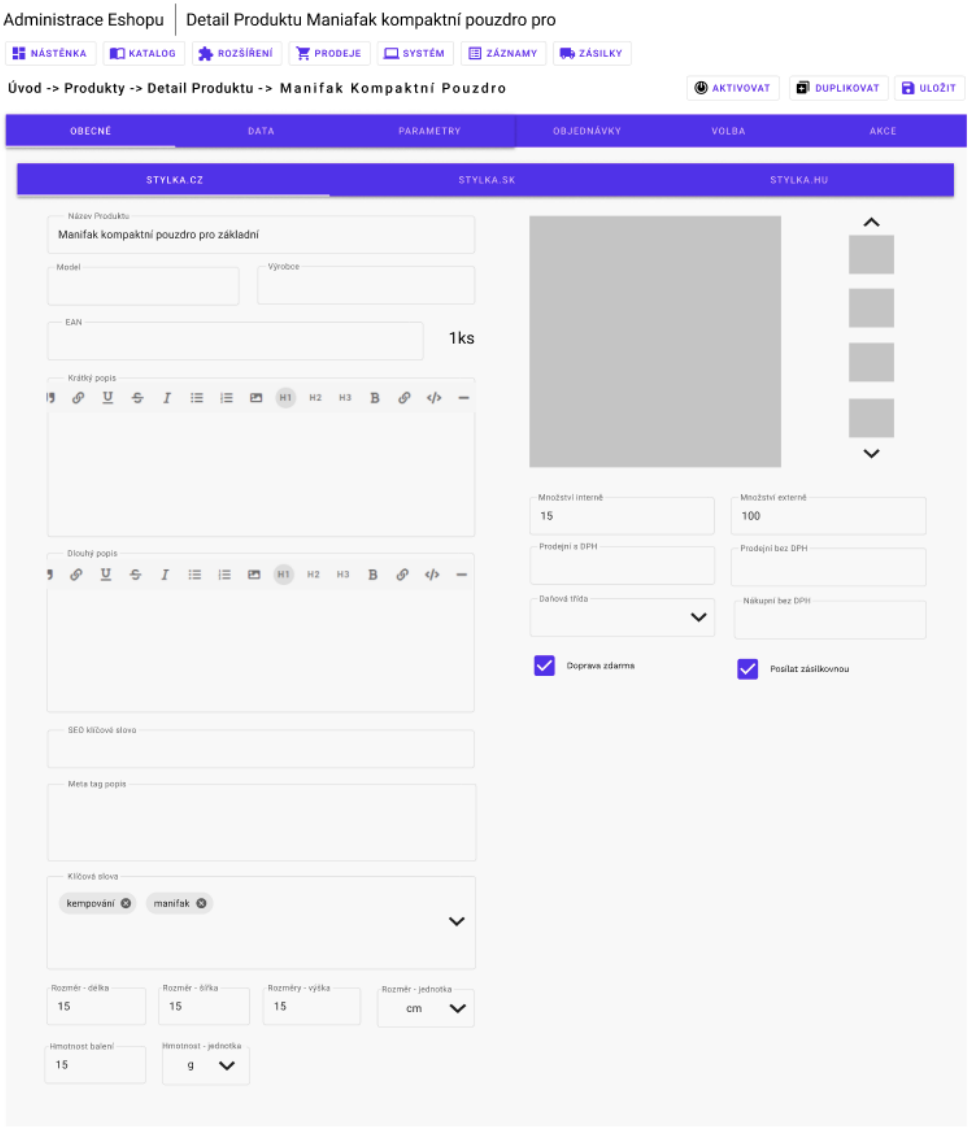
2.4.3.1 Tab Obecné

Původní návrh pro hlavní doménu lze pozorovat na obrázku 2.5, návrh pro doménu vedlejší lze pozorovat na obrázku 2.6. Začněme tabem pro hlavní doménu.

Základní design tabu splňuje všechny kladené požadavky, zajímavější bude ale podívat se na výsledky uživatelského testování a na jeho základě učinit případné úpravy. Pro taby *Obecné* bylo zjištěno, že problém může činit správa obrázků pro produkt. Uživatelé se snažili při řazení obrázků používat i velký náhled hlavního obrázku, který slouží jako vstup do správce souborů pro výběr. Nicméně, závěrem, se kterým souhlasím, je, že po informování administrátorů o tomto chování je současný způsob efektivní a není tak nutné ho měnit.

Sekundární doména opět dopadla v uživatelském testování dobře a nejsou tak provedeny úpravy uživatelského rozhraní. Výsledný návrh uživatelského rozhraní je tedy víceméně shodný s původním návrhem.

Obrázek 2.5 Původní návrh – Detail produktu – tab Obecné – primární doména



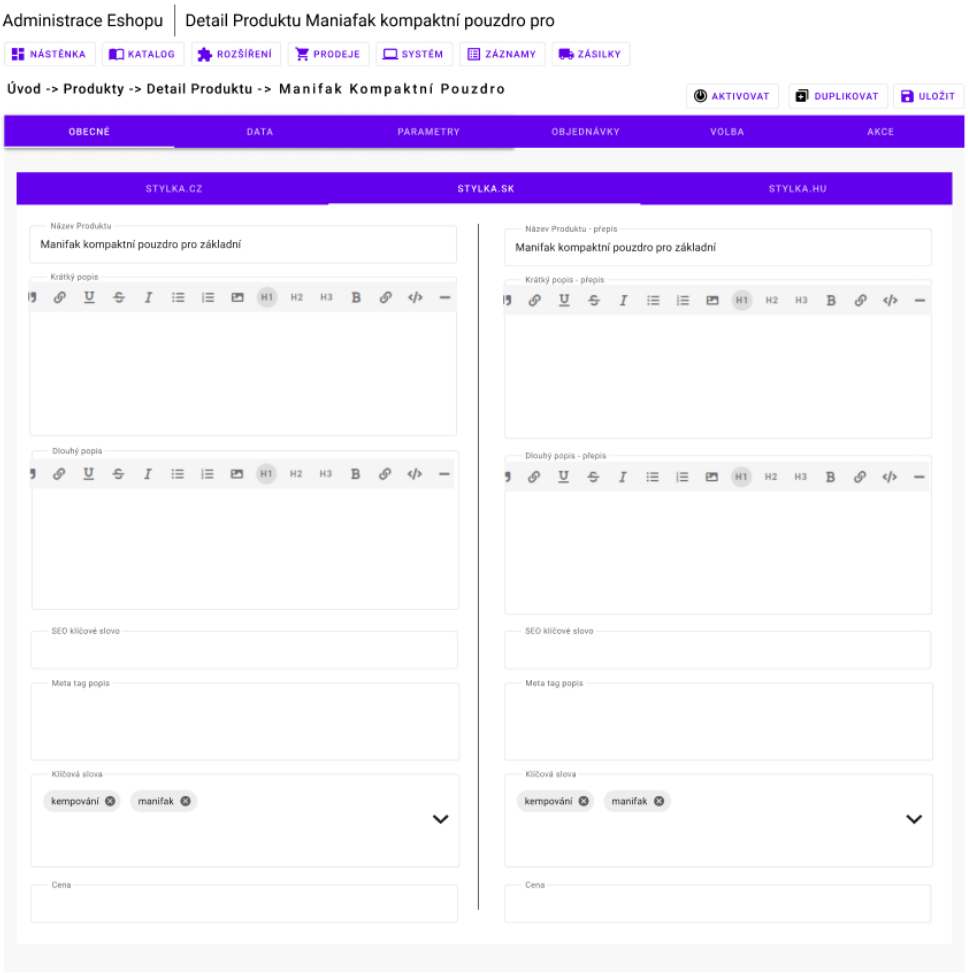
2.4.3.2 Tab Data

Původní návrh pro hlavní doménu lze pozorovat na obrázku 2.7, návrh pro doménu vedlejší lze pozorovat na obrázku 2.8.

Na rozdíl od tabu s obecnými daty je zde feedback po testování v rámci minulých prací již zajímavější. Pojďme se tedy podívat na problematické části a zjistit, zda je lze na základě připomínek upravit.

Prvním zmiňovaným problémem je selektor kategorií. V rámci uživatelského testování se ukázalo, že je vhodné přesunout jej i do tabu pro

Obrázek 2.6 Původní návrh – Detail produktu – tab Obecné – sekundární doména

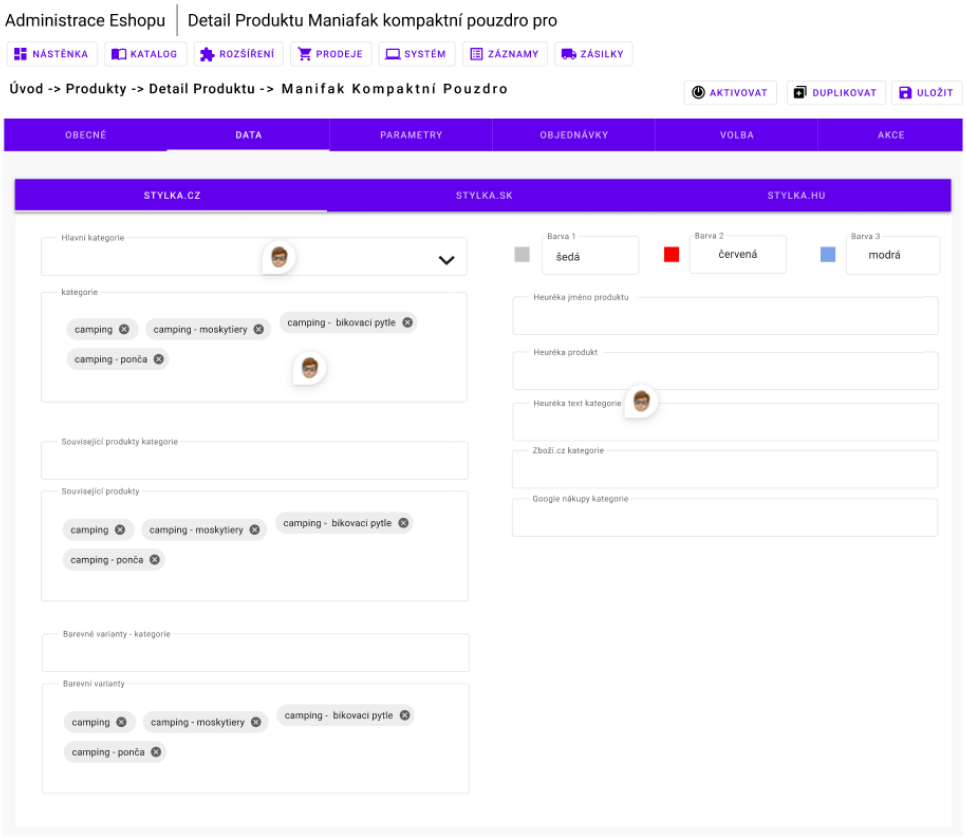


ostatní domény, a to i přesto že upravují více domén najednou. Tento krok tedy bude proveden, společně s doporučeným dodáním popisku, který o této skutečnosti informuje.

Dále je vhodné upravit selektor produktů. V původním návrhu se produkty vyhledávají pouze podle kategorie. To se ukázalo jako nevhodné, protože pamatovat si, v jaké kategorii hledaný produkt existuje není intuitivní a snadné. Rozhodl jsem se tedy implementovat vyhledávání podle názvu produktu. V případě, že v textovém poli není zadán žádný řetězec pro vyhledávání, použije se na pozadí vhodná výchozí hodnota – v případě souvisejících produktů právě jeho kategorie, v případě barevných variant produkty se stejnou barvou a v případě dárku nebyla taková hodnota nalezena, vrátí se tedy prostě prvních několik produktů bez dalšího filtru.

Výsledný návrh je lehce rozdílný navíc i v rozložení některých prvků. To ale

Obrázek 2.7 Původní návrh – Detail produktu – tab Data – primární doména



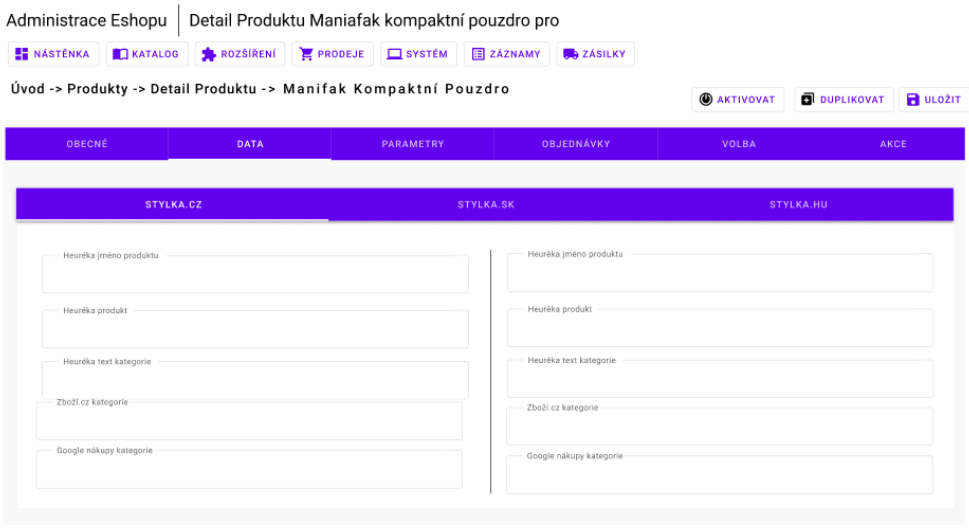
bylo provedeno již v rámci předešlé implementace. Změny, které byly relevantní pro tento návrh jsou popsány výše. Výsledný návrh lze vidět na obrázcích 2.9 a 2.10.

2.4.4 Detail produktu – tab Akce

Tento tab nebyl zatím zpracován v rámci žádné závěrečné práce a nebyl k němu proveden ani návrh v rámci závěrečné práce. To ale neznamená, že návrh neexistuje, byl totiž proveden v rámci předmětu BI-SP. Tento návrh lze pozorovat na obrázku 2.11. Na první pohled je vidět, že je mnohem jednodušší než hlavní formuláře.

Zůstává zachováno rozdělení na jednotlivé domény, protože akce je vždy definována právě pro jednu doménu. Další vyplnitelná pole jsou skupina, pro kterou tato akce platí, cena v měně dané domény a data zahájení a konce akce. Akce má definovanou také prioritu, která se nastavuje pomocí přetažení v rámci seznamu pro každou doménu. Je zde k dispozici i selektor daňové třídy, který se neváže přímo k entitě akce jako takové, nýbrž je zrcadlen z hlavního

Obrázek 2.8 Původní návrh – Detail produktu – tab Data – sekundární doména



formuláře.

