FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Parametry produktů v administraci e-shopu
Student:	Filip Dubják
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je ve spolupráci s Vojtou Benešem, Bc. Vítem Urbanem a Aloisem Koubou 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 komplexní řešení parametrů produktů využívaných pro filtraci a kategorizaci zboží. Práce klade důraz na budoucí snadné použití v administraci.
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. Proveďte důkladnou analýzu zaměřenou na parametry produktů včetně jejich doprovodných procesů ve všech částech administrace s ohledem na budoucí využití v zákaznickém e-shopu.
2. Navrhněte vhodný celek frontend a backend parametrové části administrace.
3. Návrhy řádně konzultujte se zadavatelem.
4. Implementujte kompletní a použitelnou parametrovou část administrace.
5. Navrhněte a realizujte vhodné formy testů frontend a backend části vaší realizace.
6. Otestujte výslednou realizaci buď v ostrém provozu, nebo alespoň s reálnými uživateli.
7. Zhodnoťte výsledné řešení, navrhněte úpravy do budoucna.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 7. února 2024 v Praze.

Bakalářská práce

PARAMETRY PRODUKTŮ V ADMINISTRACI E-SHOPU

Filip Dubják

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: Ing. Jiří Hunka
16. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především vedoucímu mé bakalářské práce – Jiřímu Hunkovi – za jeho cenné konzultace, věnovaný čas a podporu v průběhu vypracování celé práce. Dále bych rád poděkoval svým kolegům – Aloisovi Koubovi, Vojtěchu Benešovi a Vítku Urbanovi – za jejich spolupráci a pomoc při realizaci této práce. A nakonec, bych chtěl vyjádřit vděčnost své rodině za jejich trvalou podporu a povzbuzení 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 16. května 2024

Abstrakt

Práce se zaměřuje na inovaci a zdokonalení administrace e-shopů s cílem zlepšit uživatelský zážitek a efektivitu práce s parametry produktů a kategorií. Společně s týmem jsme vyvinuli více doménovou e-shop administraci propojenou s existující databází, přinášející moderní a efektivní řešení. Byl navržen a implementován nový přístup k správě parametrů, zvyšující efektivitu a uživatelskou přívětivost. I přes dosažený úspěch je však stále prostor pro vylepšení, zejména v oblasti řazení produktových a kategorických parametrů. Rozvoj aplikace v budoucnosti bude reakcí na uživatelskou zpětnou vazbu a pokračování ve zdokonalování administrace e-shopu v souladu s potřebami uživatelů.

Klíčová slova C#, JavaScript, e-shop, parametry produktů e-shopu, uživatelská přívětivost, návrh, administrace

Abstract

The work focuses on the innovation and improvement of e-shop administration in order to improve the user experience and the efficiency of work with product and categorical parameters . Together with the team, we developed a multi-domain e-shop administration linked to an existing database, bringing a modern and efficient solution. A new approach to parameter management was designed and implemented, increasing efficiency and user friendliness. However, despite the success achieved, there is still room for improvement, especially in the area of sorting product and categorical parameters. The development of the application in the future will be a response to user feedback and the continuation of improving the administration of the e-shop in accordance with the needs of users.

Keywords C#, JavaScript, e-shop, product parameters in e-shop, user friendliness, design, administration

Seznam zkratek

HTML	HyperText Markup Language
UI	User interface
LINQ	Language Integrated Query
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
DOM	Document Object Model

Kapitola 1
Úvod

V současném digitálním prostředí se elektronické obchody staly nepostradatelným prvkem obchodního světa. S rostoucí konkurencí a nároky zákazníků je nezbytné neustále inovovat a zdokonalovat systémy e-shopů, aby odpovídaly aktuálním potřebám a očekáváním. Práce se tedy zaměřuje na zlepšení uživatelského zážitku a efektivity správy e-shopů prostřednictvím nového a inovativního přístupu k administraci parametrů. Motivací práce je zlepšení správy parametrů za účelu zřízení přívětivějšího a intuitivnějšího prostředí pro klienta v administraci e-shopu.

Tato bakalářská práce si klade za cíl společně s kolegy Vojtěchem Benešem, Bc. Vítem Urbanem a Bc. Aloisem Koubou, kteří jsou studenty stejné fakulty, realizovat a nasadit do použitelné formy více doménovou e-shop administraci, která bude propojena se stávající databází a umožní její paralelní provoz s již existující administrací. Nová administrace má nahradit současnou a zajistit přechod na modernější a efektivnější systém. Hlavním zaměřením práce je vytvoření komplexního řešení parametrů produktů, které jsou klíčové pro filtrování a kategorizaci zboží. Důraz je kladen na snadné a intuitivní použití tohoto systému v administraci e-shopu, což přinese uživatelům větší pohodlí a efektivitu při správě parametrů produktů.

Prvním krokem je provedení důkladné analýzy současné e-shop administrace a předchozích vývojových prací, které se zabývaly touto oblastí. Tato analýza se zaměřuje na parametry produktů a jejich využití v různých částech administrace s ohledem na budoucí integraci s e-shopem. Na základě této analýzy budou navrženy vhodné struktury pro frontendovou a backendovou část administrace, které budou zajišťovat efektivní manipulaci s parametry produktů a snadnou integraci s existujícím systémem.

Po schválení návrhů bude následovat implementace kompletní a použitelné části administrace, která umožní provozování e-shopu s novým systémem. Dále budou navrženy a provedeny vhodné testy frontendu a backendu administrace, aby byla zajištěna kvalita a spolehlivost celého řešení. V závěrečné fázi bude provedeno testování v ostrém provozu nebo s reálnými uživateli, aby byla ověřena funkčnost a uživatelská přívětivost nové administrace.

Na závěr práce bude provedeno zhodnocení dosažených výsledků a případně navrženy úpravy a vylepšení pro budoucí rozvoj a optimalizaci administrace e-shopu. Cílem této práce je tedy vytvořit komplexní a funkční řešení, které splní požadavky zadavatele a přinese zlepšení uživatelské zkušenosti s administrací na e-shopu.

Kapitola 2
Analýza

První kapitola této bakalářské práce se zaměřuje na analýzu současné situace a předchozích prací týkajících se vývoje e-shop administrace u společnosti Jagu s.r.o, s důrazem na parametry produktů a parametry kategorií a jejich role v procesu správy zboží. Cílem této analýzy je poskytnout ucelený pohled na stávající stav administrace a získat důležité poznatky pro navržení a implementaci nového systému.

V této kapitole bude podrobně zkoumána současná situace e-shop administrace, analýza vývoje v týmu, používané vývojové metodiky a analýza parametrů produktů a souvisejících procesů v administraci. Analytický přístup nám umožní identifikovat klíčové oblasti, které je třeba zlepšit, a připravit pevný základ pro návrh a implementaci nové administrace.

Dále se zaměřím na využití získaných poznatků pro budoucí vývoj, abychom zajistili, že nový systém bude efektivní, snadno použitelný a odpovídající potřebám uživatelů. Analytický proces bude zahrnovat spolupráci se členy týmu a konzultaci se zadavatelem, aby byla zajištěna shoda s požadavky a očekáváními.

Tato kapitola tedy představuje klíčový krok v procesu přípravy řešení e-shop administrativního systému pro správu parametrů, který bude sloužit jako základ pro následující fáze vývoje a implementace.

2.1 Analýza technologií

Při analýze používaných technologií, jsem zjistil, že je aktuální stav založen na využití platformy OpenCart, která byla lehce upravena s pomocí PHP frameworku Symfony. Plánovaným krokem je migrace nového backendu do .NET a pokračování v připraveném frontendovém prostředí ve Vue.js.

2.1.1 Vue.js

Vue.js je moderní JavaScriptový framework, který slouží k vytváření uživatelských rozhraní pro webové aplikace. Vue.js poskytuje možnosti pro efektivní vytváření interaktivních uživatelských rozhraní a jednoduchou integraci s existujícími projekty. Tento framework je navržen tak, aby byl snadno naučitelný a použitelný, což ho činí ideální volbou pro začínající i zkušené vývojáře. [1]

2.1.2 Symfony

Symfony je PHP framework navržený tak, aby usnadňoval vývoj robustních a škálovatelných webových aplikací. Jeho modulární architektura a bohatá sadu funkcí poskytují vývojářům ná-

stroje pro rychlé a efektivní vytváření složitých projektů. [2]

2.1.3 .NET

Microsoft .NET je výkonná platforma a ekosystém pro vývoj softwaru, který poskytuje širokou škálu nástrojů a technologií pro tvorbu různých typů aplikací. Jeho klíčovou součástí je programovací jazyk C#, který je objektově orientovaný a nabízí pokročilé funkce pro efektivní vývoj. [3]

2.2 Vývoj v týmu

Jak jsem již zmínil v úvodu, na tuto práci navazují i mí kolegové – Vojtěch Beneš, Alois Kouba a Vít Urban – kteří mají z větší části za úkol přepsat aktuální stav z PHP Symfony do C# .NET, za účelu zrychlení aplikace, více se tomuto tématu věnuje Alois Kouba ve své práci [4]. Naším úkolem bude také propojit nově vytvořený backend s novou frontendovou aplikací tvořenou ve Vue.js, která v minulosti vznikla z práce Martina Dvořáka a Jana Babáka [5, 6] a je stále ve vývoji. Vyvíjet tuto práci v týmu je klíčové pro úspěšné provedení velkého projektu z několika důvodů:

- Každý člen týmu přináší své vlastní znalosti, dovednosti a zkušenosti, což umožňuje pokrýt širokou škálu potřeb projektu. Například, zatímco někdo může být zkušený ve frontendovém vývoji, jiný může mít specializaci v databázovém návrhu nebo v testování softwaru.
- Velký projekt může být složitý a náročný na zvládnutí pro jedinou osobu. Práce v týmu umožňuje rozdělit úkoly a zátěž mezi více členů, což snižuje tlak na jednotlivého člena týmu a zvyšuje efektivitu celého procesu vývoje.
- Práce v týmu umožňuje vzájemnou podporu a spolupráci mezi členy. To znamená, že když jeden člen narazí na problém nebo potřebuje radu, může se obrátit na ostatní členy týmu, kteří mu mohou poskytnout pomoc nebo nový pohled na problém. Tato spolupráce vede k lepším a komplexnějším řešením.

Celkově lze říci, že práce v týmu je klíčová pro úspěch velkého projektu, protože umožňuje využít rozmanitost znalostí a dovedností, rozdělit práci a zátěž a podporovat vzájemnou spolupráci.

2.2.1 Podpůrné nastroje

Při vývoji v týmu jsme využívali širokou škálu nástrojů, které nám umožnily efektivní komunikaci, spolupráci a řízení projektu. Mezi klíčové nástroje patřily:

- GitLab sloužil jako hlavní platforma pro správu verzí a správu kódu. Pomocí GitLabu jsme mohli spravovat naše repozitáře, vytvářet větve, provádět revize kódu a řešit konflikty. Díky integraci s dalšími nástroji bylo možné efektivně propojit správu kódu s dalšími částmi projektu. [7]
- Slack byl hlavním nástrojem pro okamžitou komunikaci v týmu. Pomocí Slacku jsme mohli vytvářet kanály pro různá témata, posílat zprávy a sdílet soubory. To nám umožnilo udržovat pravidelnou komunikaci a sdílet informace v reálném čase. [8]
- Google Meet jsme využívali pro organizaci videokonferencí. Díky možnosti sdílení obrazovky a virtuálnímu setkávání jsme mohli efektivně diskutovat o pokroku projektu, řešit problémy a plánovat další kroky. [9]

- Redmine sloužil jako nástroj pro řízení projektu a sledování úkolů. Pomocí Redmine jsme mohli vytvářet a přiřazovat úkoly, sledovat pokrok projektu, spravovat bugy a řešit požadavky. Tento nástroj nám poskytl strukturovaný přehled o stavu projektu a umožnil efektivní správu pracovních postupů. [10]

2.2.2 Metodiky vývoje

Vzhledem k povaze týmové práce je nezbytné dohodnout se na vhodné metodice vývoje, která zajistí efektivitu a organizovanost našeho úsilí. Bez jasné metodiky by naše práce mohla být chaotická a dosažení společných cílů by bylo podstatně náročnější. Dobře zvolená metodika poskytne rámec a strukturu, která nám umožní lépe řídit proces vývoje, efektivně komunikovat v týmu a dosahovat stanovených cílů. Zároveň nám pomůže identifikovat klíčové úkoly, rozdělit práci mezi členy týmu a efektivně řešit případné problémy nebo změny v průběhu projektu.