Pro jednotnost výsledného návrhu jsem udělal lehké úpravy co se týče například barev, jinak návrh nebylo potřeba měnit.

2.4.5 Detail produktu – tab Volba

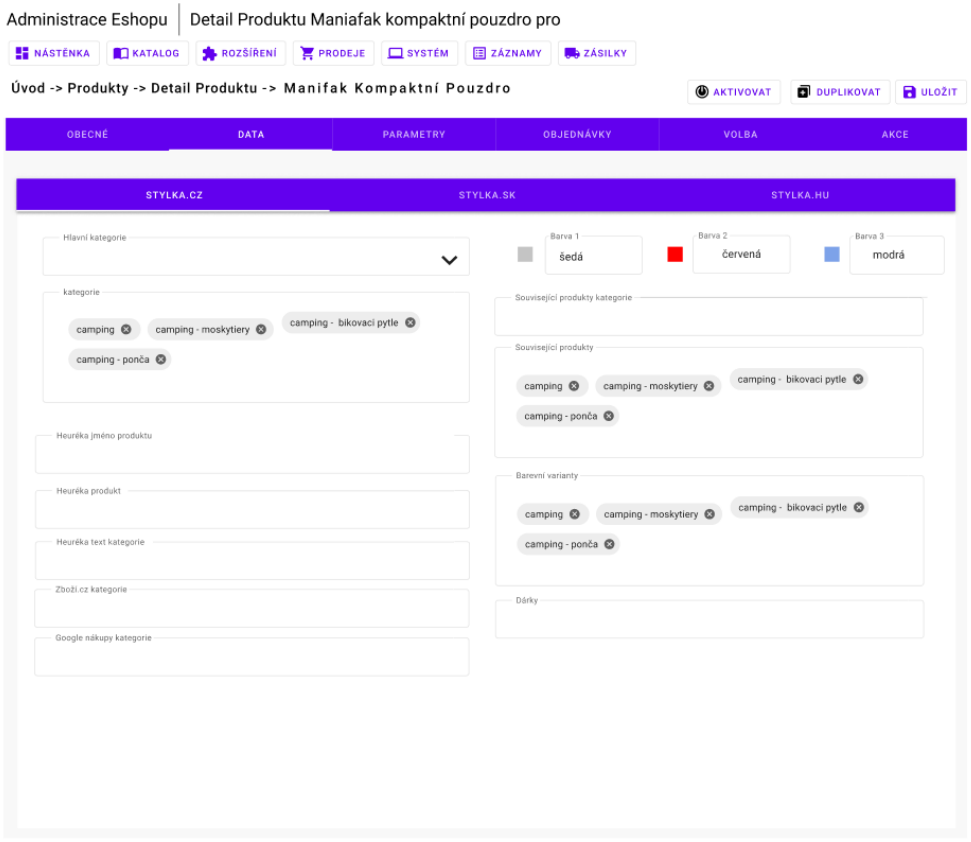
Podobně jako tab pro Akce, tab Volby zatím nebyl implementován, ale pouze navržen v rámci předmětu BI-SP. Tento návrh je k nahlédnutí na obrázku 2.12.

Entita Volby je specifická tím, že není vázána na doménu. Obsahuje ale překlady. Dále obsahuje jednotlivé zvolitelné hodnoty dané volby, kde každá opět má svůj překlad. V návrhu SP týmu shledávám v tomto ohledu nedostatky – vůbec nejsou poskytnuty některá nastavitelná data volby, jako rozdíl v ceně a právě jazykové mutace.

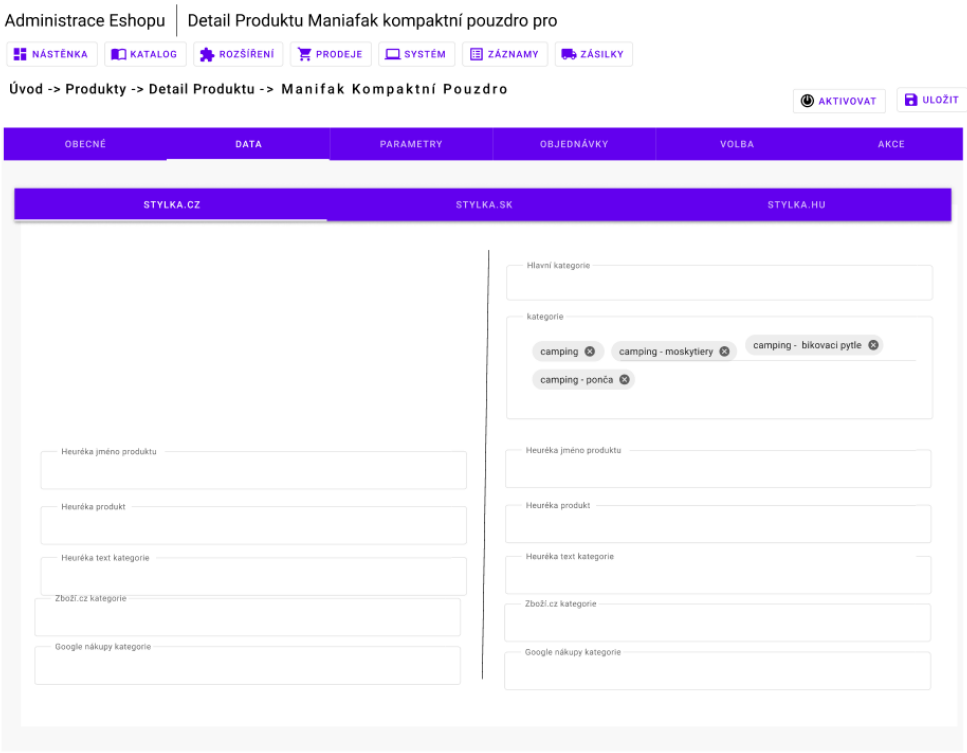
Upravený návrh je pak základní myšlenkou podobný – obsahuje jednotlivé volby, jejichž rozkliknutím uživatel dostává seznam hodnot dané volby. Opět se jedná o seznam, který se řadí pomocí přetažení. Vstupy ale obsahují navíc možnosti pro vyplnění jazykových mutací, které jsou uspořádány vertikálně. Systém na základě vybraných domén vyžaduje vyplnit jazykové mutace jazyků právě těchto domén. Rozdíl v ceně se zadává v měně výchozí domény, neboť nelze specifikovat pro různé domény.

Výsledný návrh lze pozorovat na obrázku 2.13.

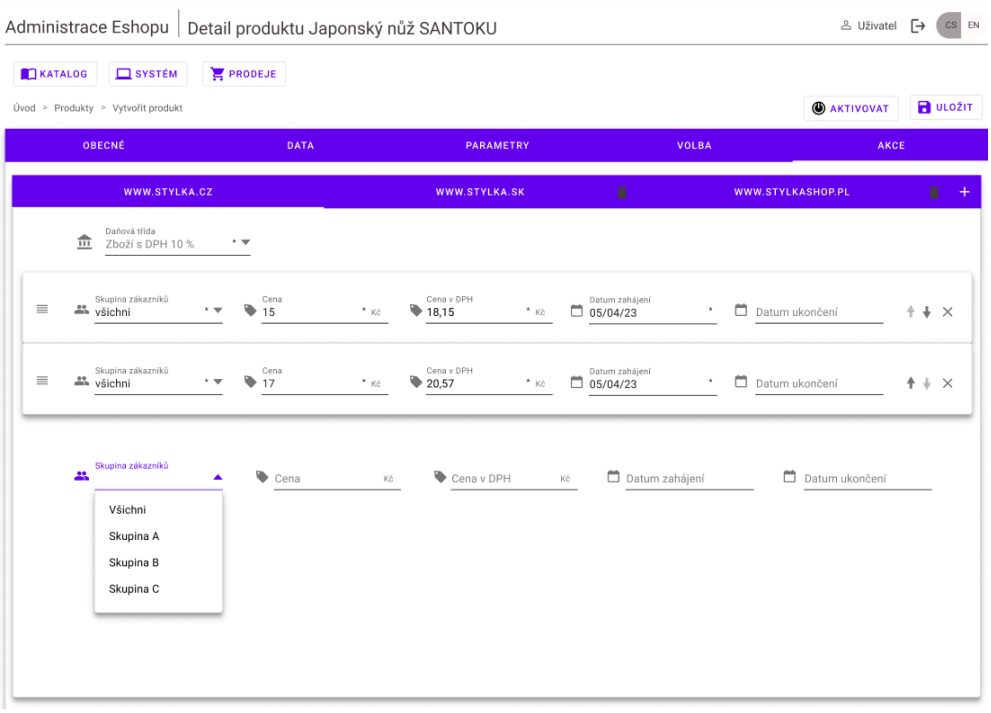
Obrázek 2.9 Výsledný návrh – Detail produktu – tab Data – primární doména



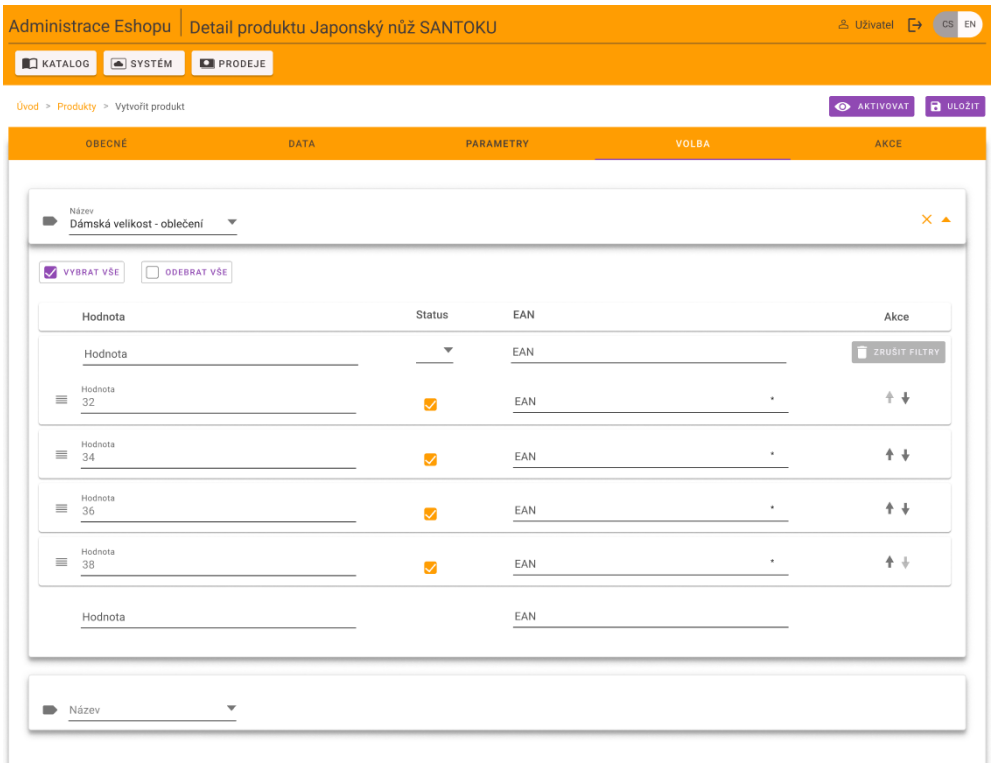
Obrázek 2.10 Výsledný návrh – Detail produktu – tab Data – sekundární doména



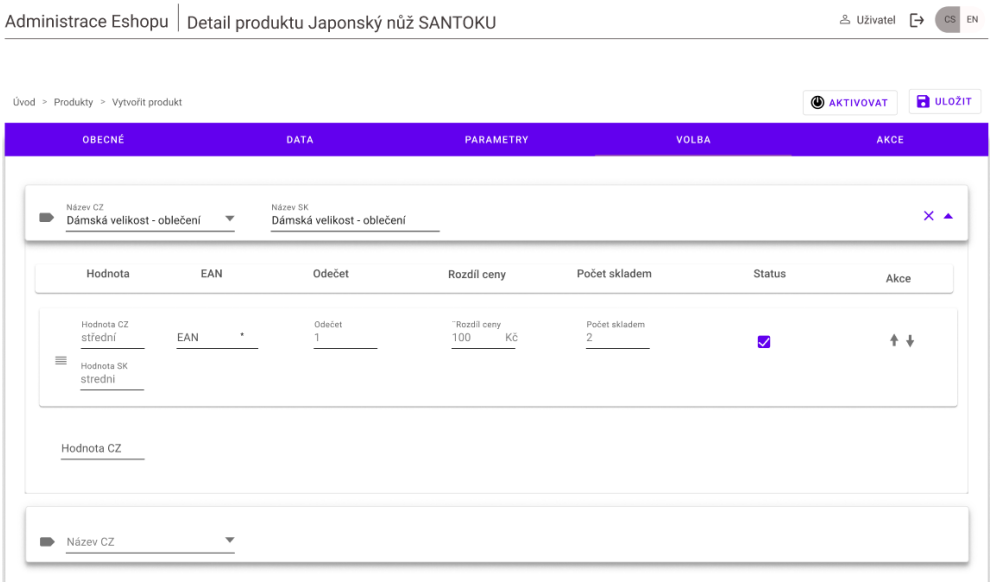
Obrázek 2.11 Původní i výsledný návrh – Detail produktu – tab Akce



Obrázek 2.12 Původní návrh – Detail produktu – tab Volba



Obrázek 2.13 Výsledný návrh – Detail produktu – tab Volba



Kapitola 3
Realizace

V rámci této kapitoly se budu zabývat implementací nové administrace podle návrhu. Kapitola bude rozdělená na backendovou a frontendovou část, což reflektuje reálnou chronologii pracovního postupu. V rámci tohoto popisu se zaměřím nejprve na použité technologie na konkrétních ukázkách kódu a neopomenu zmínit problémy a zajímavosti, na které jsem při vývoji narazil. V ideálním případě by měl čtenář po přečtení této kapitoly rozumět použití všech technologií v rámci projektu.

3.1 Implementace backendu

První kroky v rámci implementace byly realizovány jakožto implementace backendu. Jak jsem zmínil výše, pro tuto implementaci byl použit jazyk C#, konkrétně to pak znamenalo použití frameworku ASP.NET Core. V době začátku implementace byla nejnovější verzí verze 7, brzy však proběhl update na v tu dobu vydanou verzi 8. Pojďme se tedy seznámit, jak vypadá reálné použití tohoto frameworku.