Vývojové metodiky představují soubor postupů, pravidel a strategií, které slouží k usnadnění a řízení procesu vývoje softwarových projektů. Jejich použití přináší řadu výhod, jako je standardizace pracovních postupů, zjednodušení řízení projektu a snadnější sledování postupu vývoje. Mezi další výhody patří opakovatelnost procesů, definice požadovaných artefaktů a jejich obsahu, a také snadnější odhadování a kontrola postupu projektu.[11]

Nicméně, s použitím vývojových metodik mohou přicházet i určité nevýhody. Mezi nejčastější patří potřeba dodatečné práce, nutnost času na seznámení s metodikou, a složitost přizpůsobení metodiky konkrétním potřebám projektu. Některé metodiky mohou být placené, což může být pro některé týmy finančním omezením.[11]

Metodiky vývoje mohou být zaměřeny na různé aspekty vývoje, jako je implementace a testování, organizace práce, definice vstupů a výstupů, nebo pracovní postupy. Tyto metodiky mohou být buď obecné a definovat pravidla na vysoké úrovni abstrakce, nebo mohou být velmi detailní a podrobné. [11]

2.2.2.1 Klasické metodiky

Klasické metodiky vývoje jsou tradičními přístupy k řízení softwarových projektů, které se zaměřují na plánování, dokumentaci a konzistentní procesy. Tyto metodiky klade velký důraz na podrobnou dokumentaci všech fází projektu a pracují s pevně stanovenými procesy a postupy. Mezi hlavní charakteristiky patří široké plánování, řízení a řešení různých aspektů vývoje. Mezi nevýhody patří rigidita a nedostatek flexibility, což může zpomalit přizpůsobení se změnám. Klasické metodiky zahrnují Waterfall model, Spiral model, Unified Process (UP) a Rational Unified Process (RUP). Tyto přístupy jsou vhodné pro projekty, které vyžadují pevnou kontrolu a předem definované procesy.[11]

2.2.2.2 Agilní metodiky

Agilní metodiky vývoje jsou moderní přístupy k řízení softwarových projektů, které kladou důraz na flexibilitu, spolupráci a rychlou reakci na změny. Tyto metodiky podporují pružné plánování, častou iterativní práci a zapojení zákazníka do procesu vývoje. Mezi hlavní charakteristiky patří flexibilita, spolupráce, časté dodávky a zpětná vazba. Mezi nevýhody patří vyšší míra nejistoty a vyžadovaná silná spolupráce a angažovanost týmu. Agilní metodiky zahrnují například Scrum, Extreme Programming (XP), Kanban a Lean Development. Jsou vhodné pro projekty s neustále se měnícími požadavky a vyžadují rychlou reakci na změny. [11]

Pro náš rozsáhlý projekt, který pravděpodobně bude vyžadovat postupné změny a úpravy v průběhu vývoje, a kde není jisté, zda všechny práce členů týmu budou pokrývat kompletní analýzu dosavadní administrace, jsme se rozhodli upřednostnit agilní postup. Tento přístup nám

umožní flexibilitu a rychlou reakci na změny požadavků. Tato volba je v souladu s doporučením vedoucího práce.

Naše minimální požadavky pro tvorbu dokumentace se zaměří na důležitou definici rozhraní backendu. Zde využijeme openAPI dokumentaci[12], což nám umožní snadnější komunikaci mezi backendovou a frontendovou částí administrace a usnadní vývoj na frontendu.

2.2.3 Životní cyklus vývoje

Model životního cyklu softwarového vývoje je způsob, jakým bude software dodáván, a typicky je určen použitou metodikou. Existuje několik různých modelů, ale mají mnoho společných prvků. Pracovní postupy zahrnují analýzu a sběr požadavků, návrh architektury a detailní design, implementaci a testování, a nakonec dodání a údržbu. Klíčovou otázkou je, kdy budou tyto postupy provedeny, v jakém rozsahu a formě, v jaké souslednosti a zda najednou nebo v iteracích.[13]

Mezi tradiční modely patří vodopád, který se řídí sekvencí pracovních postupů a je vhodný pro projekty s pevnými a známými požadavky. Iterativní model představuje sekvenci ”malých vodopádů”s průběžnými dodávkami, což snižuje riziko a umožňuje změnu směru dle aktuální potřeby zákazníka. Agilní přístup je iterativní model, kde délka iterace je minimalizována a klade důraz na rychlou zpětnou vazbu a reakci na změny požadavků.[13]

2.2.3.1 Vodopád

Metoda Vodopád představuje sekvenční sled pracovních kroků, které se postupně realizují od počáteční analýzy a sběru požadavků, přes návrh architektury a design, implementaci a testování, až po dodání a údržbu výsledného produktu. Jedná se o lineární přístup, kde nelze v procesu vývoje vrátit se zpět, pouze o jeden krok.[13]

Tato metodika je vhodná pro projekty, kde jsou všechny požadavky známé na začátku a pravděpodobně se nebudou v průběhu vývoje měnit. Přestože je to tradiční a dobře strukturovaný přístup, příliš se nedoporučuje používat pro projekty, které vyžadují flexibilitu a rychlou adaptaci na změny, jako je například náš rozsáhlý projekt administrace, kde se očekává potřeba pružně reagovat na nové požadavky a změny v průběhu vývoje.

2.2.3.2 Iterativní model

Iterativní model představuje postup, kde vývoj probíhá formou opakovaných iterací, přičemž každá iterace je samostatným cyklem, který zahrnuje analýzu, návrh, implementaci, testování a dodání části produktu. Jedná se o sérii ”malých vodopádů”, kde každá iterace produkuje část výsledného produktu a přináší konkrétní funkční výstupy.[13]

Průběžné dodávky v průběhu každé iterace umožňují snížit riziko selhání projektu tím, že postupně vyvíjí a testuje jednotlivé části produktu. Díky tomu je možné získat zpětnou vazbu od zákazníka a upravit směr vývoje dle aktuálních potřeb a požadavků. Tento přístup umožňuje flexibilně reagovat na změny a nejistoty v průběhu projektu, což je zvláště důležité v případě komplexních a rozsáhlých projektů, jako je náš případ s vývojem administrace.

2.3 Současný stav

Současná administrace e-shopu je založena na softwaru nazvaném OpenCart, který interaguje s databází MySQL. Tento software poskytuje uživatelům prostředí pro správu produktů, objednávek, kategorií a dalších funkcionalit souvisejících s provozem e-shopu.

V průběhu let byl OpenCart upravován a přizpůsobován specifickým potřebám a požadavkům. Tato úprava zahrnovala modifikace a rozšíření funkcí, které nebyly standardně dostupné v základní verzi tohoto softwaru.

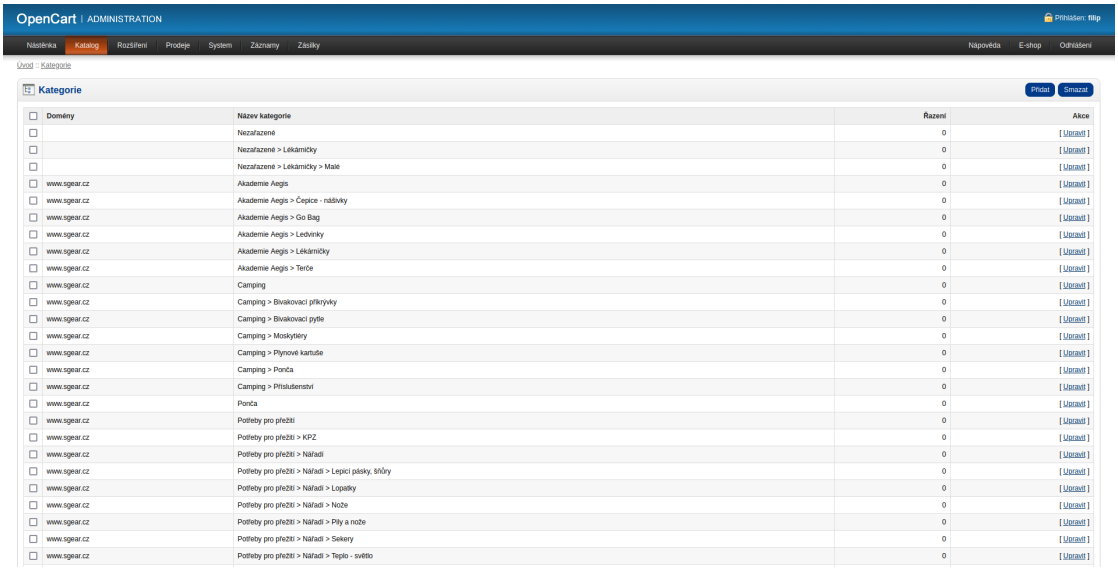
„Opencart je snadno použitelný, výkonný program pro správu online obchodu s otevřeným zdrojovým kódem, který dokáže spravovat více online obchodů z jednoho backendu. K dispozici je mnoho profesionálně napsaných rozšíření pro přizpůsobení obchodu vašim potřebám.“[14]

Mezi výhody používání systému Opencart dle [14] patří:

- Je relativně jednoduchý k instalaci a nasazení, což umožňuje rychlý start s provozem e-shopu.
- Software umožňuje poměrně snadno přizpůsobit vzhled a funkcionality e-shopu podle potřeb konkrétního podnikání.
- Nabízí širokou škálu doplňků a rozšíření, které umožňují rozšířit funkcionalitu e-shopu podle potřeb uživatele.
- Má rozsáhlou komunitu uživatelů a vývojářů, což znamená dostupnost mnoha doplňků, témat a podpory.

Nicméně mezi jeho nevýhody jsou:

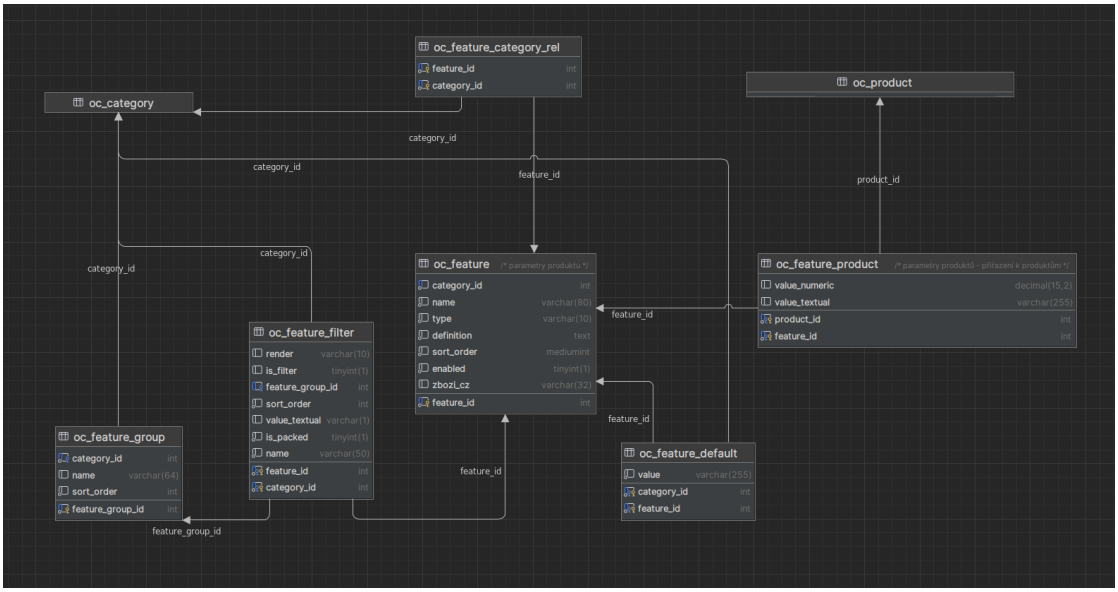
- Interface a design OpenCartu může působit zastarale v porovnání s modernějšími e-commerce platfomami, pro ukázku snímek obrazovky 2.1.
- Přestože OpenCart nabízí určitou míru personalizace, může být omezen v možnostech úprav uživatelského rozhraní a funkcí dle konkrétních požadavků klienta.
- Není dostatečně intuitivní pro pochopení toho, co mají některé hodnoty za následek ve výsledném e-shopu.



Obrázek 2.1 Ukázka tabulky kategorií v administraci OpenCart

2.3.1 Datový model parametrů

Po spuštění projektu na svém lokálním zařízením, jsem si pomocí nástroje na generaci databázových diagramů od IntelliJ[15], dokázal takový diagram vygenerovat a identifikovat v něm tabulky 2.2, které jsou spjaté s parametry. Následně tyto tabulky budu analyzovat zvlášť s pomocí reálných dat, která mi byla poskytnuta a zdrojového kódu backendu v PHP, na kterém operuje současná administrace.



Obrázek 2.2 Databázový diagram tabulek spjatých s parametry

2.3.1.1 Definice parametru

Tabulka **oc_feature** je pro tuto práci nejspíše ta nejdůležitější, zahrnuje totiž kompletní definici samotného parametru. Jeho sloupce popisuji v této tabulce 2.1.

název	typ	popis
category_id	integer	Všiml jsem si, že je tato hodnota napříč systémem nevyužívána. Všude je vyplněna hodnota -1 a kategorie s takovým id neexistuje.
name	string	Popisuje samotný název parametru.
type	string	Určuje typ parametru, který je spjatý s později vyplňovanou hodnotou. Mezi typy patří hodnoty popisované zde 2.2.
definition	string	Textově definuje hodnoty parametru.
sort_order	integer	Číslo udávající pořadí zobrazení ve výsledném detailu produktu.
enabled	bool	Označuje zdali parametr v detailu produktu zobrazovat.
zbozi_cz	string	Řetězec udávající jméno parametru na srovnávací stránce zbozi.cz [16]
feature_id	integer	Primární klíč parametru.