3.1.1 Struktura projektu

Vzhledem k tomu, že implementujeme oddělený backend a frontend, zvolili jsme variantu frameworku ASP.NET Core MVC. Ta se specializuje právě na poskytnutí typicky RESTového rozhraní nějaké oddělené uživatelské aplikaci. Z pohledu struktury je pak jádrem soubor *Program.cs*. V rámci něj probíhá velká většina nastavení a konfigurace zabudovaných funkcí ASP.NET Core. Je pak právě tou částí, která se používá pro sestavení a spuštění aplikace.

V souladu s navrženou architekturou je kód rozdělený do 3 hlavních logických celků.

- Controller – V této vrstvě dostáváme my jako vývojáři kontrolu nad prováděním kódu od ASP.NET. Z pohledu zpracování požadavku se jedná

o fázi, kde byl požadavek přijat a data z něj převedena do vstupních DTO objektů. DTO objekty definují rozhraní vystavené klientovi.

- Service – Tyto třídy slouží k obsažení aplikační logiky a jsou volány přímo z Controllerů, ale i mezi sebou podle potřeby.
- Model – V rámci této vrstvy se nachází jak databázové entity, tak DTO objekty. Tyto třídy jsou z velké části statické a slouží především k prostému uložení dat.

Dalším důležitým konceptem je asynchronní zpracování. Tento model se používá pro lepší využití výpočetních zdrojů a je k dispozici už na úrovni jazyka C#, s využitím klíčových slov *async* pro definici takové funkce a *await* pro vyčkání na dokončení provádění asynchronní funkce. Konkrétně se jedná o *Task Based Asynchronous pattern*[52]. V překladu to znamená, že volání asynchronní funkce je izolováno a přidáno do fronty úkolů, které musí runtime .NET zpracovat. V praxi to pak znamená, že výpočetní vlákna jsou při volání dekorovaném *await* uvolněna pro frontu tasků v runtime. V případě tohoto projektu je asynchronní většina funkcí Service tříd a prakticky všechny Controllery, a to proto, že tyto funkce volají databázi.

3.1.2 Model

Prvním krokem, který bylo potřeba provést, bylo napojení databáze do nové aplikace. Reálně to pak znamená použití nějakého nástroje pro objektově-relační mapování. V případě frameworku ASP.NET Core je takovým nástrojem standardně Entity Framework, který jsme použili i v rámci tohoto projektu. Základním objektem pro funkčnost Entity Frameworku je tzv. *Database Context*. Jedná se o třídu, která vlastně realizuje všechny potřebné operace – komunikaci s databází, datové operace a obecně zmíněné ORM. Jeho inicializace probíhá právě v rámci *Program.cs*, zde je konkrétní příklad z projektu Pangolin –

Výpis kódu 3.1 Inicializace DbContextu

```
builder.Services.AddDbContext<DatabaseContext>(options =>
{
    options.UseMySql(connectionString,
        ServerVersion.AutoDetect(connectionString));
    options.UseSnakeCaseNamingConvention();
});
```

Samotný objekt Database Contextu pak obsahuje jednotlivé definované entity jakožto objekty typu *DbSet*, které se používají pro samotnou manipulaci s jednotlivými enittami.

Výpis kódu 3.2 Vnitřní struktura DbContextu

```
public class DatabaseContext(
    DbContextOptions<DatabaseContext> contextOptions)
        : DbContext(contextOptions)
{
    public DbSet<Language> Languages { get; set; }
    public DbSet<TaxClass> TaxClasses { get; set; }
    ...
```

Vzhledem k tomu, že pro náš případ se jedná o přístup *database first*, tak jednotlivé entity je pak možné definovat buď ručně, nebo nechat automaticky vygenerovat z databázového schématu. Další variantou je generování databázového schématu z definovaných entit v rámci aplikace. Vlastnosti jednotlivých sloupečků se dají specifikovat přímo v třídě reprezentující danou entitu, a to ve formě anotací. Pozorný čtenář si může všimnout, že zde není žádná anotace mapující tento atribut na konkrétní sloupeček. To není chyba, nýbrž automatická detekce sloupečku se stejným názvem v režii Entity Frameworku. Konkrétní konfigurace pro tuto databázi lze pozorovat ve výpisu kódu 3.1, konkrétně příkaz *options.UseSnakeCaseNamingConvention();*.

Výpis kódu 3.3 Entita Produkt

```
[Table("oc\_product")]
public class Product
{
    [Key]
    public int ProductId { get; set; }
    [MaxLength(64)] public string Model { get; set; }
        = string.Empty;
    [MaxLength(64)] public string Sku { get; set; }
        = string.Empty;
    ...
```

Dalším způsobem konfigurace jednotlivých entit je pomocí objektů implementujících rozhraní *IEntityTypeConfiguration*. V rámci těchto objektů se konfigurují především vazby na ostatní entity v doméně. V rámci aplikace se pak tyto prvky dají použít pro přístup k navázané entitě. Tato konfigurace se dá definovat i pro relace, které nejsou podloženy cizím klíčem na úrovni databáze. Je možné však i definovat i celou entitu přímo na základě SQL dotazu.

Výpis kódu 3.4 Konfigurace vazeb entity Produkt

```
public class ProductConfiguration
    : IEntityTypeConfiguration<Product>
{
    public void Configure(
        EntityTypeBuilder<Product> builder)
    {
        builder.HasMany<DomainOverride>(p =>
                p.DomainOverrides)
```

```
            .WithOne()
            .HasForeignKey(o => o.SrcId)
            .HasPrincipalKey(p => p.ProductId)
            .IsRequired(false)
            .OnDelete(DeleteBehavior.ClientNoAction);
        ...
```

V rámci modelu je pak třeba zmínit i DTO objekty. Ty lze dále rozdělit na vstupní a výstupní, přičemž mnohem zajímavější budou právě ty vstupní. Je pro ně totiž potřeba realizovat validaci vstupních dat. Základní validace lze realizovat pomocí anotací a není pro ni potřeba instalovat žádný speciální balíček. Tato validace se spouští automaticky při parsování vstupního JSONu do vstupního DTO objektu. Při selhání takového validačního pravidla jsou chyby vráceny ve standardním formátu. Ve frontendové části aplikace je tak možné pole chyb přečíst a přiřadit ke konkrétním vstupům.

Výpis kódu 3.5 Základní validace dat

```
public class ProductRequest
{
    [MinLength(1)]
    public ICollection<Domain>
        Domains { get; set; } = new List<Domain>();
    [Required]
    [Range(1, int.MaxValue, ErrorMessage =
        "The {0} field must be a positive integer.")]
    public int MainCategoryId { get; set; }
    [Range(1, int.MaxValue, ErrorMessage =
        "The {0} field must be a positive integer.")]
    public int? MainCategoryId2 { get; set; }
    ...
```

Taková validace však neumožňuje snadno zkontrolovat existenci daných identifikátorů v databázi. Za normálních okolností by toto nebylo potřeba řešit, neboť konzistence dat bývá zajištěna již na úrovni databáze. Nicméně, jak jsem zmínil několikrát dříve, použitá databáze neobsahuje cizí klíče, a konzistenci dat je tedy nutné hlídat již na aplikační úrovni. Pro tento účel jsem do projektu přidal balíček *FluentValidation*, který poskytuje rozhraní k definici pokročilých validačních pravidel. Tato validace se spouští po provedení základních validací a chyby vrací ve stejném formátu.

Výpis kódu 3.6 Pokročilá validace dat

```
public class ProductValidator
    : EntityValidator<ProductRequest>
{
    public ProductValidator(DatabaseContext dbContext)
        : base(dbContext)
    {
        RuleForEach(pr => pr.Domains)
```

```
            .MustAsync((r, cancellationToken) =>
                DomainExists(r.Domain, cancellationToken))
            .WithMessage("'Domain' doesn't exist");

        RuleFor(pr => pr.MainCategoryId)
            .MustAsync(CategoryExists)
            .WithMessage("CategoryId doesn't exist");
    ...
```

3.1.3 AutoMapper

Překlad mezi objekty databázového modelu a DTO objekty je nutnou operací. Pro jeho realizaci v projektu Pangolin byl použit balíček *AutoMapper*. Jeho nejzákladnější použití spočívá právě v překladu DTO objektů do databázových objektů a zpět pomocí funkce *Map*. Ve výpisu kódu 3.7 proběhne nejprve mapování z DTO objektu na databázový objekt, následně se tento databázový objekt uloží do databáze, a nakonec se výsledek přeloží zpět na DTO objekt a vrátí se klientovi.

Výpis kódu 3.7 Základní použití AutoMapperu

```
public async Task<ProductWatchdogResponse> CreateAsync(
    ProductWatchdogRequest watchdogRequest,
    CancellationToken cancellationToken = default)
{
    return Mapper.Map<ProductWatchdogResponse>(
        await CreateAsync(
            Mapper.Map<ProductWatchdog>(watchdogRequest),
            cancellationToken
        )
    );
}
```

To nicméně není vše. Mezi pokročilé funkce AutoMapperu patří provádění tohoto mapování už na úrovni databáze za použití tzv. projekce. To ve skutečnosti znamená, že přeložené SQL už vybírá pouze data, která potřebuje pro splnění definovaného mapování. Bez této funkcionality by se operace získání dat z databáze prováděla následovně –

1. Provedení SQL dotazu nad databází
2. Mapování získaných výsledků na aplikační databázové objekty v rámci ORM vrstvy
3. Mapování aplikačních databázových objektů na DTO objekt

Použitím projekce je pak vynechán druhý krok, a z databáze se vrátí pouze data potřebná pro daný objekt DTO vrstvy. Toto šetří paměť a typicky zrych-

luje běh aplikace. Takové použití AutoMapperu pak lze pozorovat ve výpisu kódu 3.6.

Výpis kódu 3.8 Použití projekce v AutoMapperu

```
public async Task<ProductWatchdogResponse> FindAsync(
    int watchdogId,
    CancellationToken cancellationToken = default
)
{
    var watchdog =
        await Mapper.ProjectTo<ProductWatchdogResponse>(
            GetQueryable().Where(
                w => w.WatchdogId == watchdogId)
            ).FirstOrDefaultAsync(cancellationToken);
    if (watchdog is null)
    {
        throw new ProductWatchdogNotFoundException();
    }

    return watchdog;
}
```

Použití této funkcionality má však také určité limity. Konkrétně při získávání potřebných dat pro detail produktu je potřeba získat data z celkem více než 23 databázových tabulek. Projekce s použitím AutoMapperu pak tato data získává v rámci 1 SQL příkazu. Tento SQL příkaz je pak velmi pomalý, jeho provedení trvá déle než 30 vteřin, protože databázový stroj má problém vykonat tolik JOINů. V tom případě je pak řešením rozdělit tento dotaz na menší části, a mapování provést pouze základním způsobem pomocí funkce Map. Tím se sice vytáhne z databáze více dat, ale celkový běh bude mnohem rychlejší.

3.1.4 Service

Service vrstva je vlastně jádrem celé aplikace. Její použití v rámci projektu Pangolin je pak ještě o to důležitější, protože rozdíl mezi rozhraním poskytovaným v rámci DTO objektů a reálnou podobou databáze je relativně velký. Service vrstva pak musí tyto rozdíly pokrýt a správně realizovat potřebné operace. Navíc musí pečlivě hlídat konzistenci dat.

Z hlediska implementace v projektu Pangolin rozšiřují service třídy *BaseService*. Její funkce spočívá v poskytnutí základních metod pro práci s databází v kontextu dané entity. Z praktického hlediska tedy realizuje operace nad objektem *DatabaseContext*, který jsem ukázal v sekci 3.1.2.

Z hlediska použití technologií je zde zajímavé například použití metod *ExecuteDeleteAsync* a *ExecuteUpdateAsync*. Tyto metody umožňují mazat nebo upravovat záznamy v databázi bez jejich hydratace na základě defino-

vaných podmínek. Tyto příkazy se tedy překládají rovnou do SQL a urychlují zpracování.

Výpis kódu 3.9 Použití ExecuteUpdateAsync a ExecuteDeleteAsync

```
public async Task RemoveOrChangeImage(
    string old,
    string? new,
    CancellationToken cancellationToken = default)
{
    var q = GetQueryable().Where(
        c => c.Image.StartsWith(imagePath));
    if (new is null)
        await q.ExecuteDeleteAsync(cancellationToken);
    else
        await q.ExecuteUpdateAsync(setters =>
                setters.SetProperty(
                    b => b.Image,
                    b => b.Image.Replace(old, new)),
            cancellationToken);
}
```

3.1.5 Stránkování

V projektu Porcupine byla většina hromadných endpointu nestránkovaná. To je nevhodné, protože bez stránkování hrozí pomalé zpracování na straně backendu a zahlcení frontendu velkým počtem dat. Pro přidání stránkování nám firma Jagu poskytla svůj interní balíček, který tuto problematiku řeší. Umožňuje také definovat pro databázové entity filtrování, což je se stránkováním úzce spojené. Z hlediska implementace stačí definovat tzv. *Query* objekt, a následně použít v Controlleru.

Výpis kódu 3.10 Příklad Query objektu

```
public class ProductQuery : FilterMetadata<Product>
{
    public ProductQuery()
    {
        MapColumn("productId");
        MapCollection("productDescriptions", mapping =>
        {
            mapping.MapColumn("name");
        });
        MapColumn("categoryId");
        MapColumn("categoryId2");
        MapColumn("sku");
    ...
```

3.1.6 Controller

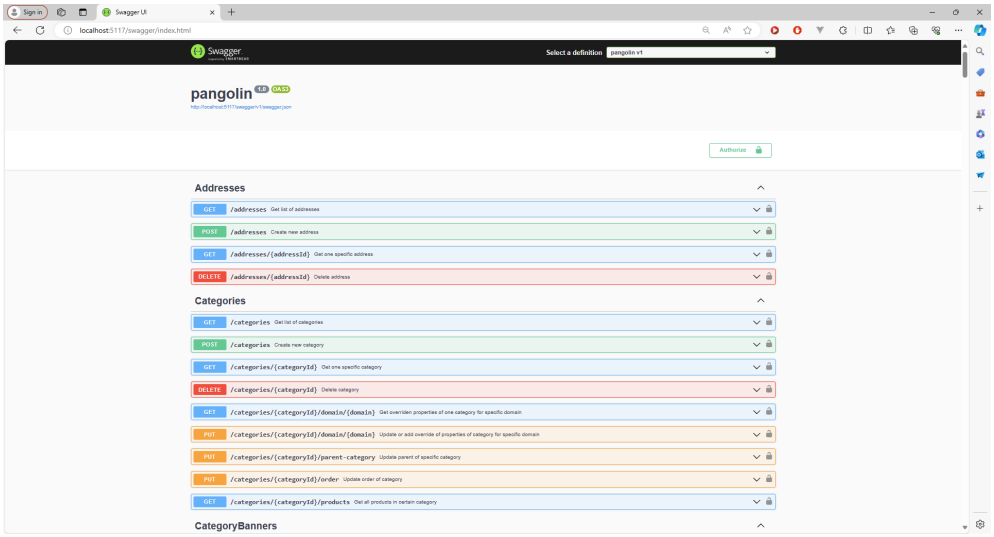
Jak jsem zmínil výše, controllery jsou třídou, ve které se začne vykonávat hlavní část uživatelského kódu. Neobsahují pak žádnou aplikační logiku, pouze předají vstupní data do příslušné service třídy a vrátí výsledek. Jednotlivé metody v rámci Controlleru pak realizují jednotlivé endpointy poskytované v rámci API.