Tabulka 2.1 Tabulka **oc_feature**

hodnota	význam	příklad	filtruje se
bool	Je určený pro parametry, které nabývají hodnoty pravdivé či nepravdivé.	Skleněný (Ano/Ne)	ANO
text	Slouží k podrobnějšímu popisu.	Celý název	NE
number	Je určený pro parametry, které nabývají nějaké číselné hodnoty.	Počet balení (ks)	ANO
file	Mezi tyto parametry patří odkazy na návody k produktům.	Návod k použití	NE
system	Existuje pro odlišení systémově definovaných parametrů.	Cena, skladem, barva...	ANO

Tabulka 2.2 Hodnoty a významy typů parametrů

2.3.1.2 Hodnoty parametru

Proto aby parametr nabýval nějakého významu, je třeba, aby pro něj byla vyplněná nějaká hodnota, tyto hodnoty připadají tabulce **oc_product_feature** 2.3. Hodnoty jsou vázané dvojicí klíčů produktu a parametru, nadále obsahují v závislosti na typu parametru2.2 buď textovou nebo číselnou hodnotu, číselná hodnota se vyskytuje i u booleovského typu, kde je hodnota 1 pro pravdu a 0 pro nepravdu.

název	typ	
value_numeric	integer	Textová hodnota parametru
value_textual	string	Číselná hodnota parametru
{product_id, feature_id}	{integer, integer}	Složený primární klíč

Tabulka 2.3 Tabulka **oc_product_feature**

2.3.1.3 Kategorické parametry

Přestože si můžeme všimnout, že v databázovém diagramu 2.2 existují dvě tabulky – **oc_feature_category** a **oc_feature_category_rel** – které by mohly odpovídat kategorickým parametrům, správná z nich je **oc_feature_category_rel**. Došel jsem tak po zanalyzování modelu kategorického parametru z PHP serveru, na kterým běží aktuální systém. Navíc je **oc_feature_category** vyplněná nesmyslnými hodnotami. Pokud bude možné se této tabulky v budoucnu zbavit, napomohlo by tak dalším členům věnujícím se tomuto projektu. Tabulka **oc_feature_category_rel** tedy funguje akorát jako vazební tabulka mezi kategorií a parametrem.

2.3.1.4 Filtrované parametry

Mezi klíčové tabulky filtrovaných parametrů patří **oc_feature_filter** a **oc_feature_group**, jak je znázorněno na obrázku 2.2. Z těchto tabulek má **oc_feature_filter** větší význam. Většina sloupců v této tabulce ovlivňuje styl zobrazení filtrovaných parametrů na výsledném e-shopu. Detailní popis jednotlivých sloupců lze nalézt v tabulce 2.4. Zajímavým poznatkem je, že primárním klíčem této tabulky je kombinace identifikátoru parametru a kategorie, což umožňuje propojení parametrů s kategoriemi a uchování podrobností o jejich stylu filtrování. Naopak tabulka **oc_feature_group** se specializuje na seskupení více filtrů s podobnými vlastnostmi do jedné skupiny, která je identifikována svým názvem.

název	typ	popis
render	string	Slouží pro výběr stylu zobrazení pro daný filtr, mezi možnosti patří posuvník, checkboxy, Ano/Ne a žádné.
is_filter	bool	Nenašel jsem v systému využití této hodnoty, je možné, že tato hodnota může filtr skrýt.
feature_group_id	integer	Smí odkazovat na skupinu filtrů, do které patří.
sort_order	integer	Řeší pořadí zobrazení filtrů v banneru na finální e-shopové stránce.
value_textual	string	Opět jsem užití pro tuto hodnotu nenašel.
is_packed	bool	Závisí na tom, zdali je filtr nejdříve zabalený. Využívá se pokud je ku příkladu počet checkboxů početný a mohl by tak zabírat velkou část stránky.
name	string	Jméno filtru zobrazované na popředí.
{feature_id, category_id}	{integer, integer}	Složený primární klíč

Tabulka 2.4 Tabulka **oc_feature_filter**

2.3.2 Aktuální nedostatky

Vedoucím práce – Jiřím Hunkou – mi byl přidělen přístup k jedné z databází administrace e-shopu, za účelem anonymního rozboru stavu. Mým úkolem bylo identifikovat nedostatky existující v oblastech tvorby a úpravy kategorických parametrů, filtrace podle parametrů v rámci dané kategorie, tvorby a úpravy produktových parametrů a slučování parametrů. Z této analýzy jsem dosáhl těchto poznatků:

- Při slučování parametrů se uživateli neohlásí chyba, pokud jsou slučovány parametry s rozdílnými typy. Systém tuto funkcionalitu totiž nepodporuje. Slučování parametrů by mohlo být

možné rovnou při vybírání parametru pro produkt i kategorii (nějaký modal). Ve chvíli, kdy si uživatel uvědomí, že vytvořil parametr, který již existuje třeba s obdobným názvem.

- Nad parametry produktů nejdou provádět CRUD operace. Měla by existovat možnost měnit číselnou jednotku i název. Je otázkou, zdali dovolit uživateli změnu typu. Při změně typu by totiž předchozí vyplněná hodnota nedávala smysl a musela by se znovu vyplnit u všech parametrů, který ji obsahovali. Tudíž se bude třeba zamyslet, jakým způsobem by se daly hodnoty parametru převést – třeba podle nějakých kritérií.
- Když si uživatel přeje vytvořit nový parametr, při vytváření nové kategorie, nejsou změny uplatněny hned, a tudíž uživatel nemá jak přidat nově vytvořený parametr do filtrovaných parametrů kategorie. Je tudíž třeba se zamyslet nad tím, zdali parametry vytvářet již ve chvíli, kdy je parametr definován ve formě, nebo přidat částečné ukládání formy.
- Při vytváření nové kategorie, by bylo pěkné znát děděné parametry kategorie, z které nová kategorie vychází.
- V systému chybí hlášení u produktů, které nemají vyplněny všechny své parametry zděděné z kategorie.
- Při vybírání filtrů pro kategorii jsme omezení pouze na parametry, které vycházejí z kategorií. Pokud tedy nějakému produktu nastavíme nějaký nový parametr, který není dosud použit v některé z kategorií, nemáme jak tuto hodnotu vyfiltrovat. Nastává otázka, zdali umožnit filtrování přes všechny existující parametry nebo tuto limitaci více zúžit a to pouze na kategorické parametry dané kategorie.
- Při vyplňování produktových parametrů by mohl systém identifikovat, že se daný parametr vyskytuje již u více produktů ve stejné kategorii a nabídl by tak uživateli možnost přidání tohoto parametru rovnou ke společné kategorii. Navíc chybí tlačítko pro odstranění parametru produktu (nyní se musí vymazat hodnota parametru a posléze uložit).
- Chybí překlady parametrů.
- Systém nedává uživateli vědět k čemu je dobré přidávat parametry ke kategoriím, ve většině případech všechny parametry ukládají pro produkt jako vlastní, tím pádem nenesou kontext při další tvorbě produktu ze stejné kategorie.
- Některé vyplnitelné položky by si zasloužily vysvětlivky, aby měl uživatel přehled o tom, proč danou položku vůbec vyplňuje.
- Jednotky by také zasloužily předělat, uživateli by mohli být nabízeny různé převody jednotek pro snazší zadávání (10^n).
- Při vyhledávání parametru se v nabízených možnostech vyskytuje pouze jméno parametru, pokud jsou tedy jména nějakých dvou parametrů identická, uživatel musí kontrolovat, jestli je to ten správný parametr (jakého je typu, jakou jednotku obsahuje). Nabízí se typ i jednotku zobrazovat rovnou v možnosti se jménem.

Na základě provedené analýzy existující administrace e-shopu bylo identifikováno několik nedostatků v oblastech tvorby a úpravy kategorických a produktových parametrů, stejně jako v oblasti filtrace podle parametrů. Tyto nedostatky zahrnují nedostatečnou podporu slučování parametrů, omezené možnosti úprav nad parametry produktů a nedostatečnou interaktivitu při vytváření nových kategorií a parametrů.

Získané poznatky budou konzultovány se zadavatelem práce – Jiřím Hunkou – a na základě této konzultace budou vytvořeny funkční požadavky pro další fázi vývoje. Je naší prioritou zajistit, aby administrace e-shopu byla plně funkční a efektivní pro uživatele, a těmito funkčními požadavky práci také řídit.

2.4 Požadavky

Tato část se zaměřuje na stanovení požadavků, které vyplynuly z analýzy aktuálního stavu a nedostatků v administraci vzorového e-shopu. Cílem této části je detailní popis potřebných změn a vylepšení, které je třeba provést v systému, aby byl lépe přizpůsoben potřebám uživatelů a splňoval moderní standardy pro správu e-shopu.

Analýza požadavků je klíčovým krokem při vývoji softwaru, který má za cíl definovat rozsah a specifikace systému. Jejím hlavním účelem je vymezit hranice systému, přesně určit funkcionality a procesy, které systém poskytne. Tím umožňuje přesnější odhad pracnosti a nákladů a zajišťuje, že výsledný systém bude plně odpovídat potřebám a očekáváním zákazníka. Současně pomáhá vyjasnit si zadání se zákazníkem a zachytit veškerá omezení, která jsou na systém kladena. Výsledkem této analýzy je dokument obsahující detailní popis požadovaných funkcí, vlastností a omezení, který slouží jako základ pro návrh a implementaci softwarového systému. [17]

Při formulaci požadavků na softwarový systém je dle [17] běžné rozdělovat je do dvou hlavních kategorií:

- **Funkční**
  - Funkční požadavky definují konkrétní funkcionality a procesy, které systém musí provádět.
  - Např. Administrátor musí mít možnost přidávat, upravovat a mazat uživatelské účty.
- **Nefunkční**
  - Nefunkční požadavky se zaměřují na omezení, kvalitu a vlastnosti systému, které nejsou přímo spojeny s jeho funkcionalitou, ale mají zásadní dopad na jeho návrh, provoz a uživatelskou zkušenost.
  - Např. Uživatelské rozhraní musí být intuitivní a snadno použitelné i pro nezkušené uživatele.

FURPS je zkratka používaná v softwarovém inženýrství, která popisuje různé aspekty a požadavky na softwarový systém. Jedná se o akronym z anglického výrazu Functionality, Usability, Reliability, Performance a Supportability. Každá z těchto položek reprezentuje důležitý rys nebo cíl, který by měl být zahrnut do návrhu a vývoje softwarového produktu [17]:

- F (Functionality) – Funkčnost:
  - Jedná se o funkční požadavky, které definují, co systém musí dělat a jaké funkce musí poskytovat uživatelům.
- U (Usability) – Použitelnost:
  - Tato kategorie se zabývá uživatelským rozhraním a zkušeností. Zahrnuje požadavky na přívětivost, intuitivnost a efektivitu uživatelského rozhraní.
- R (Reliability) – Spolehlivost:
  - Spolehlivost systému je klíčovým aspektem. Tato kategorie obsahuje požadavky na stabilitu, dostupnost a odolnost vůči poruchám.
- P (Performance) – Výkon:
  - Výkonnostní požadavky se týkají rychlosti, výkonnosti a efektivity systému. Zahrnují požadavky na rychlost odezvy, zpracování dat a škálovatelnost.
- S (Supportability) – Podporovatelnost/Rozšiřitelnost:

- Tato kategorie se zaměřuje na udržitelnost systému, možnosti aktualizace a rozšíření v budoucnu. Zahrnuje požadavky na dokumentaci, modularitu, snadnou údržbu a rozšiřitelnost systému.

Alternativou k FURPS může být například model Quality Attributes (Kvalitativní atributy), který se soustředí na různé kvalitativní vlastnosti softwarového systému, jako je bezpečnost, škálovatelnost, flexibilita, kompatibilita a další. Existuje mnoho různých metodologií a modelů, které lze použít k definování požadavků a analýze softwarových systémů v závislosti na konkrétních potřebách a cílech projektu.[18]

V této sekci jsou prezentovány konkrétní požadavky na funkcionalitu a uživatelské rozhraní, které vyplývají z identifikovaných nedostatků a požadavků zadavatele práce. Tyto požadavky jsou formulovány s ohledem na optimalizaci procesů tvorby a úpravy kategorických a produktových parametrů, filtrace podle parametrů a další důležité činnosti spojené s administrací e-shopu. Požadavkům bude přidělována priorita od 1 do 5, přičemž 1 je hlavní priorita.

2.4.1 Funkční požadavky

- **FR1 Parametry u kategorií [F]**
  - Systém umožní administrátorům přidat nový kategorický parametr. **Priorita: 1**
  - Administrátoři budou moci přidat existující parametr do určité kategorie. **Priorita: 1**
  - Systém umožní identifikovat vlastní parametry(parametry produktů, které nespadají do jejich kategorie), které jsou používány v určité kategorii a jejich procentuální vyplněnost. **Priorita: 1**
  - Systém bude schopen vypočítat procentuální vyplněnost parametrů v rámci kategorie. **Priorita: 1**
  - Administrátoři budou moci vyplňovat hodnoty parametrů při jejich přidávání nebo editaci a to i pro ostatní produkty v kategorii. **Priorita: 1**
  - Systém umožní řazení parametrů podle určené priority. **Priorita: 4**
  - Systém umožní mazání parametrů, pokud to bude možné. **Priorita: 2**
  - Systém umožní přesun identifikovaného vlastního parametru v kategorii do kategorických parametrů a zpátky. **Priorita: 1**
  - Systém umožní zobrazit parametry dané kategorie společně s informací odkud byla zděděna. **Priorita: 2**
  - Systém umožní přesun kategorického parametru mezi kategoriemi v cestě do kořene a zpátky. **Priorita: 2**
- **FR2 Filtrované parametry kategorie [F]**
  - Administrátoři budou moci vytvářet filtrované parametry pro usnadnění vyhledávání produktů. **Priorita: 1**
  - Systém umožní úpravu filtrovaných parametrů a jejich zařazení do logických skupin. **Priorita: 1**
  - Administrátoři budou moci kopírovat filtry z nadkategorií pro rychlé nastavení. **Priorita: 2**
  - Systém umožní přidání filtrovaných systémových parametrů, které jsou potřebné pro správu e-shopu. **Priorita: 1**
  - Systém umožní mazání filtrovaných parametrů. **Priorita: 1**
  - Systém umožní řazení filtrovaných parametrů. **Priorita: 4**

- **FR3 Parametry u produktů [F]**
  - Systém umožní pro produkt zobrazit parametry zděděné z kategorie. **Priorita: 1**
  - Systém umožní pro produkt zobrazit vlastní parametry. **Priorita: 1**
  - Systém umožní administrátorům přidat nový kategorický parametr. **Priorita: 1**
  - Administrátoři budou moci přidat existující parametr k vlastním, nebo kategorickým parametrům. **Priorita: 1**
  - Administrátoři budou moci vyplňovat hodnoty parametrů při jejich přidávání nebo editaci a to i pro ostatní produkty v kategorii. **Priorita: 1**
  - Systém umožní zobrazit procentuální vyplněnost pro kategorický parametr. **Priorita: 1**
  - Systém umožní zobrazit procentuální vyplněnost vlastního parametru, podle vyplněnosti u ostatních produktů v kategorii. **Priorita: 2**
  - Systém umožní přesun identifikovaného vlastního parametru ke kategorickým parametrům a zpátky. **Priorita: 2**
  - Systém umožní řazení parametrů podle určené priority. **Priorita: 4**
  - Systém umožní mazání parametrů, pokud to bude možné. **Priorita: 3**
- **FR4 Parametry obecně [F]**
  - Systém umožní zobrazení všech parametrů v rámci e-shopu pro snadnou kontrolu a úpravu. **Priorita: 1**
  - Administrátoři budou moci slučovat parametry různých typů do jednoho a popřípadě nastavit kritéria pro změnu hodnot, pokud je to možné. **Priorita: 2**
  - Systém umožní administrátorům upravovat existující parametry a popřípadě nastavit kritéria pro změnu hodnot. **Priorita: 3**
  - Systém umožní překládat názvy parametrů, jejich hodnoty, filtrované parametry a jejich skupiny do podporovaných jazyků. **Priorita: 3**
  - Administrátoři budou moci zobrazit překlady. **Priorita: 3**
  - Administrátoři budou moci upravit překlady. **Priorita: 3**
- **FR5 Aplikace bude důkladně otestována [F]. Priorita: 2**

V první kapitole bakalářské práce jsem detailně zanalyzoval současný stav e-shop administrace u společnosti Jagu s.r.o. Cílem této analýzy bylo poskytnout komplexní pohled na stávající administrativní procesy a získaní klíčových poznatků pro návrh a implementaci nového systému. Nasbírané poznatky budou klíčové pro efektivní a uživatelsky přívětivý vývoj nového systému.

2.4.2 Nefunkční požadavky

- Frontendová aplikace bude psána v JavaScriptu s pomocí frameworku Vue.js [S].
- Backendová aplikace bude psána v C# s pomocí frameworku .NET a AutoMapperu, který nám bude zajišťovat snadné mapování entit [S]. [19]
- Aplikace bude využívat aktuální databázi MySQL [S].
- Aplikace musí být kompatibilní s původním systémem administrace. Je nutné zajistit, aby nově přidané funkce neporušily paralelní běh staré administrace [S].
- Navrhneme aplikaci tak, aby byla rozdělena do jednotlivých vrstev podle návrhového vzoru MVC [S]. [20]
- Aplikace bude vybavena API, které bude postaveno na vhodné architektuře (REST) [U]. [21]
- Rozhraní backendové aplikace bude pečlivě zdokumentováno [U].

Kapitola 3
Návrh

Začátek implementace nové e-shop administrace bude zahájen návrhem wireframů. Tato část práce je klíčová pro demonstrování nových funkcionalit a usnadnění následného vývoje frontendové a backendové části systému. Návrh wireframů bude sloužit jako vizuální reprezentace rozložení stránek a interakcí mezi uživatelem a systémem.

Cíle návrhu wireframů:

- **Demonstrace nových funkcionalit**
  - Wireframy poslouží k představení nových funkcí a vylepšení, které budou integrovány do e-shop administrace.
- **Ujasnění požadavků**
  - Pomocí wireframů bude možné lépe specifikovat a ujasnit požadavky na design a funkcionality systému.
- **Zajištění jednoznačného návrhu**
  - Wireframy budou sloužit jako vodítko pro vývojáře při implementaci frontendu a backendu, což zajistí jednoznačný a konzistentní návrh systému.

Postup návrhu wireframů:

- **Identifikace klíčových stránek a funkcí**
  - Nejdříve budou identifikovány klíčové stránky a funkcionality, které budou zahrnuty do nové e-shop administrace.
- **Návrh rozložení stránek**
  - Pro každou identifikovanou stránku budou vytvořeny wireframy zobrazující rozložení jednotlivých prvků a jejich interakce.
- **Konzultace se zadavatelem**
  - Navržené wireframy budou konzultovány se zadavatelem práce, aby bylo zajištěno, že vyhovují jeho požadavkům a očekáváním.
- **Schválení wireframů**
  - Po konzultaci s zadavatelem budou wireframy formálně schváleny a připraveny pro implementaci.

Návrh wireframů bude představovat první krok v procesu vývoje nové e-shop administrace a poskytne pevný základ pro další práci na frontendu a backendu systému.

Jakožto wireframový nástroj jsem použil Figmu. Figma je výkonný nástroj pro návrh rozhraní, který je oblíbený mezi návrháři a týmy díky jeho možnostem spolupráce a prototypování. Umožňuje uživatelům vytvářet, sdílet a spolupracovat na digitálních návrzích v reálném čase, což výrazně zvyšuje produktivitu týmu a usnadňuje proces navrhování. [22]

Ve Figmě jsem měl za úkol každý týden navrhnout část řešení a posléze ho prodiskutovat s vedoucím práce. Tato praxe mi umožnila postupně iterovat nad návrhem, získávat zpětnou vazbu a neustále zdokonalovat výsledný design.

Následně budu popisovat jednotlivé iterace a myšlenkové pochody, které jsem při vytváření návrhu měl. Každá iterace představuje další vrstvu našeho návrhu, který se postupně rozvíjel a zdokonaloval podle požadavků a zpětné vazby. Tímto postupem jsme zajistili, že konečný design bude co nejvíce přizpůsoben potřebám a očekáváním uživatelů.

3.1 První iterace

První iterace návrhu v aplikaci Figma byla zaměřena především na seznámení se s prostředím a současným stavem týmového projektu. Začal jsem studiem funkcí a možností, které Figma nabízí, abych získal potřebné dovednosti pro tvorbu návrhů. Současně jsem se podrobněji seznámil s existujícím projektem v aplikaci Figma, abych získal přehled o struktuře a fungování systému.

Na základě této přípravy jsem se rozhodl použít aktuální řešení týmového projektu jako výchozí bod pro návrh. Vytvořil jsem náčrt designu, který obsahoval seznam parametrů a karty parametrů u produktů a kategorií, podobně jako v existujícím řešení. Tento krok mi pomohl lépe porozumět stávajícím prvkům a jejich interakcím v rámci aplikace.

Během schůzky s vedoucím práce [23] jsme diskutovali především o nedostatcích současného systému, které jsem dříve identifikoval a popsal zde 2.3.2. Tyto diskuse nám pomohly lépe pochopit požadavky na změny a vylepšení v návrhu aplikace. Na základě těchto poznatků jsme společně definovali první iteraci návrhu, která zahrnovala prioritní úpravy a nové funkce pro zlepšení uživatelského zážitku a funkcionalit systému.

3.2 Druhá iterace

Ve druhé iteraci návrhu jsem se zaměřil především na problém slučování parametrů v aplikaci. Aktuální stav systému neumožňuje uživateli provádět jiné operace než sloučení parametrů stejného typu. Kromě toho nedovoluje uživateli změnit hodnoty při slučování číselných hodnot s odlišnými jednotkami 3.1.

▶ **Příklad 3.1.** Slučování výkonu v kilowattech (kW) a výkonu ve wattech (W).

Během této iterace jsem se zaměřil na návrh nového řešení, které by uživatelům umožnilo flexibilněji pracovat se slučováním parametrů. Navrhl jsem rozšíření funkcionality tak, aby uživatelé mohli sloučit parametry různých typů a zároveň změnit hodnoty parametrů s odlišnými jednotkami.

3.2.1 Slučování parametrů

Při analýze možností slučování parametrů jsem zvažoval několik přístupů, které by mohly zlepšit uživatelskou zkušenost a efektivitu práce s parametry. Zde jsou hlavní zjištění a návrhy:

3.2.1.1 Unifikace jednotek

Nápad unifikace jednotek na základní jednotky SI přinesl několik zajímavých úvah o možnostech zlepšení práce s parametry v systému. Myšlenka byla prostá: uživatel by si mohl vybrat jednotku SI [24], ve které by chtěl zapsat hodnotu parametru, a následně by mohl vybrat exponent, který by určil, v jaké mocnině se má tato hodnota vyjádřit. Tímto by se všechny hodnoty překonvertovaly na základní jednotky, což by umožnilo snadnější porovnávání a sloučení parametrů napříč různými kategoriemi.

Avšak při analýze aktuálního stavu databáze jsem si uvědomil, že tato myšlenka má své limity a technické obtíže. Konverze existujících jednotek na základní jednotky SI by byla nesmírně složitá, zejména kvůli existenci různých zápisů jednotek napříč různými systémy. Pro efektivní konverzi by bylo nutné stanovit jednotnou konvenci zápisu jednotek, což by vyžadovalo obrovské úsilí a mohlo by být problematické zpětně aplikovat na existující data v databázi.

Zároveň by byla zpětná kompatibilita s existujícími daty velmi komplexní a náročná. Tyto technické a organizační výzvy nakonec vedly k rozhodnutí zavrhnout tento nápad a hledat jednodušší a praktičtější alternativy pro zlepšení práce s parametry v systému.

3.2.1.2 Slučování mezi parametrickými typy

- **Slučování číselných parametrů**
  - Při slučování číselných parametrů je proces relativně jednoduchý, zejména pokud se jedná o parametry stejného typu s podobnými jednotkami. Zde je zjednodušený postup:

    1. **Výběr parametrů k sloučení:** Uživatel vybere číselné parametry, které chce sloučit. Tyto parametry by měly mít obvykle stejný typ a podobné jednotky 3.2.

       ▶ **Příklad 3.2.** Výkon ve wattech, kilowattech a megawattech.
    2. **Nastavení výsledného parametru:** Uživatel zadá název pro výsledný sloučený parametr a vybere jednotku, ve které chce výsledek zobrazovat. Tato jednotka by měla být vhodně zvolena pro celkový výsledek slučování.
    3. **Zadání hodnot pro každý parametr:** Pro každý vybraný parametr uživatel zadá číslo, které má být použito k vynásobení všech hodnot daného parametru. Tímto způsobem může uživatel upravit hodnoty parametrů tak, aby odpovídaly požadovanému výsledku slučování.
    4. **Provedení slučování:** Po zadání všech hodnot a nastavení výsledného parametru systém provede slučování a vytvoří nový parametr s odpovídajícími hodnotami.

    Tento jednoduchý proces umožňuje uživatelům flexibilně sloučit číselné parametry podle svých potřeb a představ. Na základě tohoto popisu jsem vypracoval ukázku formy 3.1. Ve formě můžete najít pole pro výsledný název, pole pro výslednou definici jednotky, seznam parametrů s příslušným typem společně s polem udávajícím hodnotu násobitele, tlačítko pro návrat a tlačítko k uložení.

- **Slučování textových a souborových parametrů**
  - Slučování textových a souborových parametrů je relativně jednoduché a probíhá podle následujícího postupu:

    1. **Výběr parametrů k sloučení:** Uživatel vybere textové nebo souborové parametry, které chce sloučit do jednoho parametru 3.3.

       ▶ **Příklad 3.3.** Tyto parametry mohou být různé popisy produktů, cesty k návodům nebo jiné textové informace.
    2. **Nastavení názvu výsledného parametru:** Uživatel zvolí název, pod kterým budou sloučeny ostatní parametry. Tento název by měl co nejvíce vystihovat obsah a význam slučovaných parametrů.

3. **Provedení slučování:** Po vybrání názvu výsledného parametru systém provede slučování a vytvoří nový parametr s vybraným názvem, do kterého jsou zahrnuty všechny vybrané textové parametry.

Tento proces umožňuje uživatelům snadno a rychle sloučit textové parametry pod společným názvem, což může usnadnit organizaci a správu informací v systému. Na základě tohoto popisu jsem vypracoval ukázku formy 3.5. Ve formě můžete najít pole pro výsledný název, seznam parametrů s příslušným typem, tlačítko pro návrat a tlačítko k uložení.

Slučování booleovských parametrů

- Slučování booleovských parametrů má akorát jeden problém, který nastane v případě, kdy se uživatel začne pokoušet sjednotit parametry, které si protiřečí 3.4.

  ▶ **Příklad 3.4.** „Obsahuje displej“ a „Neobsahuje displej“

  Proto je třeba vybrat jeden do, kterého se budou muset sloučit a hodnoty zbylých parametrů znegovat. Proces slučování by probíhal následovně:

  1. **Výběr parametrů k sloučení:** Uživatel vybere booleovské parametry, které chce sloučit do jednoho parametru. Tyto parametry mohou být například různé popisy vlastností produktů.
  2. **Nastavení názvu výsledného parametru:** Uživatel zvolí název, pod kterým budou sloučeny ostatní parametry. Tento název by měl co nejvíce vystihovat obsah a význam slučovaných parametrů.
  3. **Zadání negovaných hodnot:** Uživatel zvolí parametry, pro jejichž hodnoty se uplatní negace.
  4. **Provedení slučování:** Po zadání všech hodnot a nastavení výsledného parametru systém provede slučování a vytvoří nový parametr s odpovídajícími hodnotami.

  Tento jednoduchý proces umožňuje uživatelům flexibilně sloučit booleovské parametry podle svých potřeb a představ. Na základě tohoto popisu jsem vypracoval ukázku formy 3.2. Ve formě můžete najít seznam parametrů s příslušným typem a checkboxem rozhodujícím o negaci parametrické hodnoty, tlačítko pro návrat a tlačítko k uložení.