Výpis kódu 3.11 Příklad endpointu v rámci Controlleru

```
[Authorize(Roles = Roles.TopAdmin)]
[HttpGet("minimal", Name = "GetProductsMinimal")]
[SwaggerOperation(
    Summary = "Get a list of minimal product responses",
    OperationId = "GetProductsMinimal")]
[ProducesResponseType(
    typeof(PagedResult<ProductMinimalResponse>),
    StatusCodes.Status200OK)]
[ProducesResponseType(
    typeof(BadRequestResponse),
    StatusCodes.Status400BadRequest)]
[SwaggerFilterable(typeof(ProductQuery))]
public async Task<Results<Ok<PagedResult<
        ProductMinimalResponse>>,
        BadRequest<BadRequestResponse>>>
    GetAllMinimal(
        [FromQuery] SearchParams searchParams,
        CancellationToken cancellationToken)
{
    try
    {
        return Ok(await filterableMapper.GetPaginated<
            Product,
            ProductQuery,
            ProductMinimalResponse>
                (searchParams, cancellationToken));
    }
    catch (BaseFilterableException e)
    {
        return BadRequest(new BadRequestResponse(e));
    }
}
```

Zajímavým prvkem je pak automatické generování dokumentace. V rámci projektu Pangolin byl přidán balíček *Swashbuckle.AspNetCore*, který umožňuje na základě anotací generovat dokumentaci ve formátu OpenAPI. Tato dokumentace lze buď vytvořit přímo jako .yaml soubor, nebo lze rovnou poskytovat interaktivní dashboard, ve kterém vývojář nejen vidí vygenerované rozhraní, ale může ho rovnou použít pro volání daných endpointů.

Obrázek 3.1 Swagger interaktivní dashboard



3.1.7 Autentizace a autorizace

Autentizace a autorizace uživatele byla jedním z nejdůležitějších požadavků. Podle návrhu byla zvolena integrace autorizačního serveru Keycloak. Pro realizaci tohoto požadavku stačilo použít na straně backendu balíček *AspNet.Security.OAuth.Keycloak* a nakonfigurovat jej v rámci *Program.cs*. Použití autorizace na základě konkrétní role lze pak vidět už ve výpisu kódu 3.11 z Controlleru, kde je daný endpoint přístupný pouze pro uživatele s rolí TopAdmin.

Výpis kódu 3.12 Přidání autentizace a autorizace

```
builder.Services.AddAuthentication()
    .AddJwtBearer(x =>
    {
        x.RequireHttpsMetadata = true;
        x.MetadataAddress = $"{builder.Configuration
            ["Keycloak:metadata-url"]}";
        x.TokenValidationParameters =
            new TokenValidationParameters
        {
            ValidateIssuer = true,
            RoleClaimType = ClaimTypes.Role,
            NameClaimType = $"{builder.Configuration
                ["Keycloak:name-claim"]}",
            ValidAudience = $"{builder.Configuration
                ["Keycloak:audience"]}",
        };
    });
```

```
builder.Services.AddAuthorizationBuilder()
    .SetDefaultPolicy(new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build());
```

3.1.8 Správa filesystému

Poslední větší částí, kterou jsem se zabýval, byla implementace správce souborů. Tu jsem navrhl tak, aby nebyla nutná další synchronizace se systémem OpenCart. Service třída zajišťující tuto funkcionalitu tak přistupuje přímo k souborovému systému. Při přejmenování nebo přesunutí souboru nebo složky je pak nutné na všech místech v databázi, které obsahují nějaké soubory, provést aktualizaci databázových dat. Jednu z těchto volaných metod jsem ukázal již ve výpisu kódu 3.9, kde probíhalo nahrazení v případě přesunu či přejmenování a smazání v případě smazání souboru nebo složky.

Výpis kódu 3.13 Příklad smazání souboru nebo složky

```
public async Task Delete(
    FileFolderDeleteRequest request,
    CancellationToken cancellationToken = default)
{
    using var scope = new TransactionScope(
        TransactionScopeAsyncFlowOption.Enabled);
    var relativePath = Path.GetRelativePath(
            Environment.CurrentDirectory,
            request.Path)
        .Replace("\\", "/");
    await UpdateDbPaths(
        relativePath,
        null,
        cancellationToken);
    var fullPath = Path.GetFullPath(request.Path);
    try
    {
        if (Directory.Exists(fullPath))
            FileSystem.DeleteDirectory(
                fullPath,
                DeleteDirectoryOption.DeleteAllContents);
        else
            FileSystem.DeleteFile(fullPath);
    }
    catch (System.Exception e)
    {
        throw new FileCreationException();
    }
    scope.Complete();
}
```

Nový backend pak poskytuje soubory na adrese, která kopíruje cestu k souboru v souborovém systému.

Výpis kódu 3.14 Konfigurace poskytovatele souborů

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(
            builder.Environment.ContentRootPath,
            FileSystemConstants.BaseFolder)),
    RequestPath = "/data"
});
```

3.1.9 Zpracování operací na pozadí

Při implementaci části service zajišťující vytváření a mazání produktu jsem narazil na zajímavý problém. V rámci operace vytvoření a smazání produktu je mj. třeba vytvářet nebo mazat záznamy v tabulce *oc_product_cross*. Tato tabulka realizuje M:N vazbu produktu s produktem a obsahuje vazbu každého produktu s každým. Její účel spočívá v ukládání dat o přesunech uživatele mezi jednotlivými produkty. Na základě této tabulky se pak zákazníkovi e-shopu zobrazují produkty, na které by mohl chtít pokračovat.

Problém spočíval v objemu záznamů v této tabulce ve spojení s způsobem fungování indexu v databázi MySQL. Jakožto dekompozice M:N vazby je hlavní klíč této tabulky složen z identifikátorů daných 2 produktů, které spojuje. Tato skutečnost automaticky způsobí vytvoření složeného indexu s těmito dvěma sloupečky. Při smazání produktu je pak mj. potřeba vykonat následující operace –

Výpis kódu 3.15 Mazání z tabulky oc_product_cross

```
public async Task DeleteProductCross(int productId)
{
    await DbSet.Where(pc =>
            pc.ProductId == productId
            || pc.RelatedId == productId)
        .ExecuteDeleteAsync();
}
```

To se přeloží na jednoduchý SQL příkaz.

Výpis kódu 3.16 Mazání z tabulky oc_product_cross – SQL

```
DELETE FROM oc_product_cross WHERE
    product_id = @param or related_id = @param;
```

Při testování vytvořených endpointů jsem však zjistil, že v databázi se zhruba 10000 produkty tato operace trvá na mém počítači zhruba 20 vteřin.

I kdyby na reálném serveru byla 2x tak rychlá, čekat 10 vteřin na dokončení takové operace je nepřípustné. Důvod, proč se tak děje je pak vcelku zajímavý, protože to není způsobeno pouze počtem záznamů v tabulce. Při hledání příčiny jsem zkontroloval exekuční plán výše zmíněného příkazu a zjistil jsem, že pro jeho vykonání databázový stroj nevyužívá indexu. Tato skutečnost je způsobena tím, že složený index lze číst pouze z jedné strany. Jinými slovy, dokud se maže podle *product_id*, operace trvá několik milisekund. Jakmile ale je třeba mazat i podle jiného parametru, databázový stroj je nucen použít *Full Table Scan*, což při počtu záznamů v tabulce vykonávání příkazu velmi zpomalí.

Řešením by mohlo být vytvoření druhého indexu pro sloupeček *related_id*, nicméně to by tento problém pouze přeneslo do operace vytváření produktu – sestavení indexu při vložení 20000 záznamů do tabulky o 100 milionech záznamů databázovému stroji také nějaký čas zabere.

Přirozeným způsobem, jak tento problém vyřešit pak je přidání služby, který zajistí zpracování tohoto požadavku na pozadí. Ačkoliv taková služba nebyla obsažena v návrhu, dříve nebo později by stejně byla přidána – je velmi vhodným nástrojem pro zpracování externích zdrojů kategorií, čímž se zabýval kolega Vojtěch Beneš, nebo například implementaci e-mailové služby.

Jako tuto službu jsem zvolil balíček *Hangfire*, jelikož jej již firma Jagu používá u jiných projektů a protože je jedním z nejpopulárnějších balíčků zabývajících se touto problematikou. Jeho konfigurace pak byla snadná, protože pro současné potřeby naprosto stačí použít způsob uložení jobů přímo v paměti.

Výpis kódu 3.17 Konfigurace Hangfire

```
builder.Services.AddHangfire(configuration =>
    configuration.UseInMemoryStorage()
);
```

Výpis kódu 3.18 Použití Hangfire pro vytvoření background jobu