Slučování booleovských parametrů s číselnými parametry

- Existuje akorát pár případů, kdy uživatel potřebuje sjednotit spolu booleovský parametr s číselným, většinou se booleovské parametry týkají vlastnosti, kterou produkty buď mají, nebo nemají. Jediná možnost kdy by dávalo smysl slučovat číselný parametr s booleovským, by nastávala v případě, kdyby booleovský parametr vypovídal informaci o nějaké číselné hodnotě. V tomto případě bychom mohli sloučit číselné parametry do booleovského parametru podle vypovídající hodnoty 3.5.

  ▶ **Příklad 3.5.** Booleovský parametr: „Obsahuje 5 displejů“ a číselný parametr „Počet displejů“.

  Pro tento případ musíme uvažovat pouze jeden booleovský parametr a mnoho číselných, protože logicky nejde převádět větší počet booleovských parametrů, kvůli potencionálním konfliktům na intervalu hodnot. Uvažoval jsem i potencionální sloučení booleovského parametru na číselný, v tomto případě ale není vždy možné poskytnout booleovskému parametru hodnotu, která by odpovídala reálné vlastnosti produktu, buď z důvodu nepravdy, nebo kvůli vlastnosti 3.6.

  ▶ **Příklad 3.6.** „Doba účinku aspoň 5 minut“, v tomto případě by musel uživatel vybrat specifickou hodnotu po sloučení, která nutně nemusí odpovídat realitě.

  Ve zbytku případů, kdy booleovské parametry udávají přesnou hodnotu, je možné tento parametr převést na jeden číselný parametr.

- **Sloučení číselných parametrů na booleovský**
  1. **Výběr parametrů k sloučení:** Uživatel vybere booleovský parametr společně s číselnými parametry, který na sebe logicky navazují.
  2. **Nastavení názvu výsledného parametru:** Uživatel zvolí název, pod kterým budou sloučeny ostatní parametry. Tento název by měl co nejvíce vystihovat obsah a význam slučovaných parametrů.
  3. **Zadání porovnávaných hodnot:** Uživatel pro číselné parametry zvolí typ porovnání (Př. menší než, vetší než, rovná se…) společně s hodnotou, která pro číselné hodnoty rozhodne pravdivost.
  4. **Provedení slučování:** Po zadání všech hodnot a nastavení výsledného parametru systém provede slučování a vytvoří nový parametr s odpovídajícími hodnotami.

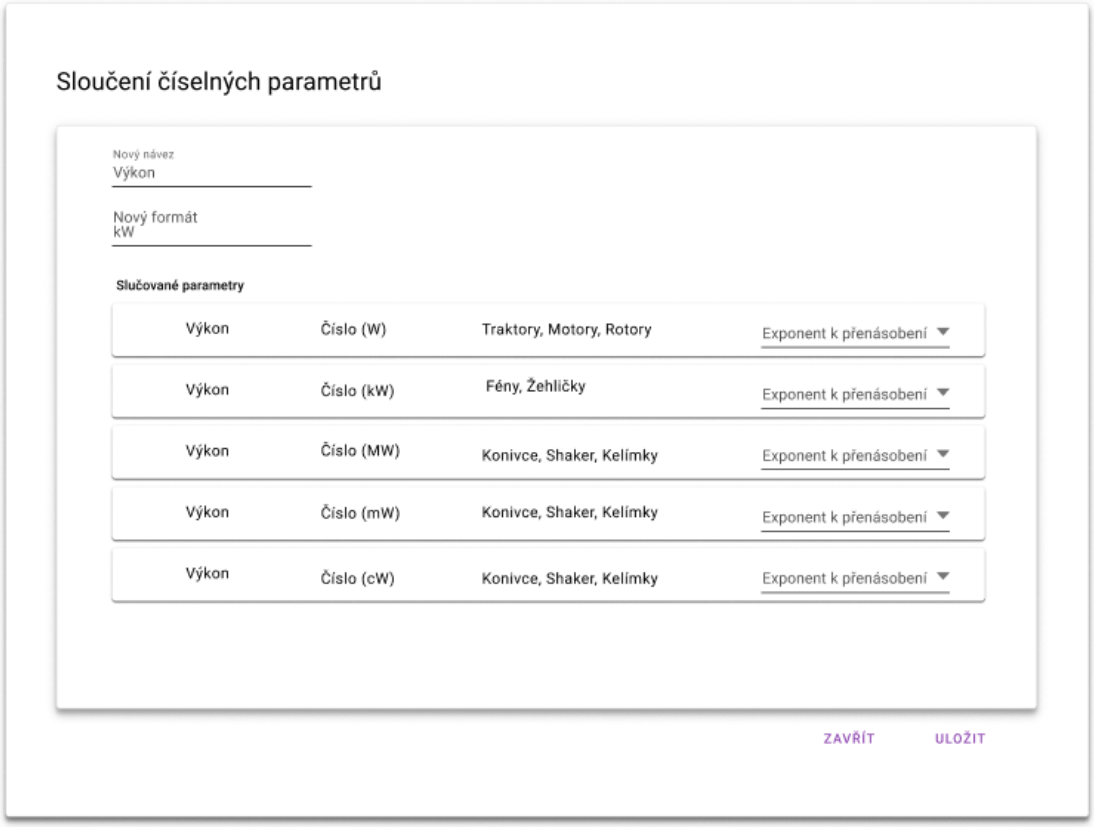
  Z tohoto popisu jsem navrhl formu pro sloučení číselných parametrů na booleovský 3.3. Ve formě se vyskytuje booleovský parametr, na který se bude slučovat, seznam číselných parametrů společně s polem udávajícím typ porovnání, polem hodnoty a případným exponentem. Formulář obsahuje také tlačítko pro návrat a tlačítko k uložení.
- **Sloučení booleovských parametrů na číselné**
  1. **Výběr parametrů k sloučení:** Uživatel vybere číselný parametr společně s booleovskými parametry, u kterých je možné specifikovat přesnou číselnou hodnotu.
  2. **Nastavení názvu výsledného parametru:** Uživatel zvolí název, pod kterým budou sloučeny ostatní parametry. Tento název by měl co nejvíce vystihovat obsah a význam slučovaných parametrů.
  3. **Zadání reálných hodnot:** Uživatel pro booleovské parametry zadá přesné číselné hodnoty, na které se překonvertují.
  4. **Provedení slučování:** Po zadání všech hodnot a nastavení výsledného parametru systém provede slučování a vytvoří nový parametr s odpovídajícími hodnotami.

  Z tohoto popisu jsem navrhl formu pro sloučení booleovských parametrů na číselné 3.4. Ve formě se vyskytuje číselný parametr, na který se bude slučovat, seznam booleovských parametrů společně s hodnotou, která bude v případě pravdivosti aplikována. Formulář obsahuje také tlačítko pro návrat a tlačítko k uložení.

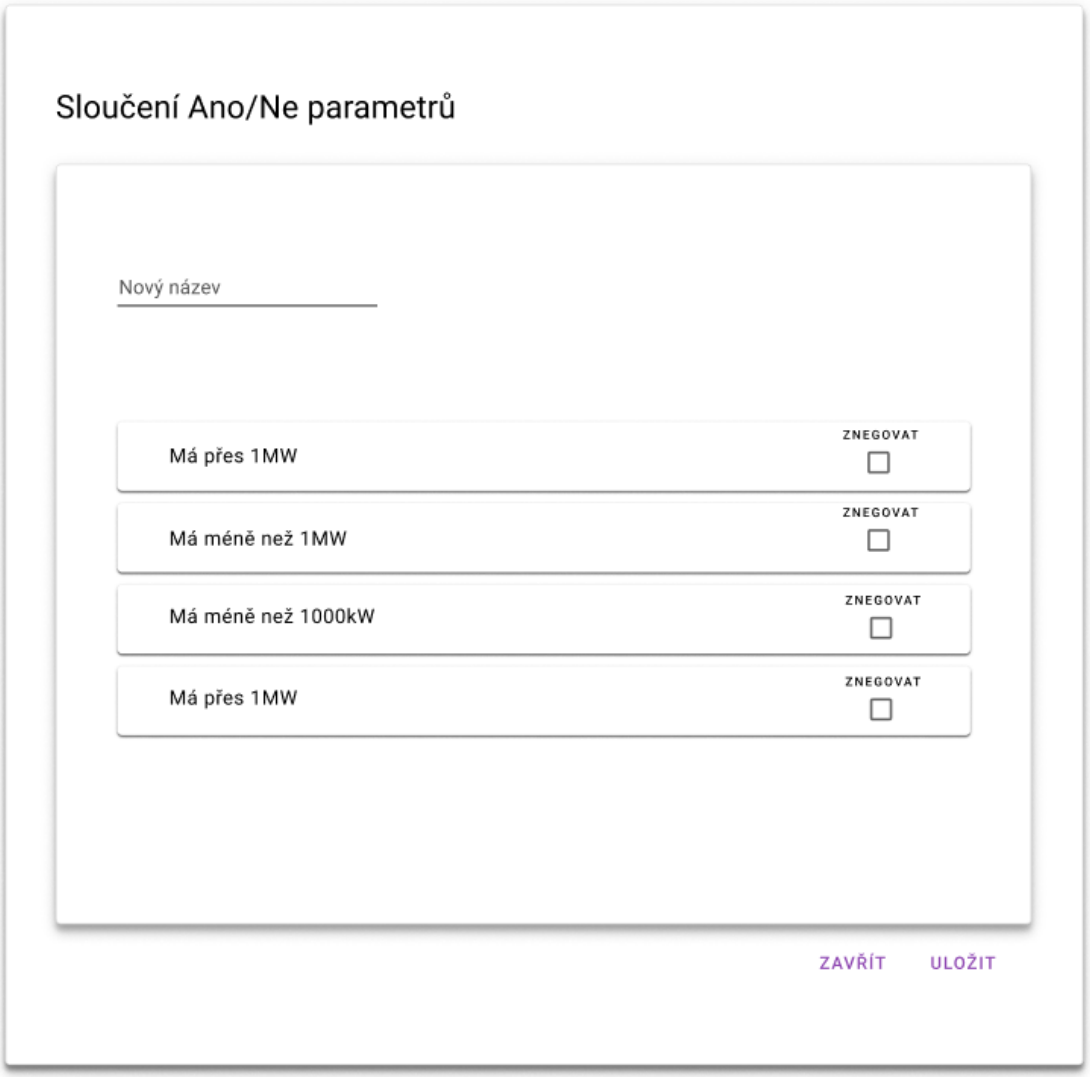
  Tento jednoduchý proces umožňuje uživatelům flexibilně slučovat booleovské parametry společně s číselnými parametry podle svých potřeb a představ.

- **Slučování textových parametrů s různými typy**
  - Sloučení textových parametrů na jakýkoliv jiný typ je velmi komplexní záležitost. Textové parametry se totiž většinou akorát rozepisují o dané vlastnosti produktu a je složité z nich vytěžit nějakou z informací, která by byla užitečná pro konverzi parametru. Jediná možnost by byla v textu kontrolovat výskyty číselných hodnot, jenomže na základě výskytu číselné hodnoty nelze definitivně rozhodnout o reálné vlastnosti produktu. Tímto zamítám tento typ sloučení.

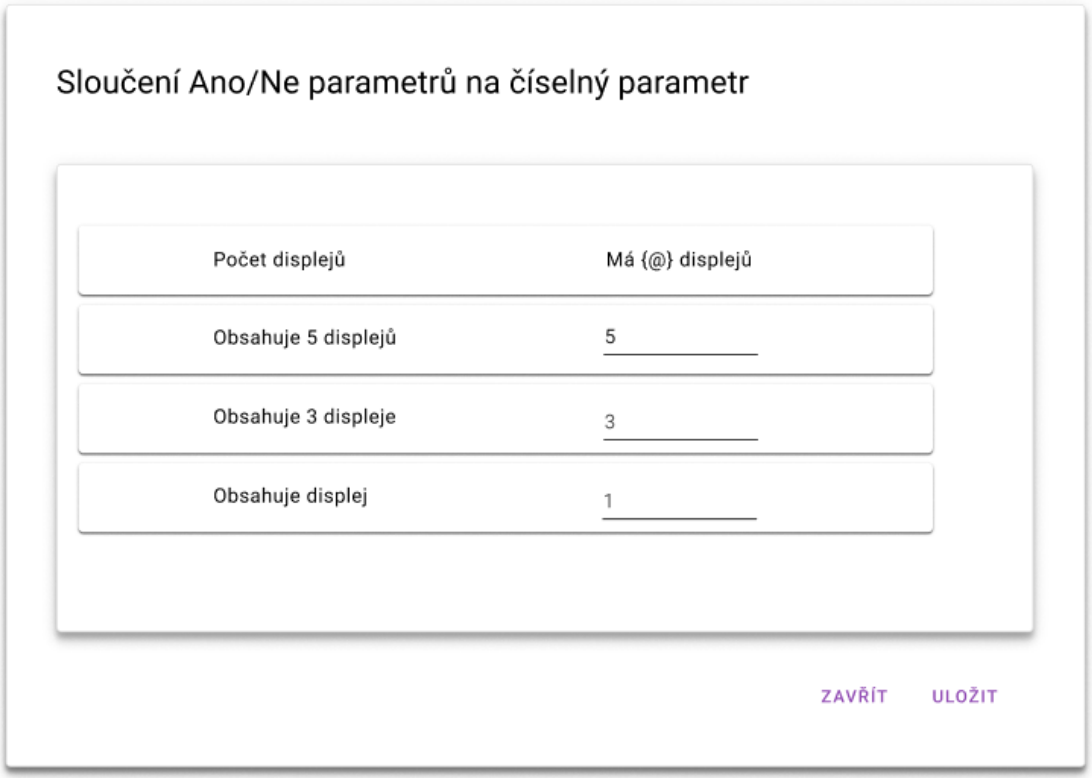
Tento krok by měl výrazně zlepšit uživatelský zážitek a umožnit uživatelům efektivněji spravovat a organizovat parametry v systému. Po dokončení návrhu jsem konzultoval navrhovaná vylepšení s vedoucím práce [23], abychom zajistili, že nové řešení bude splňovat požadavky a očekávání uživatelů.



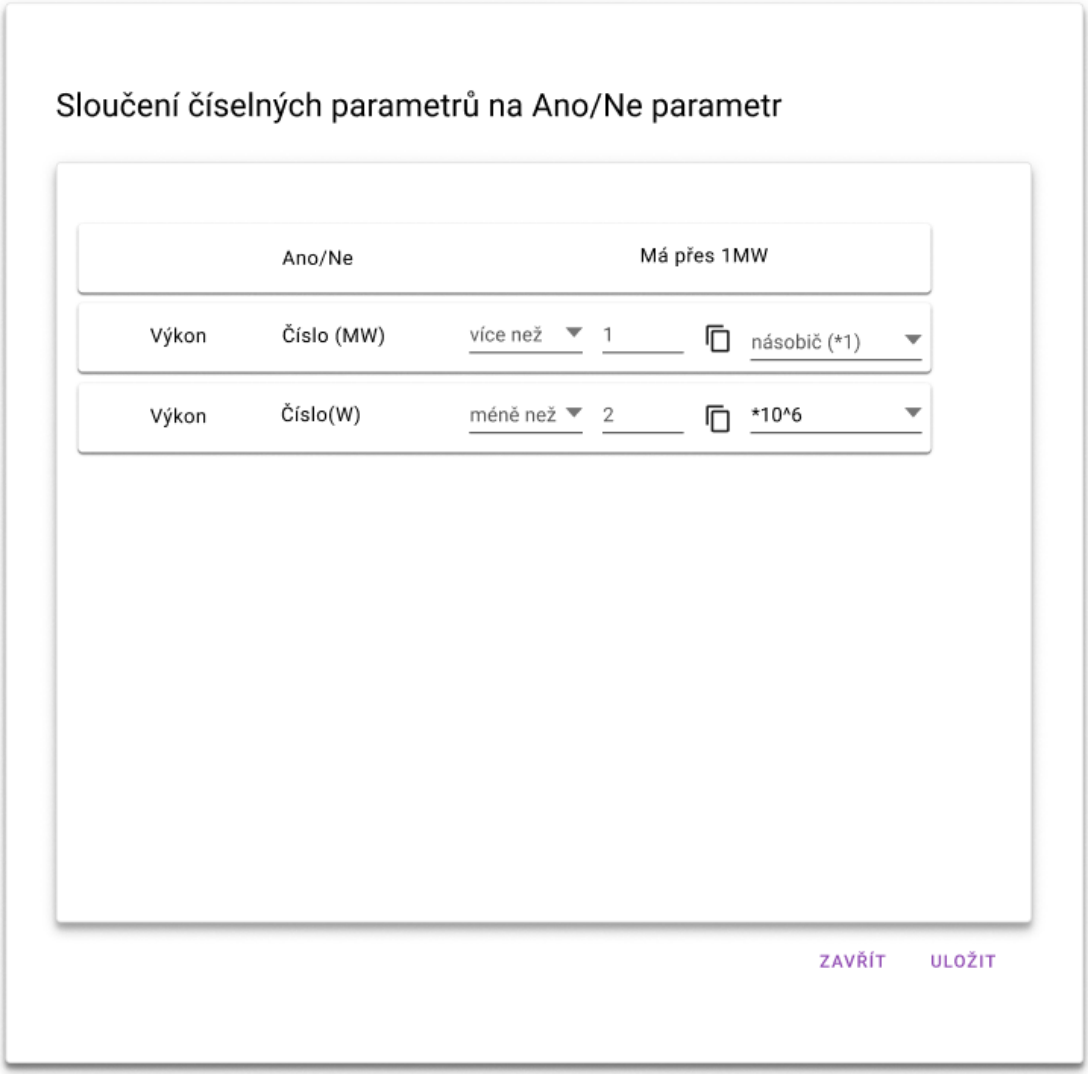
Obrázek 3.1 Návrh rozhraní formuláře sloučení číselných parametrů



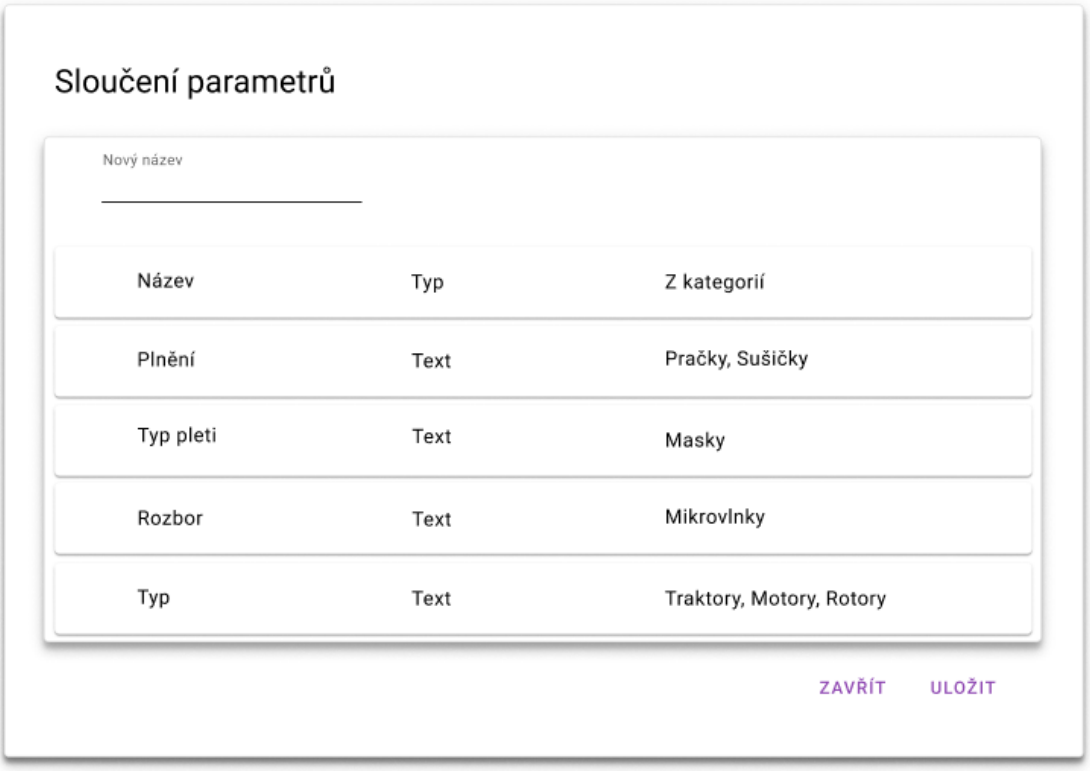
Obrázek 3.2 Návrh rozhraní formuláře sloučení booleovských parametrů



Obrázek 3.3 Návrh rozhraní formuláře sloučení číselných parametrů na booleovský



Obrázek 3.4 Návrh rozhraní formuláře sloučení booleovských parametrů na číselný



Obrázek 3.5 Návrh rozhraní formuláře sloučení pro ostatní parametry

3.3 Třetí iterace

V třetí iteraci návrhu jsem se zaměřil na klíčové aspekty související s překlady parametrů a na tvorbu tabulky pro parametry, která zároveň plní funkci výběru parametrů k slučování. Při této fázi vývoje jsem se zaměřil na dva hlavní body, které přinášejí zásadní přínosy pro celkovou funkcionalitu systému a uživatelskou přívětivost.

Prvním krokem v této fázi je navržení způsobu, jakým budou parametry překládány v systému. Tento proces zahrnuje identifikaci parametrů, které je možné přeložit, a definování mechanismu pro jejich překlad v rámci uživatelského rozhraní. Důraz je kladen na uživatelsky přívětivý a intuitivní způsob práce s překlady, který zajišťuje snadnou a efektivní správu parametrů v různých jazycích.

Druhým klíčovým bodem je návrh tabulky pro zobrazení parametrů. Tato tabulka slouží jako prostředek vizuálního zobrazení parametrů a zároveň umožňuje uživatelům snadno vybírat parametry k jejich slučování. Navrhovaná tabulka musí splňovat požadavky na přehlednost, flexibilitu a možnost detailního filtrování a řazení parametrů podle různých kritérií.

Kromě samotného zobrazení parametrů je důležité navrhnout funkcionality, které by měla tato tabulka splňovat. Mezi tyto funkcionality může patřit možnost přidávat, upravovat nebo mazat parametry přímo z tabulky, rychlé filtrování a vyhledávání parametrů.

Cílem této fáze návrhu je tedy vytvořit komplexní a uživatelsky přívětivé prostředí pro práci s parametry v systému. Implementace navržených prvků bude přispívat k lepší přístupnosti a efektivitě práce s parametry, což má pozitivní vliv na celkovou uživatelskou zkušenost.

3.3.1 Překlady parametrů

Tato část práce se zaměřuje na identifikaci sloupců v databázi, které budou podléhat překladu, a následně na navržení úložiště pro tyto překlady. Na základě analýzy datového modelu 2.2 a funkcionality systému jsme identifikovali následující sloupce, které budou překládány:

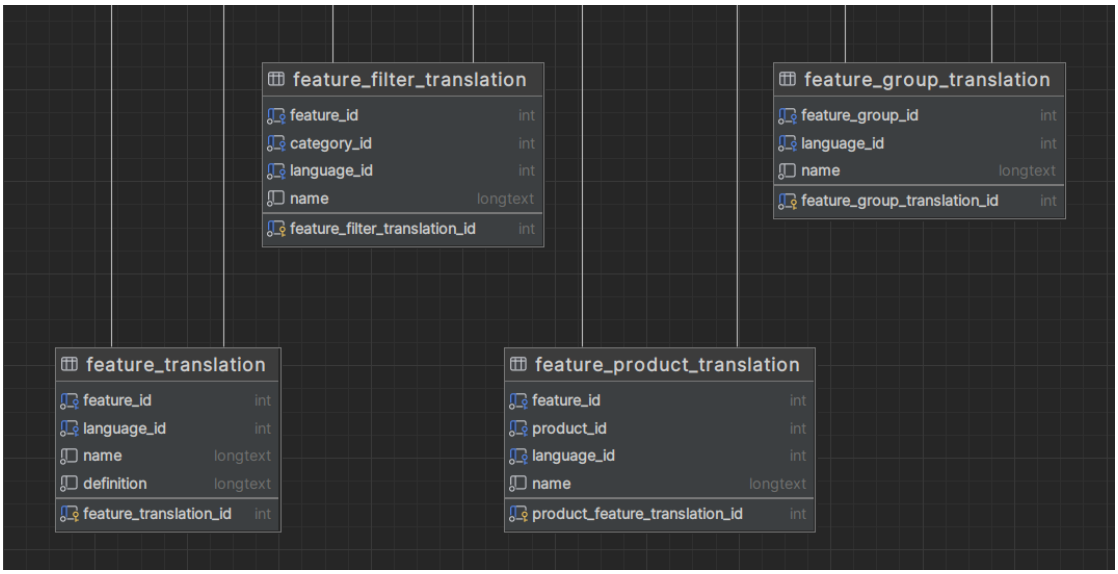
- **Z tabulky `oc_feature` 2.1:**
  - `name`: Tento sloupec obsahuje názvy parametrů, které jsou zobrazovány u produktů a potřebují být přeloženy do různých jazyků.
  - `definition`: Sloupec `definition` obsahuje definici pro číselné parametry. Tyto popisy by měly být také přeloženy pro každý jazyk.
- **Z tabulky `oc_feature_product` 2.3:**
  - `textual_value`: Sloupec `textual_value` obsahuje textové hodnoty parametrů pro jednotlivé produkty. Tyto hodnoty by měly být přeloženy do jazyků, aby byly správně zobrazeny u produktů v různých jazycích.
- **Z tabulky `oc_feature_filter` a `oc_feature_group`:**
  - `name` sloupec z tabulky `oc_feature_filter` 2.4 obsahuje názvy filtrů, které slouží k filtrování produktů na základě různých parametrů.
  - `name` sloupec z tabulky `oc_feature_group` obsahuje názvy skupin filtrů, které mohou obsahovat několik filtrů souvisejících s určitými parametry.

Nyní je třeba zvážit různé možnosti pro úložiště překladů. Můžeme uvažovat o vytvoření samostatné tabulky pro překlady parametrů, která bude propojená s odpovídajícími tabulkami obsahujícími původní hodnoty. Další možností je tvorba sloupců mapovaného typu na základě standartu ISO 639.