```
...
BackgroundJob.Enqueue<IProductCrossService>(s =>
    s.GenerateProductCross(product.ProductId));
...
```

3.1.10 Souhrn

Výsledkem této práce je tak funkční část backendu administrace e-shopu zabývající se potřebami produktového katalogu. Jedná se o rozhraní obsahující zhruba 50 endpointů s pokročilou aplikační logikou. Backend je napojený na autorizační server Keycloak.

Nicméně, do budoucna bude potřeba systém dále rozšířit a případně upravit. V této práci zmíněné náměty na úpravy a rozšíření do budoucna byly

přidány do Redmine jakožto úkoly k vypracování. Prvním námětem je dokončení všech funkcionalit. Ačkoliv je v současnosti implementováno téměř všechno co se týče přímé správy produktů, některé menší části hotové nejsou. Konkrétními identifikovanými částmi, pro které byl vytvořen úkol, jsou –[1]

- Správa kuponů
- Správa „Informations“[2]
- Návrh a implementace „landing page“ administrace
- Napojení na některé externí služby, jako například skladový systém nebo dopravci

Dále bude v budoucnu potřeba přenést některé periodicky vykonávané skripty ze systému OpenCart. Pro potřeby přímé administrace nebyly přímo nutné, ale pro její kontinuální běh už ano. Tyto skripty se zabývají například aktualizací kurzu měn.

Vhodným námětem je také náhrada přímého přístupu k souborovému systému za nějaký jiný systém. V rámci této práce jsem zvážil použití systému MinIO, a zjistil jsem, že by bylo vhodné jej použít, pokud proběhne nahrazení systému OpenCart.

Dále bude do budoucna potřeba přidat do systému nějaký generátor náhledů obrázků, respektive obecně službu zajišťující vygenerování obrázků v různé velikosti. V současnosti aplikace posílá pouze velký obrázek, i když ho FE aplikace potřebuje pouze např. pro avatar.

Menším námětem je také konfigurace *Hangfire* tak, aby ukládala data do reálné databáze a případný restart aplikace tak neměl možnost přerušit vykonávání jobů na pozadí.

[1] Tento seznam ve vší pravděpodobnosti není úplný

[2] Různé popisky, jako například kontaky, obchodní podmínky, atp.

3.2 Implementace frontendu

Ve fázi analýzy jsme v týmu rozhodli, že pro implementaci frontendové části bude vhodné navázat na již existující projekt Kangaroo. Pro připomenutí, jedná se o SPA webovou aplikaci, původně napsanou ve Vue 2 za použití knihovny Vuetify 2, vytvořenou kolegy Martinem Dvořákem[11] a Janem Babákem[12]. Dále byla provedena migrace do Vue 3 a ve spojení s touto migrací bylo pracováno na kartách objednávek. O toto navázání se zasloužíl kolega Martin Salaj[23].

Vzhledem k tomu, že se jedná o navázání na již existující projekt, nevnímám jako vhodné popisovat použití konkrétních technologií a balíčků jako u backendové části, pokud tedy jeho přidání neproběhlo v rámci této implementace. Zaměřím se tedy spíš na postup, jakým implementace probíhala a to chronologicky a zmíním i problémy a zajímavosti, na které jsem při implementaci narazil.

3.2.1 Stav produktového katalogu

Relativně velká množina věcí byla před začátkem mojí implementace připravená. Byly navrženy hlavní styly, aplikace byla napojená na autorizační server a v aplikaci bylo napsáno velké množství komponent. V nejlepším stavu byly komponenty ohledně objednávek, protože na těch pracoval kolega Salaj právě při provádění migrace na Vue 3. Na komponentách týkajících se produktového katalogu, bohužel, nikdo takto nepracoval a zjistil jsem tak, že kód většiny těchto komponent byl sice zmigrovaný do správného syntaxu Composition API a aplikace jako celek se dala spustit, ale produktová sekce byla zcela nefunkční.

Po bližším ohledání jsem zjistil, že některé komponenty z knihovny Vuetify ještě v době migrace nebyly k dispozici a byly tak odstraněny z kódu. Také byly používány již neexistující sloty a rozhraní Vuetify komponent. Zjišťování, které konkrétní části rozbíjí komponenty produktového katalogu a zprovoznění alespoň minimální funkční verze, která se alespoň spustí a nespadne byla netriviální operace, která mi zabrala několik hodin.

Přes to všechno si ale myslím, že existence tohoto řešení mi spíš práci ulehčila. Ačkoliv jsem ve finále musel přepsat velkou část aplikační logiky a upravit použití Vuetify komponent, tak existence navrženého uživatelského rozhraní z velké části eliminovala potřebu starat se o layout komponent na stránce.

3.2.2 Generování typů z dokumentace

Předchozí závěrečné práce velmi často zmiňovaly problémy s napojením na BE část, jednak z důvodu pomalejšího vývoje backendu, ale také z důvodu

nekvalitní dokumentace. V projektu Pangolin jsme použili automatické generování dokumentace na základě anotací. Logickým druhým krokem ze strany FE je pak provádět inverzní operaci. Pro tento účel byl použit balíček *openapi-typescript*[53], který vygeneruje do nějakého souboru typescript typy pro konzumaci daného OpenAPI rozhraní. Tímto krokem jsme se zbavili jednoho z potenciálních problémů při integraci s backendem a usnadnili si práci při definici typů. Rád bych závěrem k této sekci pro úplnost poznamenal, že tento přidání tohoto konkrétního balíčku jsem nenavrhl já, nicméně dostal jsem se k implementaci projektu Kangaroo z týmu jako první, a přidal jsem ho tak já a zařídil jsem exportování vygenerovaných typů.

3.2.3 Správce souborů

První komponentou, kterou jsem zpracoval, byl správce souborů. V projektu Kangaroo se využívá pro realizaci výběru a správy obrázků a jiných souborů. Původní návrh jsem relativně dost změnil a z hlediska implementace se tedy jednalo o téměř celou novou komponentu. Pro novou komponentu jsem chtěl realizovat přesuny souborů a složek pomocí drag and drop, což zabudované komponenty ve Vuetify nepodporovaly. Ve skutečnosti v době začátku implementace dokonce nebyla ve Vuetify 3 přidána ani komponenta *VTreeview*, kterou používala původní implementace. Pro zobrazení stromové části správce souborů jsem tedy použil komponentu *VueTreeDnD*[54].

Tato komponenta má relativně jednoduché použití – předá se jí stromová struktura v určitém formátu a komponenta, která zajišťuje vykreslení jednotlivých uzlů stromu. Protože jsem neměl přímý přístup k této komponentě, vybírání souborů a složek nelze realizovat přímo přes eventy, použil jsem tak Vuex store pro uložení vybraných souborů a složek.

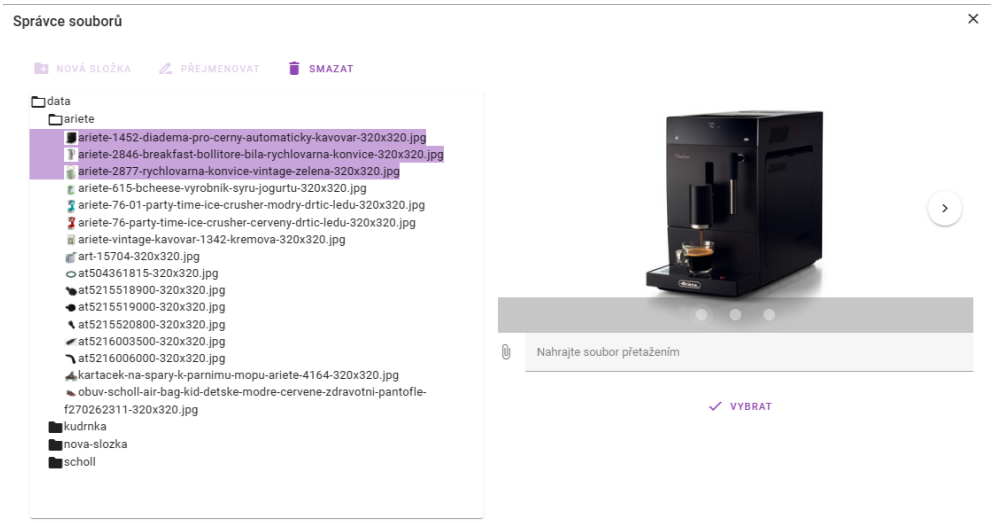
Při napojování na backend jsem objevil zajímavou vlastnost webového serveru Kestrel, který používám pro spuštění aplikace backendu lokálně. Při posílání obrázků a dokumentů jsem zjistil, že některé požadavky selhávaly se stavovým kódem 500. Nicméně, v konzoli backendu žádné informace o přijatém požadavku nebyly. Po hledání chyby na různých místech implementace jsem konečně zjistil, že se vlastně nejedná o chybu, nýbrž o chybnou konfiguraci na straně backendu. Kestrel ve výchozím nastavení přijímá data o velikosti maximálně 30MB, cokoliv většího odmítne. Z toho důvodu jsem nic neviděl v konzoli backendu. Řešení však nakonec bylo velmi snadné, pro účely daného endpointu stačilo anotací *RequestSizeLimit* tento limit zvýšit.

Výsledek implementace lze pozorovat na obrázku 3.2.

3.2.4 Přehled produktů

Pro přehled produktů jsem se využil stránkované tabulky. Vuetify pro tento účel obsahuje komponentu *VDataTableServer*. Projekt Kangaroo využíval znovupoužitelnou komponentu tabulky, prvním cílem implementace bylo tedy

Obrázek 3.2 Výsledek implementace – správce souborů



tuto komponentu opravit. Na rozdíl od projektu Porcupine obsahují prakticky všechny hlavní endpointy stránkování a filtrování – prvním krokem bylo tedy vymyslet, jak v rámci tabulky tvořit filtry ve správném formátu.

Realizované řešení spočívalo v použití definice hlavičky sloupečku pro tabulku. Do jejího rozhraní jsem přidal atribut, který realizoval tvorbu konkrétních filtrů.

Výpis kódu 3.19 Rozhraní hlavičky tabulky

```
interface Header {
  label?: string;
  align?: string;
  sortable?: boolean;
  sort?: (left: string, right: string) => number;
  customFilter?: Filter;
}
```

Tabulka pak přijme pole hlaviček, které použije pro přidání správných vstupů pro vytvoření vstupních formulářů pro filtrování. Řazení je pak realizováno přímo komponentou *VDataTableServer*. Při každé změně řazení nebo filtrů se pak provede požadavek na server s konkrétní konfigurací parametrů. Pro vytvoření query parametrů filtrů ve správném formátu nám společnost Jagu poskytla interní balíček, který pracuje ve stejném formátu jako balíček pro stránkování a filtrování na backendu.

Výpis kódu 3.20 Příklad konfigurace sloupce pro tabulku

```
...
sku: {
  sortable: true,
```

```
  customFilter: {
    filterType: tableFilters.TEXT,
    filterFunction: (input: string) => {
      return {
        [APIFilterOP.LIKE]: {
          sku: input,
        },
      };
    },
  },
},
...
```

[3]

Zbytek funkcionality tabulky šel více či méně přepoužít z původního řešení. Výsledná tabulka se tak vlastně neliší od obrázku 2.1 z hlediska vnějšího vzhledu.

3.2.5 Detail produktu

Nejsložitější částí, kterou jsem v rámci produktového katalogu implementoval byla sekce detailu produktu. V rámci této sekce jsou vystaveny k úpravě všechny důležité atributy produktu. Celá sekce obsahuje celkem 6 tabů – 2 hlavní, ve kterých se nachází základní data pro produkt, dále tab ovládající nastavování slev pro daný produkt a nakonec tab zajišťující vytváření variant produktu. Zbylé 2 taby se zabývají objednávkami a parametry, které řeší kolegové paralelně.

Nejprve bych chtěl zmínit způsob, jakým spolu všechny tyto taby fungují. V rámci analýzy bylo zjištěno, že je potřeba celou sekci detailu produktu ukládat jedním společným tlačítkem. Mít všechna data uložená v nějaké kořenové komponentě a následně je sdílet mezi komponentami je tedy nutnost. Základním způsobem, jak toto lze provést je data předávat mezi rodičem a potomkem přímo. Nicméně, to by zvýšilo komplexitu jednotlivých komponent a lepším způsobem je tedy použít *Vuex*. Tato knihovna se v projektu již používá a slouží jakožto centralizovaný bod, ve kterém lze ukládat data. Pro taby produktů do něj tedy uložím všechny potřebné prvky a následně je v aplikaci budu získávat a upravovat.

3.2.5.1 Hlavní formuláře detailu produktu

První 2 hlavní taby byly v aplikaci již v nějakém stavu připravené, tudíž základní strukturu bylo možné použít. V rámci implementace jsem postupoval iterativně – nejprve jsem upravil taby tak, aby alespoň fungovaly, následně jsem opravil nalezené chyby a nakonec jsem se zabýval implementací odlišností

[3] „sku“ je atribut, který reprezentuje EAN. Tato konfigurace tedy umožňuje vyhledávání produktů podle EAN.

od původního návrhu a úprav, které vyvstaly po uživatelském testování v minulosti.

Při napojování na Pangolin jsem objevil zajímavou chybu. Pro získání všech potřebných dat je potřeba provést zhruba 20 požadavků na server. Tyto požadavky se provádějí paralelně, a občas selhaly. Toto chování bylo nepředvídatelné – desetkrát se mi podařilo všechna data načíst v pořádku a při jedenáctém pokusu se jeden z requestů nepodařil. Chyba, kterou backend hlásil byla následující –

Výpis kódu 3.21 Chyba náhodně vznikající při požadavcích na server

```
Exception message:An exception has been raised that is
likely due to a transient failure. Consider enabling
transient error resiliency by adding
'EnableRetryOnFailure()' to the 'UseMySql' call.
```

Udělal jsem tedy to, co tato chyba doporučovala, a přidal *EnableRetryOnFailure()* do inicializace backendu. To okamžitě vedlo na další chybu, která se tentokrát neobjevovala náhodně, nýbrž při každém požadavku –

Výpis kódu 3.22 Chyba po přidaní EnableRetryOnFailure()