ISO 639 je standardizovaný systém pro zobrazení jazyků a jazykových skupin prostřednictvím přidělení jedinečných kódů každému jazyku. Tyto kódy jsou obvykle dvou nebo třípísmenné a jsou

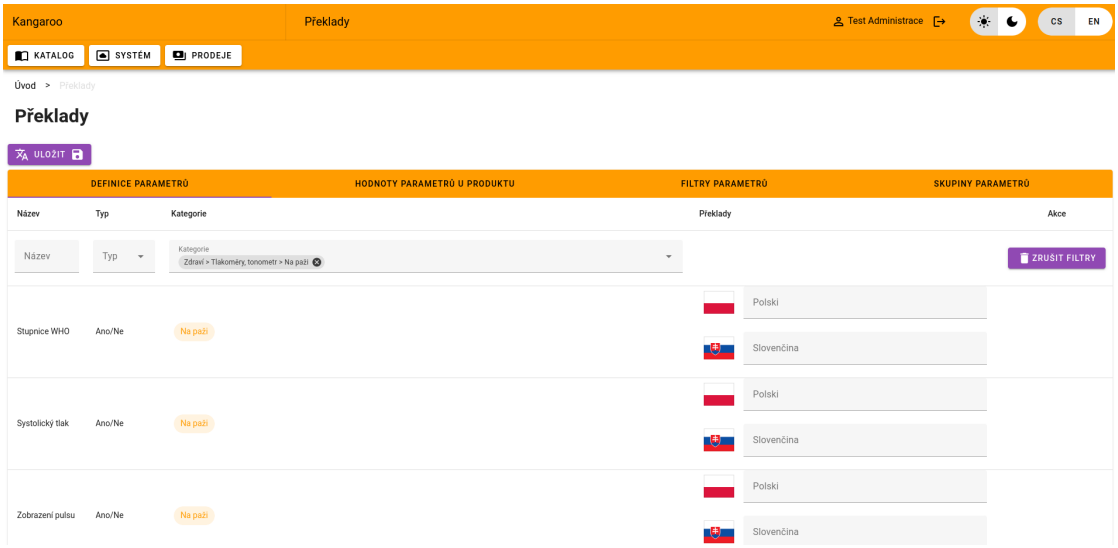
používány mezinárodně k identifikaci jazyků. Standard ISO 639 spravuje Mezinárodní organizace pro normalizaci (ISO). [25]

Na základě konzultace s Janem Matouškem [26], jsem dosáhl závěru, že nejlepší možností bude vytvořit pro každou ze zmiňovaných tabulek 3.3.1 novou překladovou tabulku, která bude obsahovat cizí klíč primárního klíče původní tabulky a cizí klíč `language_id`, který pochází z tabulky `oc_languge`. Tabulka `oc_languge` v systému již existuje a zapisují se do ní nově podporované jazyky. Dále bylo otázkou, zdali by tyto cizí klíče neměli tvořit složený primární klíč, jelikož by měl pro tyto hodnoty existovat akorát jeden záznam. Ale když pomyslíme na slučovaní parametrů, musíme počítat s tím, že při slučování by bylo třeba tyto záznamy převázat na správný `feature_id` a jelikož systémová databáze nepodporuje změnu primárního klíče, museli bychom staré tabulky odstranit a vytvořit nové tabulky se správným klíčem. Tudíž jsem se rozhodl, že tabulky budou obsahovat vlastní generovaný primární klíč za účelu lehčí práce při slučování parametrů. Výsledné tabulky budou vypadat následovně 3.6.



Obrázek 3.6 Modelový návrh tabulek pro překlady

Pro tyto tabulky bude vytvořena samostatná stránka zabývající se překlady, bude tak jednoduché upravit specifický parametr namísto prohledávání celého systému. Na stránce bude pro každou tabulku jedna záložka. Každá tabulka bude navržena tak, aby uživateli poskytla co nejvíce informací k překládanému kontextu a umožnila mu možnost filtrace přes tyto prvky. Navrhl jsem tabulku pro překlad `oc_feature` 2.1, která se pomocí dostupných sloupců dá vyfiltrovat přes název, typ a obsažené kategorie 3.7.



Obrázek 3.7 Navržené rozhraní tabulek pro překlady

3.3.2 Tabulka parametrů

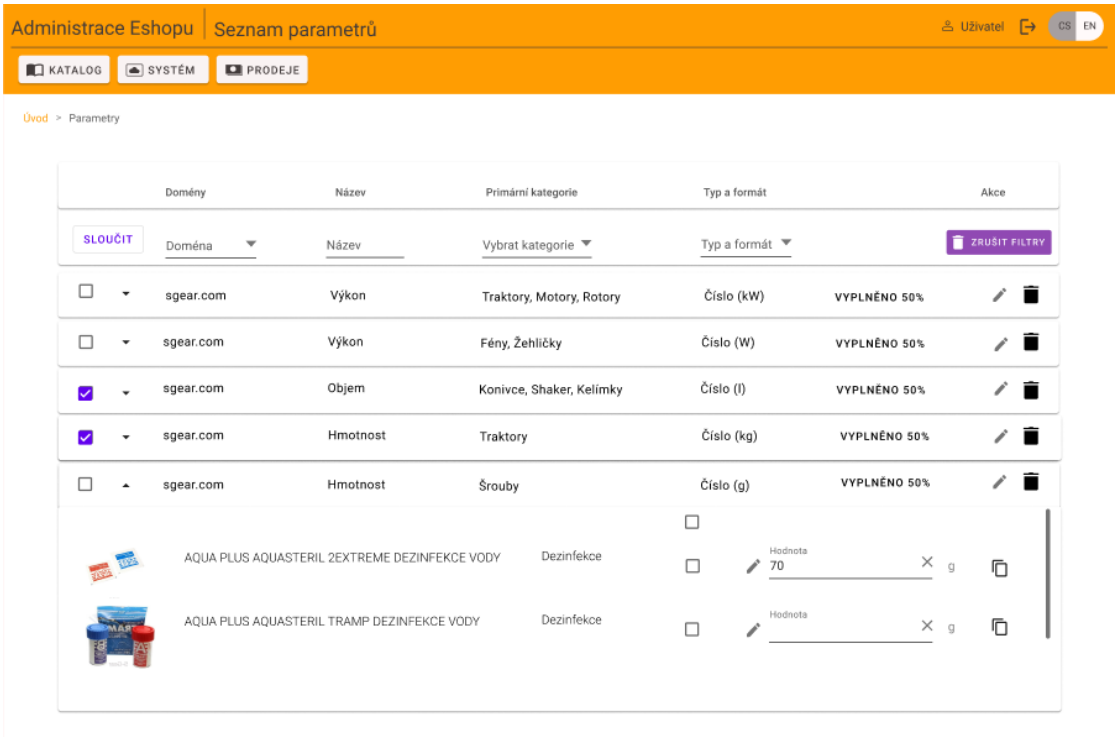
Pro úspěšný návrh této tabulky musíme nejprve identifikovat, jaké sloupce tabulka dle `oc_feature` 2.1 a vytvořených požadavků 2.4.1 bude muset obsahovat:

- **Výběr**
  - Sloupec výběru bude nutný k výběru parametrů, které si uživatel bude přát sloučit. Sloučení bude povolené pouze podle kritérií popsaných v 3.2.1.2.
- **Domény**
  - Jelikož parametr může spadat pod některou z kategorií, tak kategorie může následně spadat pod nějakou z registrovaných domén. Tímto si uživatel bude moct být jistý, že vyplňuje parametr pro správnou doménu.
- **Název**
- **Primární kategorie**
  - Pokud je parametr kategorický, můžeme dát uživateli vědět do kterých kategorií spadají.
- **Typ a formát**
  - Každý parametr obsahuje jeden z typů 2.2. Navíc pro číselné parametry můžeme zobrazit formát zápisu pro jednotku.
- **Vyplněnost**
  - Díky tomu, že parametr spadá pod nějakou z kategorií, můžeme vypočíst počet vyplněných parametrických hodnot oproti celkovému počtu produktů v kategorii.
- **Akce**
  - Podle požadavků na seznam parametrů 2.4.1. Mezi akce bude patřit úprava a mazání parametru. Smazání pro bezpečnost bude možné pouze tehdy, kdy parametr nebude mít vyplněnou žádnou hodnotu. Pro akce budou poskytnuty intuitivní ikonky, a to tužka pro úpravu a koš pro mazání.

Dalším důležitým požadavkem je uživateli dát možnost vyplnění parametrických hodnot pro produkty, u kterých jsou očekávané – to jsou ty produkty, které jsou v kategorii s kategorickým parametrem. Aby uživatel mohl efektivně vyplňovat hodnoty parametrů, je důležité aby měl přehled o tom, pro jaký produkt hodnotu vyplňuje. Tudíž je třeba poskytnout obrázek a název produktu. Pro efektivnost se navíc bude hodit implementovat funkce, které zpříjemní vyplňování. Většinou uživatele může trápit to, že pro nějakou množinu produktů potřebuje vyplnit stejnou hodnotu. Tudíž by se hodilo pro každou hodnotu ve formuláři vytvořit checkbox, který po zvolení bude při editaci pole hodnoty automaticky kopírovat jeho hodnotu do všech zvolených. Dále se může hodit pole obdařit tlačítkem kopie, jenž automaticky překopíruje aktuální hodnotu do zvolených.

Na základě tohoto popisu jsem sestrojil návrh rozhraní 3.8, tabulku jsem obdařil filtračními poli a tlačítkem sloučení.

V této iteraci se mi povedlo, kompletně navrhnout tabulky pro překlady a tabulku pro parametry, splňující mnou stanovené funkční požadavky 2.4.1.



Obrázek 3.8 Navržené rozhraní tabulky pro parametry

3.4 Čtvrtá iterace

Ve čtvrté iteraci mého návrhu jsem se rozhodl zaměřit na formulář úpravy a tvorby parametrů u produktu a také na formulář úpravy a tvorby parametrů u kategorie. Tyto prvky jsou klíčové pro správnou konfiguraci parametrů produktů a kategorií v administraci e-shopu. Cílem této fáze návrhu je vytvořit uživatelsky přívětivé prostředí, které umožní uživatelům snadno spravovat a vyplňovat parametry jednotlivých produktů a kategorií.

V rámci této iterace budu zkoumat různé možnosti, jak efektivně zobrazit a editovat parametry v administrativním rozhraní e-shopu. Bude zohledněna uživatelská přívětivost a přehlednost, aby bylo co nejjednodušší pracovat s parametry a provádět potřebné úpravy.

Cílem této fáze návrhu je optimalizovat proces práce s parametry v administrativním rozhraní e-shopu a zlepšit uživatelskou zkušenost při správě produktů a kategorií.

Jelikož z mé strany se formuláře pro úpravu a tvorbu liší akorát tím, že jedna zahrnuje identifikátor pro produkt či kategorii a druhá ne, budou formuláře úpravy a tvorby zkoumány jako celek.

3.4.1 Správa parametrů v kategorii

Při návrhu správy parametrů v kategorii se zaměřím na optimalizaci uživatelského rozhraní tak, aby uživatelé měli snadný přístup ke všem dostupným parametrům a zároveň byli schopni správně nastavit parametry pro danou kategorii produktů. Tato sekce bude rozdělena do dvou částí, které budou poskytovat uživatelům různé typy parametrů.

3.4.1.1 Kategorické parametry

První část bude poskytovat uživatelům kategorické parametry. Těmito parametry jsou ty, které jsou registrované v tabulce `oc_feature_category_rel` 2.3.1.3. Uživatelé budou mít možnost prohlížet a spravovat tyto parametry. Mezi funkcemi spadající pod správu budou:

- **Tvorba nového parametru**
  - Uživatel bude schopný pomocí polí názvu, typu a popřípadě jednotky vytvořit nový parametr.
- **Přidání existujícího parametru**
  - Uživateli budou při zadávání názvu nabízeny již existující parametry.
- **Přesun**
  - Tabulka s parametry bude interaktivní a uživatelé budou moci přetažením měnit pořadí parametrů. Tato funkce umožní uživatelům upravit pořadí parametrů, což se projeví při zobrazení produktů ve výsledném e-shopu.
- **Úprava hodnot**
  - Uživatelé budou mít možnost rozložit parametr a podobně jako u tabulky parametrů 3.3.2 budou moci snadno upravovat hodnoty parametrů u jednotlivých produktů. Tímto způsobem mohou snadno dynamicky upravovat hodnoty parametrů.
- **Změna původu kategorického parametru**
  - Pokud uživatel při tvorbě parametru zjistí, že by měl být parametr vytvořen spíše pro nadkategorii, umožníme mu změnit původ kategorického parametru 3.7.

▶ **Příklad 3.7.** Pokud se uživatel nachází v kategorii „Stolní počítače“ a chce přidat parametr „Frekvence“, může si uvědomit, že tento parametr by měl spadat do nadřazené kategorie „Počítače“.

Tímto způsobem mohou uživatelé lépe strukturovat své kategorie a parametry a zajistit konzistenci dat.

- **Přesun do úpravy nadkategorií**
  - Pokud kategorický parametr pochází původně z některé nadkategorie, uživateli bude k ní zpřístupněn odkaz.
- **Mazání**
  - Pokud žádná z parametrických hodnot u parametru není vyplněna, uživatelé budou mít možnost smazat kategorický parametr. Tato funkce umožní uživatelům snadno odstranit nepotřebné parametry a udržovat systém přehledný.

3.4.1.2 Vlastní parametry

Vlastní parametry představují hodnoty specifické pro jednotlivé produkty, které nejsou sdíleny mezi různými produkty a nejsou definovány jako kategorické parametry. V této části formuláře budou uživatelům zobrazeny všechny tyto vlastní parametry, které jsou již vyplněny pro produkty. Tímto uživatelům poskytneme přehled o všech parametrech, které jsou aktuálně přiřazeny k produktům, ale nejsou součástí kategorických parametrů.

Uživatelé budou mít možnost v této části formuláře provádět následující akce:

- **Zobrazení parametrů**
  - Všechny vlastní parametry budou zobrazeny v tabulce, která uživatelům umožní prohlížet jednotlivé parametry a jejich hodnoty.
- **Přesun do kategorických parametrů**
  - Uživatelé budou mít možnost provést přesun vybraného vlastního parametru do kategorických parametrů. Tato akce umožní začlenit vlastní parametr mezi standardní kategorické parametry, což přispěje k lepší organizaci a strukturování parametrů v rámci e-shopu.
- **Upravení hodnot parametrů**
  - Uživatelé budou mít také možnost upravit hodnoty existujících vlastních parametrů. To umožní snadnou úpravu a aktualizaci informací o parametrech pro jednotlivé produkty.