```
System.InvalidOperationException: The configured execution
strategy 'SqlServerRetryingExecutionStrategy' does not
support user-initiated transactions.
```

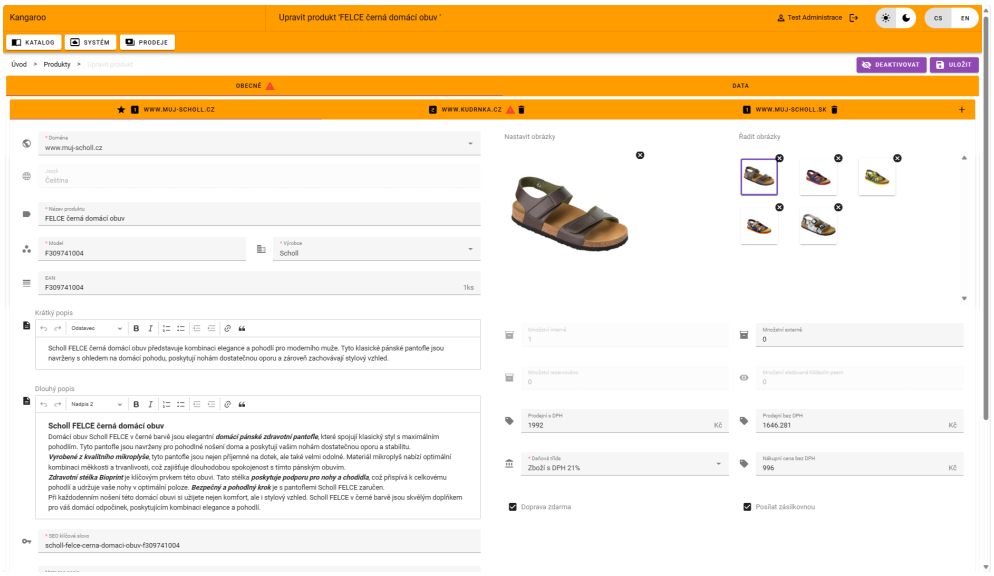
Odstranit námi definované transakce nebyla možnost a já jsem tedy problém zkoumal dál. Ukázalo se, že problém je v tom, že příkaz *EnableRetryOnFailure()* každou jednu operaci provedenou nad databází považuje za jednotku práce. Nad tímto konstruktem není přímo možné definovat transakci. Řešení spočívá v ručním vytvoření strategie provádění, do kterého je pak třeba vložit víceméně celý transakční kód. Toto řešení bylo funkční, avšak moc se mi nelíbilo ze dvou důvodů. Prvním důvodem bylo, že „*Povolení opakování při selhání způsobí interní ukládání sady výsledků EF do vyrovnávací paměti, což může výrazně zvýšit požadavky na paměť pro dotazy vracející velké sady výsledků.*“[55] Také jsem po dalším zkoumání zjistil, že vznik původní chyby byl nejspíše způsoben lokálním prostředím, ve kterém aplikace běžely, a že chyba vznikala při vytvoření nového připojení k databázi. Problém jsem tak nakonec vyřešil povolením poolingu připojení pro databázi.

Výsledek implementace pro tab Obecné lze pozorovat na obrázcích 3.3 a 3.4, a na obrázcích 3.5 a 3.6 pro tab Data.

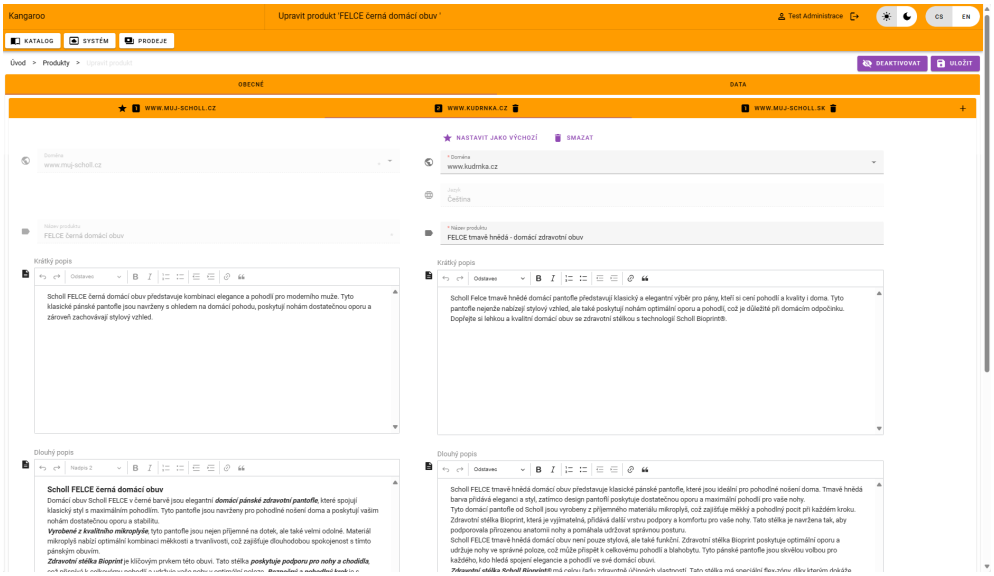
3.2.5.2 Cenové akce a varianty produktu

Na rozdíl od hlavních formulářů tyto části nebyly v předchozích pracech implementovány. Protože tyto taby nejsou tak komplexní, jejich implementace nezabrala příliš mnoho času. Nejsložitější na této implementaci bylo napojit

Obrázek 3.3 Výsledek implementace – detail produktu – tab Obecné pro hlavní doménu



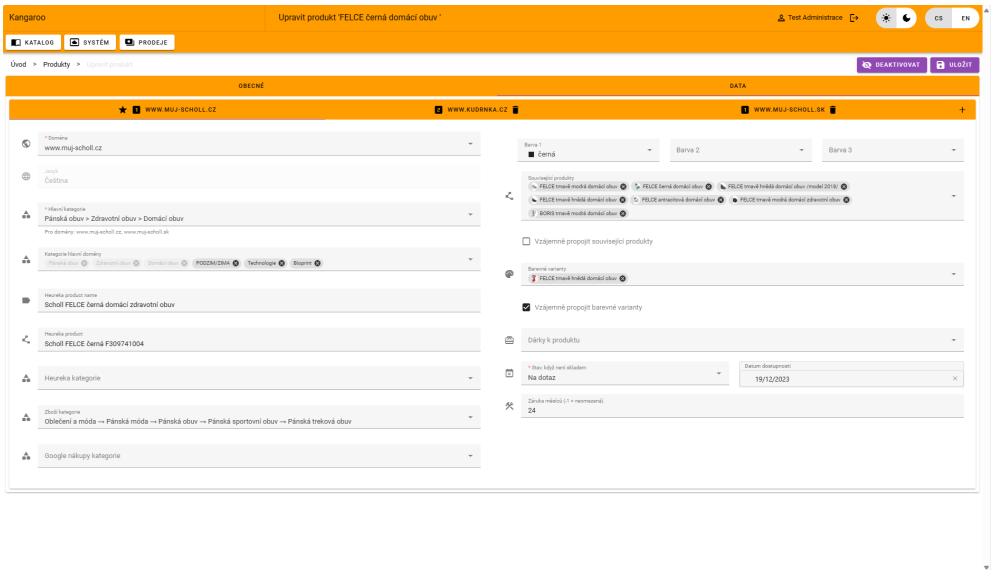
Obrázek 3.4 Výsledek implementace – detail produktu – tab Obecné pro sekundární doménu



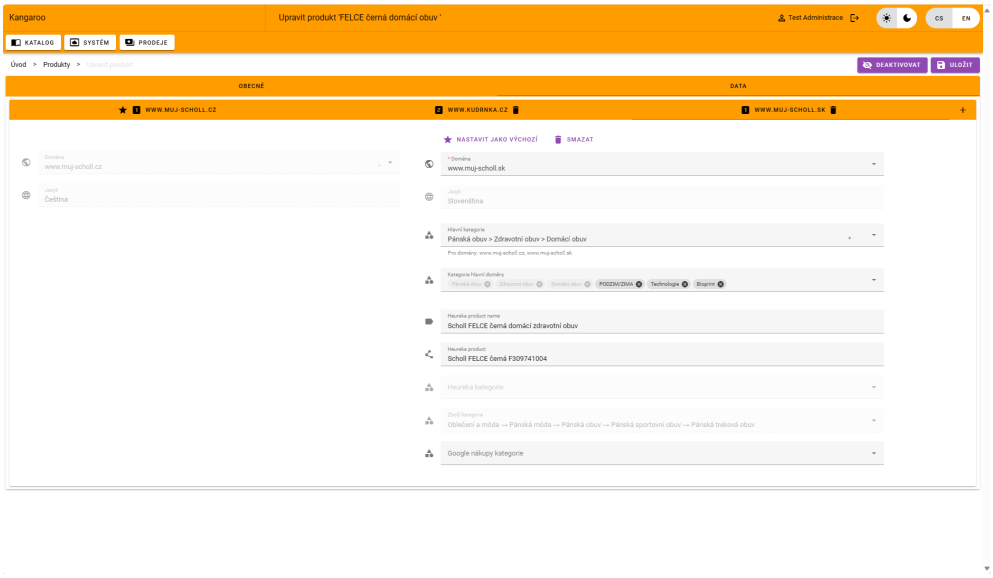
tyto formuláře k prvním 2 tabům, protože stále je vyžadováno jedno centrální tlačítko pro uložení.

Výsledná podoba implementace lze pozorovat na obrázcích 3.7 a 3.8.

Obrázek 3.5 Výsledek implementace – detail produktů – tab Data pro hlavní doménu



Obrázek 3.6 Výsledek implementace – detail produktů – tab Data pro sekundární doménu

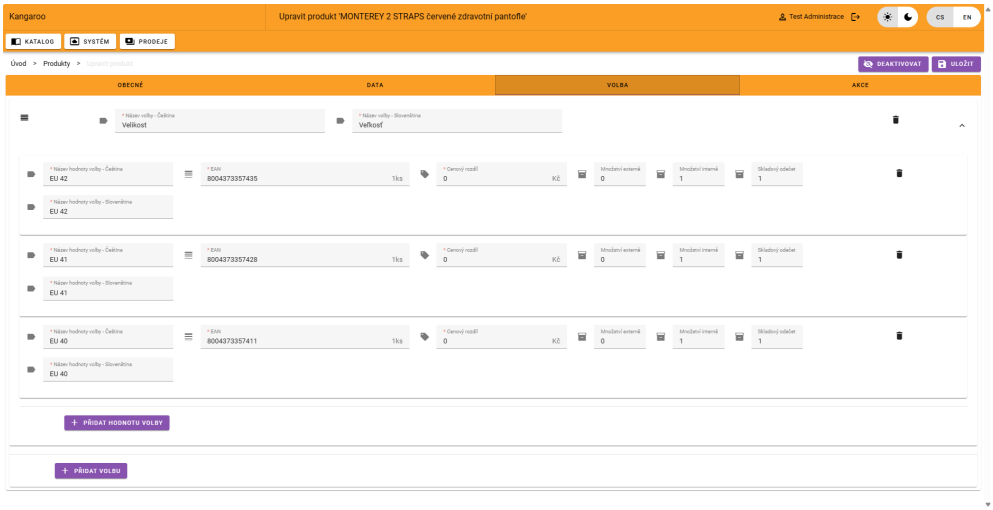


3.2.6 Souhrn

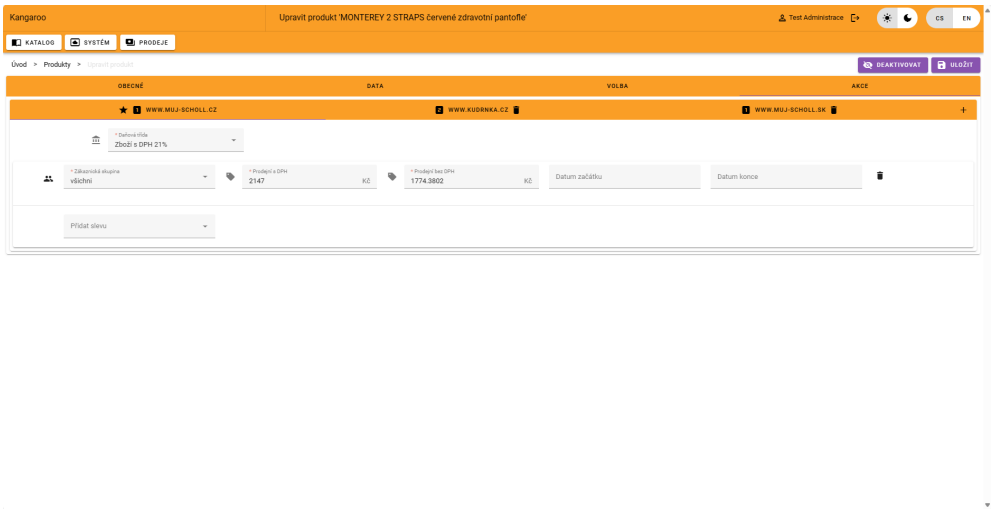
Výsledkem implementace je produktový katalog administrace. Aplikace je napojena na backend a umožňuje správu produktů, jejich cen a jejich variant.

Podobně jako u backendu však je potřeba vzniklé řešení dále rozvíjet. Prvním námětem do budoucna může být nahrazení knihovny *Vuex*. Pro pro-

Obrázek 3.7 Výsledek implementace – detail produktů – tab Volba



Obrázek 3.8 Výsledek implementace – detail produktů – tab Akce



jekty ve Vue 3 je v současnosti doporučována knihovna *Pinia*. Ta se považuje za snažší na použití, její rozhraní lépe zapadá do standardu Composition API a má lepší podporu pro TypeScript.

Dále je samozřejmě třeba dokončit i zbylé části administrace. Na velké části z nich pracují v současnosti kolegové Vojtěch Beneš, Vít Urban a Filip Dubják, nicméně některé menší části bude stále potřeba dokončit.

Další náměty souvisí s výsledky uživatelského testování v další kapitole, tyto budu tedy prezentovat až tam.

Kapitola 4
Testování

Důležitým požadavkem na systém bylo všechny komponenty otestovat. V této kapitole se budu zabývat testováním vytvořené implementace. Podobně, jako kapitoly předchozí, rozdělím text na sekci ohledně testování backendu a frontendu.

4.1 Testování projektu Pangolin

U backendové části požadavek testování nutně vede na nějakou formu automatizovaných, nebo alespoň poloautomatizovaných testů. Testovat totiž rozhraní napřímo, posíláním jednotlivých požadavků, je velmi neefektivní a zdlouhavé. Pojďme se tedy podívat na jednotlivé formy testů, které byly použity pro backendovou komponentu.

4.1.1 Testování v průběhu vývoje

4.1.2 Statická analýza

První automatickou formou testů, která byla pro backendovou část projektu použita je tzv. statická analýza.

„Statická analýza je proces zhodnocení kódu za účelem dosáhnutí požadovaného standardu. Kód se v tomto případě nespouští, je pouze analyzován na základě porovnání s best practices, je hodnocen jeho code style a jsou nacházeny potenciální chyby.“[56]

Tento způsob testování je často integrován už přímo do vývojového prostředí.

4.1.3 Integrační testy

Je vhodné také otestovat reálný běh aplikace. V případě komponenty jako je tento backend, tudíž nějakého CRUD rozhraní komunikujícího s databází,

vnímám unit testy jako víceméně zbytečné. Velké množství funkcionalit moderních frameworků pracujících s databází je integrováno těsně a je tedy problematické databázovou vrstvu simulovat. Rozhodl jsem se tedy testování řešit až na integrační úrovni, tzn. s funkční databází.

„Integrační testování je typ softwarového testování, kde se různé jednotky, moduly nebo komponenty testují jako kombinovaná entita.“[57]

V rámci projektu tedy vznikl testovací podprojekt, v rámci kterého jsou implementovány integrační testy. Tyto testy fungují tak, že sestaví a spustí danou webovou aplikaci a následně vůči ní posílají HTTP požadavky. Z pohledu vývoje se jedná o white-box testy, neboť znám implementaci testovaného kódu. Testy jsou navrženy tak, aby fungovaly nad nějakou předem připravenou databází, a se znalostí této databáze očekávají určité výsledky.

4.1.4 Gitlab CI

Posledním dílkem skládanky je použití Gitlab CI/CD.

„CI/CD je metoda vývoje softwaru, v rámci které se kontinuálně vytváří, nasazuje a testuje. Tento iterativní proces umožňuje zmírnit pravděpodobnost, že proběhne vývoj na nefunkčním základu předchozí verze.“[58]

Pro projekt byla nakonfigurována pipeline, která automaticky spouští statickou analýzu kódu a testovací projekt s integračními testy. V rámci CI scriptu se pro tento účel připraví lokální databáze s předem připravenými daty. V rámci statické analýzy jsem integroval oficiální konfigurace *SAST* a *CodeQuality* od Gitlabu. Tyto konfigurace zkoumají známá bezpečnostní rizika v případě prvního zmíněného, a kvalitu kódu v případě druhého. Díky nim byla odhalena a opravena bezpečnostní slabina související se čtením XML souboru. Konkrétní informace o této slabině popsal ve své práci Vojtěch Beneš, který danou část kódu měl na starosti. Tuto pipeline lze do budoucna nakonfigurovat tak, aby se aplikace automaticky nasazovala na server.

4.2 Testovaní projektu Kangaroo

Testování frontendové části má oproti backendu výhodu, že obsahuje uživatelské rozhraní. Při vývoji lze pozorovat změny okamžitě při implementaci. Lze jej také testovat přímo s koncovými uživateli a na základě jejich zpětné vazby navrhnout a případně i implementovat úpravy.

4.2.1 Automatické testování

Ačkoliv se aplikace dá testovat s uživateli, vhodné je i použít nějakou automatickou formu testů. Ty typicky bývají ve formě CI pipeline, která se spouští při každém nahrání nového kódu do repozitáře. Taková pipeline již v rámci Kangaroo byla nakonfigurována a obsahovala statickou analýzu kódu. V rámci této práce nebylo nic dalšího přidáno v tomto směru. Při přidání privátního balíčku pro sestavování query parametrů pro zajištění filtrování a stránkování však tato pipeline přestala fungovat. Po zkoumání se ukázalo, že na vině je použití nástroje *yarn* pro sestavení projektu. Tento package manager měl problém s přístupem k privátnímu repozitáři v rámci Jagu gitlabu, který obsahoval zmiňovaný balíček.

Řešení spočívá v použití nějakého přímo podporovaného nástroje, jako například *pnpm*. Toto v rámci této práce provedeno nebylo a stává se to tak prvním doporučením na úpravy do budoucna, vzešlým z testování.

4.2.2 Testování s uživateli

Hlavní formou testů, kterou byla výsledná implementace testována bylo uživatelské testování. Z pohledu teorie se jedná o kvalitativní testy. Jejich hlavním výstupem je tak zpětná vazba reálných uživatelů a případná analýza jejich akcí při průchodu testovacím scénářem. Důležitým číslem je pro uživatelské testování počet testovaných uživatelů. Ve většině studií se doporučuje 5. Tento počet umožňuje identifikovat 85% všech chyb, které se podaří v rámci takového testování nalézt.[59] Já jsem se rozhodl testovat se 3 uživateli, neboť se jedná o iteraci předchozího testování.

Testování s reálnými uživateli probíhalo koncem dubna. V době testování ještě nebyly připraveny taby Akce a Volba, testování se tudíž zabývalo vytvářením a upravováním produktu a hledáním v přehledu produktů. Konkrétně uživatelé v rámci úkolu nejprve vytvořili produkt, následně nalezli a upravili jiný a nakonec hledali v rámci přehledu produktů podlě různých kritérií. Znění zadaných úloh lze nalézt v příloze.

Testování probíhalo online s použitím programu *AnyDesk* a bylo nahrávané. Nahrávky lze najít na odkazu specifikovaném na přiloženém médiu. Backend i frontend běžely lokálně na mém počítači. Pro testování jsem použil data z reálné databáze jednoho z e-shopů používajících platformu OpenCart od Jagu. Celý proces testování se tak skládal z následujících kroků –

1. S testerem jsem se spojil a poslal mu údaje potřebné k připojení přes *AnyDesk*.
2. Získal jsem souhlas s pořízením záznamu schůzky
3. Před testováním jsem se ho zeptal na informace ohledně jeho zdatnosti v problémové doméně
4. Dále jsem uživateli postupně posílal úkoly z dotazníku. V průběhu testování jsem minimalizoval svojí komunikaci, abych testerovi dal prostor.
5. Po splnění posledníhp úkolu jsem na základě dotazníku získal zpětnou vazbu
6. Poděkoval jsem za účast a ukončil jsem testování.

4.2.2.1 Testování se zadavatelem

Prvním uživatelem, který se zúčastnil uživatelského testování, byl sám zadavatel, Ing. Jiří Hunka. Toto testování mělo i sekundární cíl – vyzkoušet si proces testování a doladit případné nedostatky.

Pan Hunka má bohaté zkušenosti v problémové doméně. Zná systém staré administrace a má přehled i v rámci projektu nové administrace. Má vysokoškolské vzdělání a pohybuje se v IT sféře, očekávání tedy bylo, že ovládání aplikace mu nebude činit problém, ale byl jsem zároveň připraven i na přísné zhodnocení dodané implementace. Bohužel jsem však mojí chybou v rámci nastavování nahrávacího software nenahrál zvuk schůzky, k dispozici je tedy jen video.

Průběh testování

Prvním úkolem bylo vytvoření produktu. V první iteraci jsem si myslel, že bude vhodné úkoly posílat po jednom, abych uživatele nezahltil. Poslal jsem tedy jen první část úkolu. Tu pan Hunka splnil relativně rychle a i provedl nevědomky některé činnosti specifikované v dalších částech úkolu.

Po vyplnění informací, které považoval za důležité si pan Hunka všiml červeného trojúhelníku v navigaci formulářů, který značí nevalidní data. Pan Hunka měl ale problém nalézt, kde se chyba nachází. Po pokusu o odeslání tohoto formuláře se mu již dané vstupy vybarvily červeně a chybějící data tak doplnil už rychle.

Druhý úkol se zabýval úpravou existujícího produktu. Zde jsem již poslal v reakci na první úkol všechny informace najednou. Lehké problémy se objevily opět u zadávání obrázků pro tento produkt, protože pan Hunka neodhalil, že velký obrázek reprezentuje vstup do správce souborů. Při pokusu o nahrání také očekával funkci drag and drop přímo z formuláře produktů, což nebylo implementováno. Po otevření správce souborů se pokusil nahrát obrázek do složky, kde již obrázek se stejným jménem existoval, což vyústilo v generickou chybu. Jako neintuitivní se také ukázalo ovládání otevírání složek, kde se část

s ikonou složky používala k otevření dané složky a zbytek řádku se používal k jejímu označení. Po nápovědě se mu podařilo vybrat obrázek a přidat ho danému produktu.

Dále způsobilo zmatek mé zadání. Po uživateli jsem chtel, aby pracoval se skupinou domén číslo jedna. To ho zmátlo, protože očekával reálný název domény. Úkol byl nicméně nakonec splněn.

Třetí sada úkolů se zabývala prostředím přehledu produktů. Zmatek způsobila především nová funkcionalita, která umožňovala řadit podle více kritérií. Tato skutečnost je pak indikována jako číslo vedle řazeného sloupce. Toho si však pan Hunka nevšiml a reklamoval tuto skutečnost jako chybu aplikace. Zbytek úkolů byl zpracován rychle.

Dotazník po testování

Po testování jsem se pana Hunky zeptal na otázky dle připraveného dotazníku.

1. S aplikací byl spokojen co se týče vzhledu a rozložení funkcionalit. Nelíbilo se mu zobrazování pouze generických chyb a preferoval by zprávu s reálným popisem problému. Také uvedl, že při vytváření produktu nebyly žádné prvky označeny hvězdičkou jako povinné. Toto se mi nezdálo, avšak ukázalo se, že se jednalo o chybu v aplikaci, kde se tyto hvězdičky ukázaly až s vyplněnými daty.

2. Chyběla mu reakce na drag and drop v rámci file manageru. Nicméně uvedl, že po úpravě bude tato funkcionalita významně přístupnější, než ve staré administraci. Byl spokojený s tím, že byly provedeny některé úpravy od předchozí testované verze, jako třeba přesun nastavení kategorií ke konkrétní doméně.

3. Největší problém mu činila změna obrázku u již existujícího produktu.

4. Navrhl lepší realizaci funkce drag and drop pro správce souborů.

5. K této otázce jsem si nepoznamenal odpověď hned, a odpověď byla tak vlivem špatného nastavení nahrávacího software ztracena.

Zpětná vazba k testovacímu procesu

Jak jsem zmínil výše, tak toto pilotní testování mělo sloužit i k odlazení testovacího procesu. Ukázalo se, že mnou připravený dotazník měl několik nedostatků. Prvním z nich byly pro uživatele nesrozumitelná zadání. Tento problém jsem zmínil v popisu průběhu testování, a jeho řešení bylo konkretizovat zadání reálným názvem domény.

Také se ukázalo, že některé otázky naváděly uživatele ke správné akci. Požadavek na „Aktivaci produktu“ například přímo naváděl na tlačítko „Aktivovat produkt“. Toto jsem vyřešil rozepsáním takových otázek, aby popisovaly obecně akci, co se má provést.

4.2.2.2 Testování 2. uživatelem

Druhým uživatelem, který mi věnoval svůj čas, byl pan Libor. Ten je administrátorem e-shopů, které jsou provozovány na platformě od Jagu. Má tedy bohaté zkušenosti se starším systémem a obecně s problémovou doménou.

Pro toto druhé testování jsem již implementoval některé změny navrhované po prvním testování, jako opravu lepší zvýraznění povinných atributů.

Průběh testování

Při testování byl hovor zatížen zvukovou zpětnou vazbou. Tento problém byl relativně rušivý a neumožňoval panu Liborovi průběžně popisovat své akce. Nicméně, testování proběhlo navzdory této skutečnosti.

Při vytváření produktu zadal prodejní cenu s DPH a následně nastavil daňovou třídu. Aplikace ale bere jako základ cenu bez DPH, takže se mu přepočítala na nezaokrouhlené číslo. Dále nepochopil způsob prezentace sekundární domény. Na slovenské doméně se mu automaticky zkopíroval popisek z hlavní domény, a pan Libor si neuvědomil, že pro zadání překladu ho musí upravit na pravé straně obrazovky. Překvapilo ho, že se u produktu nastavují vazby na externí kategorie, protože ve staré administraci se nastavují pro kategorii. Nakonec nastal problém s správcem souborů, kde se nepodařilo nahrát soubory do vybrané složky. Respektive, soubory se nahrály, ale uživateli se nezobrazily. Úkol nahrání obrázků jsme tak nakonec řešili přidáním již existujících obrázků. Zde pan Libor neodhalil možnost vybrání více obrázků najednou, která ve staré administraci není.

V rámci druhého úkolu požádal o pomoc při úpravě pořadí obrázků. Neodhalil, že jsou obrázky přetahovatelné. V úkolu přidávání nehlavních kategorií ho zmátlo, že jsou ve formuláři dva vstupy pro kategorii a jednotlivé kategorie chtěl přidávat přes selektor hlavních kategorií. Vyjádřil, že nerozumí pojmu výchozí doména, ale úkol splnil.

Při testování přehledu produktů pana Libora také zmátlo řazení v tabulce podle více sloupců. Po nápovědě úkol vyřešil. Také se pokusil překliknout na seznam kategorií, který v té době nebyl implementovaný. Toto způsobilo rozbití stylu stránky, kde se ve vrchní části objevil prázdný prostor. Příčinu tohoto jsem v době testování neznal, a nebyl jsem mu tedy schopný odpovědět, proč tam tento volný prostor je. Nakonec doporučil, aby tlačítko „Zrušit filtry“ zrušilo i řazení sloupců, ne jen filtrování.

Dotazník po testování

1. V aplikaci se mu nelíbil vzhled. Především by chtěl lépe zvýraznit, v jaké části aplikace se uživatel nachází. Také by rád zvětšil obecně font v aplikaci. Navzdory tomu by preferoval, aby při zadávání dat pro různé domény viděl data ze všech domén najednou. Naopak se mu líbil správce souborů, a to navzdory chybě při testování. Dále se mu líbilo našeptávání, které ve staré administraci nefunguje tak dobře.
2. Chybí mu nějaký tab, ve kterém by spravoval externí zdroje, jako například napojení na skladový systém. Také by se mu líbilo mít možnost vytvoření náhledu, jak by produkt viděl v e-shopu zákazník. Toto však není ani ve staré administraci. Jinak vyjádřil spokojenost.
3. Některé úkoly nebyl schopen vyřešit intuitivně hned, ale uvedl, že po zjištění, jak má daná funkcionalita fungovat, není žádný úkol těžký.
4. Tato a poslední otázka byla již víceméně obsažena v předchozích odpovědích, takže jsem je již nekladl.

4.2.2.3 Testování 3. uživatelem

Třetím uživatelem byla paní Veronika. Ta se také zabývá administrací e-shopu.

Průběh testování

Při vytváření si hned všimla hvězdičky označujícím povinné atributy. Podobně, jako pan Libor, zadala paní Veronika jako první cenu prodejní a až následně upravila daňovou třídu. Cena s DPH se tedy změnila, ačkoliv očekávala změnu ceny bez DPH. Následně ji zmátlo hlavní rozhraní detailu produktu, kde si neuvědomila, že *Data* je tab, na který může kliknout a vyplnit další data. Nakonec byl problém aktivovat vytvořený produkt. Očekávala, že stačí kliknout na tlačítko „Aktivovat“. Nicméně, změna je pak třeba ještě uložit. Nakonec neodhalila nahrávání obrázků a místo toho vybrala již existující.

Při úpravě produktu nastal problém při nastavování hlavní kategorie. Lví podíl na tom měla skutečnost, že jsem v aplikaci zapomněl z předešlého testování domény v opačném pořadí. Paní Veronika pak nastavila primarní doménu na vstup, který v systému existuje kvůli chybě databáze. Vlivem toho se ji nezobrazovaly žádné kategorie pro výběr, protože žádná kategorie nebyla definována pro onu primární doménu. Bohužel si nevšimla návodné zprávy v selektoru kategorie a potřebovala poradit. Od tlačítka „Uložit“ očekávala přesměrování do přehledu produktů, podobně jako při vytváření produktu.

Při navigaci v přehledu produktů nastalo již tradiční zmatení nad multisortem, který tabulka podporuje. Zbytek testování proběhl bez problémů.

Dotazník po testování

1. Paní Veronice se nelíbilo chování tlačítka „Uložit“ při úpravě produktu a chování tlačítka „Aktivovat“. Očekávala by okamžitou aktivaci. Také se jí zdálo, že není dostatečně zvýrazněné, kde se uživatel v aplikaci nachází. Líbilo se jí, že je aplikace vzhledově podobná, jako aplikace skladového systému.
2. Neoznačila žádnou chybějící část.
3. Největší potíže působil koncept výchozí domény.
4. Tato byla již víceméně obsažena v předchozích odpovědích, takže jsem je již nekladl.
5. Paní Veronika uvedla, že další komentář nebyl potřeba.

4.3 Výsledky uživatelského testování

V této sekci rozepíši chyby a zpětnou vazbu získanou uživatelským testováním. Tyto výstupy seřadím podle priority opravy či implementace. V každém podseznamu také jako první uvedu a tak označím již implementované změny.

4.3.1 Chyby

Vysoká priorita –

- Někdy se nahrané soubory nepřidají na frontendu do vybrané složky. V řešení této chyby jsem učinil určité kroky a od té doby jsem tuto chybu nezaznamenal. Nejsem si ale jistý jejím vyřešením.
- Ne chybou vzešlou přímo z testování, ale z úprav vázajících se na testování bylo rozbití správce souborů, když uživatel přetáhl nějaký objekt, který neexistoval v daném stromu. Toto bylo opraveno.

Střední priorita –

- Tlačítko „Zrušit filtry“ by mělo rušit i řazení.
- Prodejní cena se automaticky nepřepočítává při změně domény na doménu s jinou měnou na cenu s touto měnou.

Nízká priorita –

- Hvězdička označující povinný vstup se zobrazovala jen při vyplnění daného vstupu. Toto bylo opraveno.
- Zvýrazněná hlavní doména v přehledu produktů má málo kontrastní barvu v tmavém režimu.

4.3.2 Návrhy na zlepšení

Vysoká priorita –

- Správce souborů by měl lépe reagovat na drag and drop souborů. Toto bylo již implementováno. Správce se nyní otevře při přesunutí souboru na něj.
- Chyby při validaci přijatých dat na backendu by se měly mapovat na konkrétní formulářová pole. Backend tyto chyby vrací ve standardizovaném formátu.
- Filtrování produktů podle data umožňuje vybrat jen jeden den, je vhodné přidat novou komponentu, která umožní vybrat časový úsek a další operace.
- Aplikace by měla mít tlačítko pro duplikaci současných dat do formuláře pro vytvoření nového produktu.

Střední priorita –

- Vnější ikona pro vložení obrázku by měla uživateli dát vědět, že se používá pro přidání souboru. Toto bylo implementováno akcí při přejetí myší nad danou komponentou.
- Zdroj při výpočtu ceny pří změně datové třídy by měl být cena s DPH. Toto je cena, co vidí zákazník jako první v e-shopu. Toto již bylo implementováno.
- Aktivace produktu by měla mít svůj vlastní endpoint, který by sloužil pouze pro aktivaci produktu.

Nízká priorita –

- Pole ve formuláři pro rozměry by mělo specifikovat, že se jedná o rozměry balení.
- Administrace by pro produkty měla mít další tab, který ovládá synchronizaci s případnými dalšími systémy.
- Administrace produktu by mohla zobrazovat náhled daného produktu na reálném zákaznickém frontendu.

Kapitola 5
Závěr

Cílem této práce bylo vytvořit funkční část administrace e-shopu, vybudovanou nad existující databází původního řešení při zajištění kompatibility při paralelním běhu obou aplikací. V rámci této práce konkrétně měly proběhnout analýza, návrh, implementace a otestování produktového katalogu administrace. Nový systém měl na většině míst kopírovat funkčnost staršího řešení, na vybraných místech jej měl však i rozšiřovat. Vývoj probíhal ve čtyřčlenném týmu, což si vyžádalo přihlédnutí k potenciálním problémům týmové spolupráce. Celá práce navazovala na předchozí řešení, bylo tudíž potřeba i zhodnotit výsledky předchozích prací a i podle nich navrhnout nejlepší cestu ke splnění zadání.

Tento cíl se podařilo splnit. V rámci první kapitoly – analýzy – byl zhodnocen stav projektu nové administrace po předcházejících závěrečných pracech. V rámci této analýzy bylo rozhodnuto, že vývoj backendové části proběhne v jazyke C# a nebude navazovat na předchozí implementaci v jazyce PHP. Oproti tomu vývoj frontendové části využije již existujícího projektu, napsaného v typescriptu s použitím frameworku Vue. Byl zhodnocen i způsob zpracování v týmu v rámci předchozích závěrečných prací a na jeho základě bylo rozhodnuto o vhodné formě spolupráce při vývoji, která se na rozdíl od předchozích iterací zaměřovala na eliminaci vzájemných závislostí jednotlivých členů v týmu mezi sebou. Nakonec byly specifikovány funkční a nefunkční požadavky, které sloužily jako vstup do následující kapitoly.