Díky těmto funkcím uživatelé budou schopni efektivně spravovat vlastní parametry produktů a případně je začlenit mezi kategorické parametry podle potřeby. Tímto se zlepší struktura a přehlednost parametrů v rámci e-shopu.

3.4.1.3 Validace formuláře

Je klíčové zajistit, aby při tvorbě nebo úpravě parametrů nedocházelo k vytváření duplikovaných položek, které by mohly způsobit zmatek uživatelům. Proto je nezbytné provádět důkladnou validaci formulářů na straně klienta i serveru.

- **Kontrola duplicity parametrů v rámci kategorie**:

- Při vytváření nových parametrů je důležité kontrolovat unikátnost parametrů na základě vyplněných polí. Tím se zabrání vytvoření parametrů se stejnými vlastnostmi, což by mohlo vést k nesrovnalostem v systému. Provádění této kontroly pomáhá udržovat konzistenci a přehlednost v administraci e-shopů.

- **Kontrola unikátnosti nových parametrů**:
  - Při ukládání nových parametrů je nutné ověřit, že ve formuláři nejsou žádné položky se stejným identifikátorem. To zajišťuje, že v rámci jedné kategorie nevzniknou duplicitní parametry, což by mohlo vést k nedorozuměním a nekonzistencím v systému.

Zajištění těchto validací ve formuláři přispívá k zlepšení uživatelské zkušenosti a prevenci vzniku chyb ve správě parametrů. Díky nim je proces tvorby a úpravy parametrů efektivnější a spolehlivější.

3.4.1.4 Ukládání

Po úspěšné validaci dat je třeba data uložit. Pro parametry bez identifikátoru bude nutné vytvořit nový parametr a ten až posléze přidat mezi kategorické parametry. Navíc je třeba distribuovat kategorický parametr všem kategoriím, kterými jsou potomkem kategorie, do které se ukládá. Tato operace bude splňována pomocí jedné `PUT` operace. Tímto zajistíme, že nově vytvořený parametr bude k dispozici nejen v rámci aktuální kategorie, ale také ve všech potomcích. Proces distribuce kategorického parametru se provede automaticky jako součást uložení dat. To zajistí konzistenci a přehlednost parametrů napříč e-shopem.

Na základě tohoto popisu jsem vytvořil návrh rozhraní 3.9

3.4.2 Správa parametrů u produktu

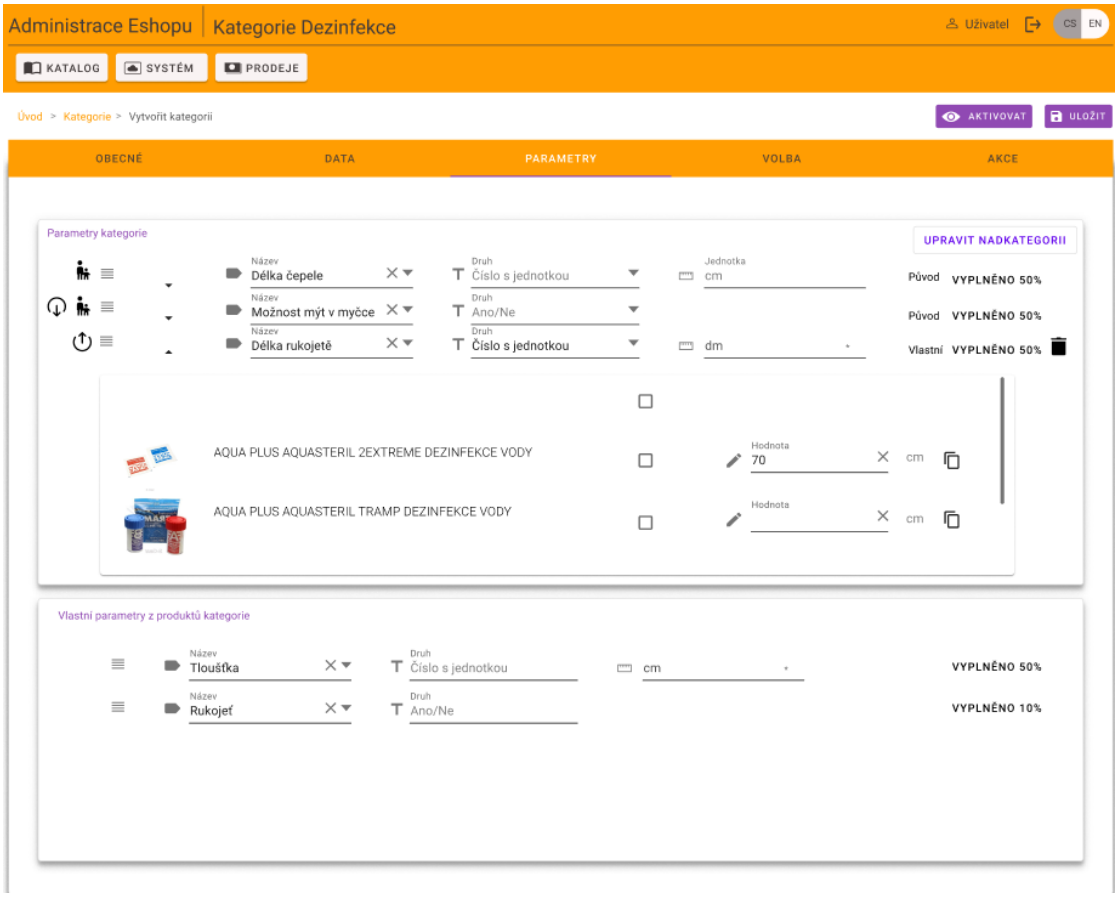
Formulář pro správu parametrů produktu se zaměřuje na úpravu a správu parametrů produktů v rámci e-shopu. Tento formulář bude strukturována podobně jako formulář pro správu kategorických parametrů, avšak s několika specifickými úpravami:

- **Podobnost s formulářem pro kategorické parametry**:
  - Forma pro správu parametrů produktu bude mít podobnou strukturu jako formulář pro kategorické parametry. To zahrnuje podobný layout, zobrazení seznamu parametrů a možnost jejich úprav a mazání.

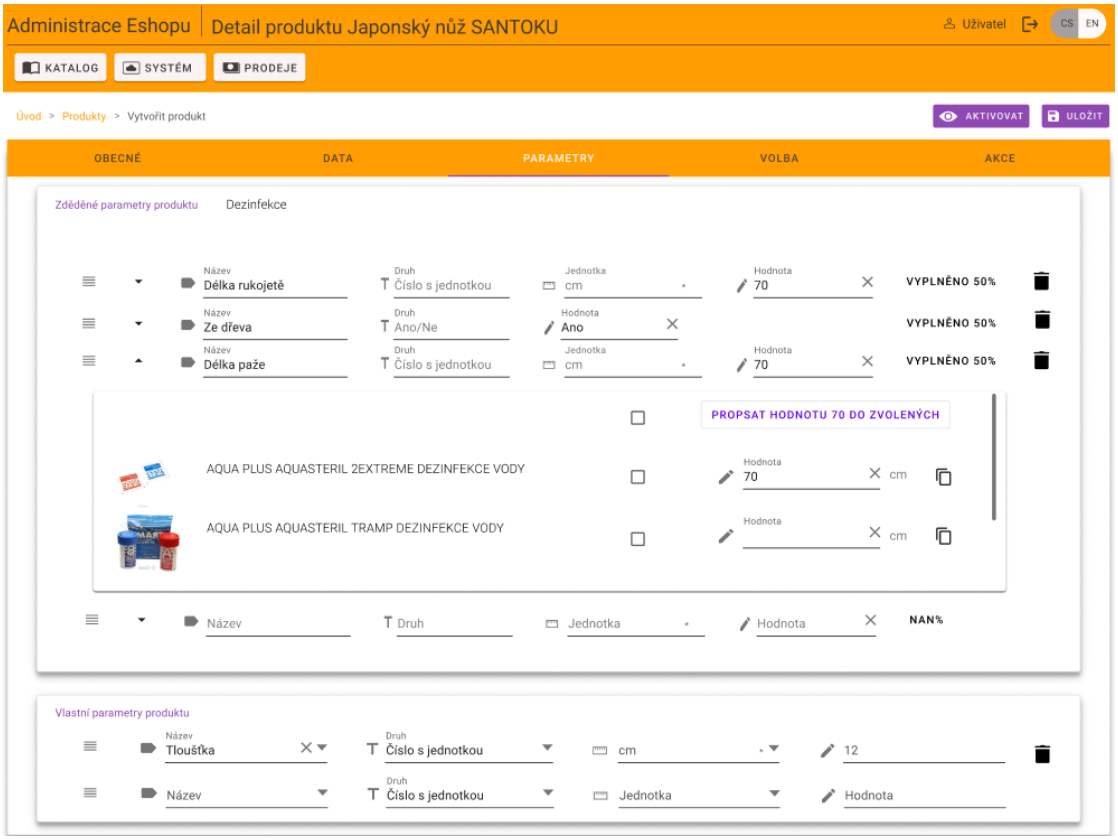
- **Rozšíření o pole pro úpravu hodnoty parametru**:
  - Jako hlavní rozdíl oproti formuláři pro kategorické parametry přibude do formuláře pole pro úpravu hodnoty parametru. Toto pole umožní uživatelům upravovat konkrétní hodnoty parametrů přímo pro daný produkt.

- **Omezení funkcionalit spojených s kategorickým stromem**:
  - Na rozdíl od formuláře pro kategorické parametry, ve formuláři pro správu parametrů produktu nebudou dostupné funkce používané k přesunu parametrů po kategorickém stromě. Uživatelé se zde zaměří především na úpravu a správu parametrů konkrétních produktů.

- **Zobrazení pouze vlastních parametrů produktu**:
  - V této části formuláře budou zobrazeny pouze parametrické hodnoty vázané k upravovanému produktu. Uživatelé zde budou moci přehledně vytvářet a upravovat hodnoty parametrů konkrétního produktu.



Obrázek 3.9 Navržené rozhraní formuláře pro správu kategorických parametrů



Obrázek 3.10 Navržené rozhraní formuláře pro správu parametrů u produktu

3.5 Pátá iterace

V páté iteraci jsem se zaměřil na úpravu parametrů a správu filtrovaných parametrů. Tato fáze práce přináší několik klíčových funkcionalit, které jsou zaměřeny na zlepšení správy parametrů v rámci administrace.

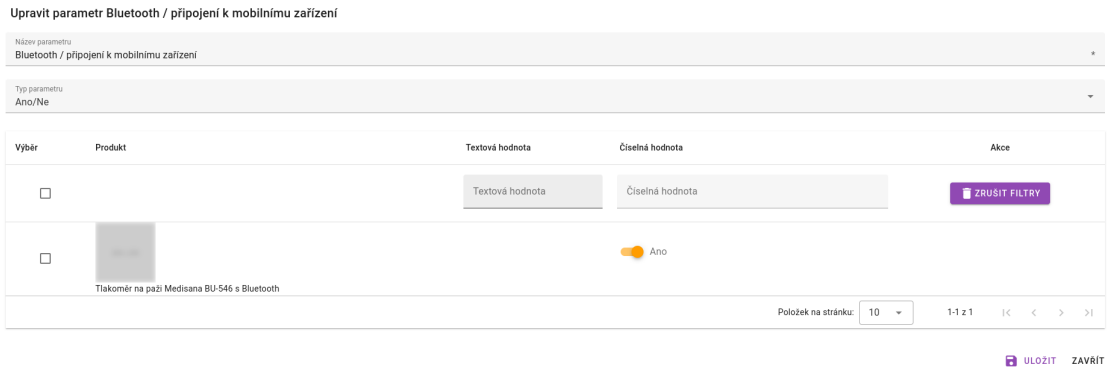
3.5.1 Úprava parametrů

Při úpravě parametru je důležité zvážit, jak bude probíhat změna typu. Umožnit uživatelům měnit typ parametru bez dalších akcí není vhodné, protože vyplněné hodnoty parametru ztrácejí relevanci při změně typu. Je proto nezbytné při úpravě typu parametru prezentovat uživateli obě vyplnitelné hodnoty z tabulky `oc_feature_product` 2.3 – `value_numeric` a `value_textual`, aby mohl při vyplňování jedné či druhé hodnoty vycházet z kontextu jedné z již vyplněných hodnot.

Pokud je v nastavení parametru definován textový nebo souborový typ, uživatelé budou moci upravovat hodnoty v poli `value_textual`. Pro booleovské a číselné typy bude určeno pole `value_numeric`. Takto zajišťujeme, že uživatel pracuje s odpovídajícím typem hodnoty v souladu s charakteristikou daného parametru.

Pro multiselektivní úpravy můžeme využít funkcionality podobné té, která je použita při vyplňování hodnot produktů, jak je definováno v tabulce parametrů 3.3.2. Tím uživatelům poskytneme možnost multiselekce z předem definovaných hodnot, což zvyšuje efektivitu a přesnost při úpravě parametrů. Tato funkce umožní uživatelům snadno vybrat více hodnot z nabízených možností a následně je aplikovat na vybrané produkty.

Z tohoto popisu jsem navrhl uživatelské rozhraní, které umožňuje uživatelům snadno upravovat parametry 3.11. Rozhraní obsahuje funkce pro změnu typu parametru a pro multiselektivní úpravy, což