V rámci následující kapitoly probíhal návrh samotné aplikace. Tento návrh byl rozdělen na 2 hlavní části – návrh backendu a následně návrh frontendu. V rámci návrhu backendové části bylo nejprve rozhodnuto o celkové architektuře systému. Výsledná aplikace se tedy skládá ze 3 hlavních částí – autorizační server a server poskytující rest API pro správu administrace na straně backendu, a konzument onoho rozhraní na straně frontendu. Při návrhu backendové části byl kladen důraz především na rozdíly oproti návrhům provedeným v rámci předchozích závěrečných prací. Návrh frontendové části se zabýval především uživatelským rozhraním, neboť se jednalo o rozšíření již existující aplikace.

Největší částí závěrečné, alespoň tedy co se objemu vykonané práce týče, byla realizace. V rámci té byl vytvořen nový backend a rozšířen existující frontend e-shopové administrace. Z pohledu textu závěrečné práce byly především zmíněny konkrétní ukázky použitých technologií.

Výsledek implementační části byl také otestován. Pro backendovou část byly realizovány integrační testy v rámci Gitlab CI pipeline. Ty by měly zajistit, že další provedené změny v rámci tohoto projektu nezpůsobí vytvoření chyb v jiných částech systému. Po vytvoření frontendové části bylo provedeno uživatelské testování, jehož výstupem byl seznam chyb a požadavků na úpravu. Některé změny či požadavky byly zpracovány ještě v rámci této závěrečné práce, ostatní jsou připraveny ke zpracování v budoucnu.

Hlavním výstupem této práce je tedy aplikace administrace e-shopů, respektive její část zabývající se administrací produktového katalogu. Aplikace zatím není nasazená na reálném serveru, nicméně je schopna paralelního běhu se starou administrací a realizuje funkčnost specifikovanou v rámci zadání. Kolegové Vojtěch Beneš, Vít Urban a Filip Dubják dodali části administrace zabývající se administrací výrobců, kategorií, produktových parametrů a objednávek. Tyto části společně pokrývají hlavní funkčnosti systému. Díky konkrétnímu popisu použitých řešení v rámci kapitoly implementace a zaměření na psaní udržitelného a rozšiřitelného kódu by mělo být možné na tuto práci navázat bez větších potíží a doplnit chybějící části administrace.

Příloha A

Dotazník pro uživatelské testování

A.1 Úkoly pro uživatele

A.1.1 Vytvoření produktu

1. Vytvořte nový produkt “Sandály” pro doménu www.muj-scholl.cz a www.muj-scholl.sk.
2. Vyplňte informace, které považujete za důležité.
3. Nahrajte alespoň 5 nových obrázků do složky “data/testing” a nastavte je k tomu produktu. Obrázky jsou k dispozici ve složce “Stažené soubory”.
4. Nastavte hlavní kategorii na “Pánská obuv ¿ Zdravotní obuv ¿ Sandály”.
5. Přidejte vhodný související produkt.
6. Uložte produkt.

A.1.2 Úprava existujícího produktu

1. Nalezněte produkt „Ariete Espresso Coffee Machine 1312“ v přehledu produktů.
2. Změňte hlavní obrázek na obrázek zobrazující fotografii stříbrného kávovaru (tento obrázek je obsažen v seznamu obrázků daného produktu).
3. Pro doménu www.kudrnka.cz přidejte kategorie Překapávače a Čistič oken.
4. Nastavte doménu www.moje-ariete.cz jako výchozí doménu.

5. Produkt se v současnosti nezobrazuje v eshopu pro zákazníky. Zajistěte, aby se zobrazoval.
6. Uložte změny.

A.1.3 Vyhledávání

1. Nalezněte naposledy upravené produkty.
2. Nalezněte produkty s nejvíce shlédnutími prodávané na doméně www.muj-beuer.cz.
3. Nalezněte libovolnou váhu.
4. Nalezněte a odstraňte vámi vytvořený produkt.
5. Zjistěte, kolik kusů je potřeba dokoupit k "Ariete Espresso Coffee Machine 1312"

A.2 Dotazník po

1. Co se vám v aplikaci líbilo a co naopak nelíbilo?
2. Jak jste spokojeni s funkčnostmi aplikace, chybí vám něco?
3. Který úkolů vám dělal největší potíže?
4. Máte nějaké nápady na zlepšení/úpravu funkčnosti?
5. Máte nějaký další komentář?

Bibliografie

1. KOUBA, Alois. *Backend administrace e-shopu*. 2022. Dostupné také z: https://dspace.cvut.cz/handle/10467/102088.
2. *OpenCart - Open Source Shopping Cart Solution*. [B.r.]. Dostupné také z: https://www.opencart.com/.
3. *OpenCart*. [B.r.]. Dostupné také z: https://en.wikipedia.org/wiki/OpenCart.
4. *Technical Debt*. [B.r.]. Dostupné také z: https://www.productplan.com/glossary/technical-debt/.
5. *InnoDB vs MyISAM: A Detailed Comparison of Two MySQL Storage Engines*. [B.r.]. Dostupné také z: https://blog.devart.com/myisam-vs-innodb.html.
6. *SQL FOREIGN KEY Constraint*. [B.r.]. Dostupné také z: https://www.w3schools.com/sql/sql_foreignkey.asp.
7. NOVÁČEK, Tomáš. *MIGRACE CUSTOMIZOVANÝCH E-SHOPŮ OPENCART 1.1*. 2019. Dostupné také z: https://dspace.cvut.cz/handle/10467/80253.
8. EVSEENKO, Iuliia. *Frontend administrace e-shopu*. 2021. Dostupné také z: https://dspace.cvut.cz/handle/10467/94629.
9. KOUDELA, Radomír. *API pro backend eshopu*. 2021. Dostupné také z: https://dspace.cvut.cz/handle/10467/92867.
10. HOJEK, Tomáš. *Backend a CI administrace e-shopu*. 2022. Dostupné také z: https://dspace.cvut.cz/handle/10467/107230.
11. DVOŘÁK, Martin. *Frontend administrace e-shopu*. 2022. Dostupné také z: https://dspace.cvut.cz/handle/10467/102105.
12. BABÁK, Jan. *Frontend administrace e-shopů*. 2022. Dostupné také z: https://dspace.cvut.cz/handle/10467/102225.

13. GOLMGREN, Nikita. *Backend e-shopu - nákupní košík a zpracování objednávek*. 2021. Dostupné také z: https://dspace.cvut.cz/handle/10467/109631.

14. MAREŠ, David. *Backend e-shopu – doprovodné procesy*. 2023. Dostupné také z: https://dspace.cvut.cz/handle/10467/109835.

15. *What is PHP*. [B.r.]. Dostupné také z: https://www.php.net/manual/en/intro-whatis.php.

16. *What is Symfony*. [B.r.]. Dostupné také z: https://symfony.com/what-is-symfony.

17. *A tour of the C# language*. [B.r.]. Dostupné také z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

18. *Přehled ASP.NET Core*. [B.r.]. Dostupné také z: https://learn.microsoft.com/cs-cz/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0.

19. MO, Joshua. *Why Type Safety is Important*. [B.r.]. Dostupné také z: https://www.shuttle.rs/blog/2023/11/29/type-safety.

20. SHAH, Jigar. *C# vs PHP for Web Development*. [B.r.]. Dostupné také z: https://wpwebinfotech.com/blog/c-vs-php-for-web-development/.

21. *Typescript*. [B.r.]. Dostupné také z: https://www.typescriptlang.org/.

22. *Vue.js*. [B.r.]. Dostupné také z: https://vuejs.org.

23. SALAJ, Martin. *Frontend administrace e-shopu - objednávky*. 2023. Dostupné také z: https://dspace.cvut.cz/handle/10467/110102.

24. *Options API vs Composition API*. [B.r.]. Dostupné také z: https://vueschool.io/articles/vuejs-tutorials/options-api-vs-composition-api/.

25. *Vue.js - Introduction*. [B.r.]. Dostupné také z: https://vuejs.org/guide/introduction.html.

26. *ASP.NET Core Blazor*. [B.r.]. Dostupné také z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0&WT.mc_id=dotnet-35129-website.

27. HILTON, Jon. *Blazor vs Vue*. [B.r.]. Dostupné také z: https://www.telerik.com/blogs/blazor-vs-vue-web-developers.

28. *What is Full Stack?* [B.r.]. Dostupné také z: https://www.w3schools.com/whatis/whatis_fullstack.asp.

29. BILIAWSKA, Julia. *Full Stack vs. Specialized Developer*. [B.r.]. Dostupné také z: https://distantjob.com/blog/full-stack-vs-specialized-developer/.

30. DIAS, Renan Benatti. *When Can I Call Myself a Senior Developer?* [B.r.]. Dostupné také z: https://www.kodeco.com/38327766-when-can-i-call-myself-a-senior-developer.
31. MONNIER, Fred. *Software Development Manager: What They Do*. [B.r.]. Dostupné také z: https://www.revelo.com/blog/software-development-manager.
32. *How to do a code review*. [B.r.]. Dostupné také z: https://google.github.io/eng-practices/review/reviewer/.
33. *SDLC Tutorial*. [B.r.]. Dostupné také z: https://www.tutorialspoint.com/sdlc/index.htm.
34. *SDLC - Iterative Model*. [B.r.]. Dostupné také z: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm.
35. *Software Engineering — Classification of Software Requirements*. [B.r.]. Dostupné také z: https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/.
36. *Functional vs Non Functional Requirements*. [B.r.]. Dostupné také z: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/.
37. *What is FURPS+?* [B.r.]. Dostupné také z: https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/.
38. *Overview of ASP.NET Core authentication*. [B.r.]. Dostupné také z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-8.0.
39. *Introduction to Identity on ASP.NET Core*. [B.r.]. Dostupné také z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio.
40. *Keycloak*. [B.r.]. Dostupné také z: https://www.keycloak.org/docs/latest/server_admin/index.html.
41. *What is OIDC?* [B.r.]. Dostupné také z: https://www.microsoft.com/en-us/security/business/security-101/what-is-openid-connect-oidc.
42. *What is OAuth 2.0?* [B.r.]. Dostupné také z: https://auth0.com/intro-to-iam/what-is-oauth-2.
43. MIZRACHI, Aviad. *OAuth Grant Types: Explained*. [B.r.]. Dostupné také z: https://frontegg.com/blog/oauth-grant-types.
44. *Introduction to JSON Web Tokens*. [B.r.]. Dostupné také z: https://jwt.io/introduction.

45. MAITI, Anupam. *AccessToken Vs ID Token Vs Refresh Token - What? Why?When?* [B.r.]. Dostupné také z: https://www.c-sharpcorner.com/article/accesstoken-vs-id-token-vs-refresh-token-what-whywhen/#:~:text=Access%20tokens%20are%20used%20in,which%20must%20be%20a%20JWT..

46. *Validate JSON Web Tokens.* [B.r.]. Dostupné také z: https://auth0.com/docs/secure/tokens/json-web-tokens/validate-json-web-tokens.

47. *RFC 7662: Token Introspection.* [B.r.]. Dostupné také z: https://oauth.net/2/token-introspection/.

48. *MVC architektura.* [B.r.]. Dostupné také z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.

49. *MinIO Documentation.* [B.r.]. Dostupné také z: https://min.io/docs/minio/kubernetes/upstream/.

50. *What is Object storage?* [B.r.]. Dostupné také z: https://cloud.google.com/learn/what-is-object-storage.

51. *Object vs File Storage: When and Why to Use Them.* [B.r.]. Dostupné také z: https://blog.purestorage.com/purely-informational/object-vs-file-storage-when-and-why-to-use-them.

52. *Asynchronous programming scenarios* [online]. [B.r.]. [cit. 2022-04-24]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios.

53. *openapi-typescript.* [B.r.]. Dostupné také z: https://www.npmjs.com/package/openapi-typescript.

54. *Vue-Tree-Dnd.* [B.r.]. Dostupné také z: https://github.com/jcuenod/vue-tree-dnd.

55. *Connection Resiliency.* [B.r.]. Dostupné také z: https://learn.microsoft.com/en-us/ef/core/miscellaneous/connection-resiliency#execution-strategies-and-transactions.

56. *Static Testing: What You Need to Know.* [B.r.]. Dostupné také z: https://www.cprime.com/resources/blog/static-testing-what-you-need-to-know.

57. AWATI, Rahul. *integration testing or integration and testing.* [B.r.]. Dostupné také z: https://www.techtarget.com/searchsoftwarequality/definition/integration-testing.

58. *Get started with GitLab CI/CD.* [B.r.]. Dostupné také z: https://docs.gitlab.com/ee/ci/.

59. STRBA, Marek. *Usability Testing Studies: How Many Participants?* [B.r.]. Dostupné také z: https://blog.uxtweak.com/usability-studies-how-many-participants-are-enough/.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `openapi.yaml` ..... vygenerovaná dokumetace ve formátu OpenAPI
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF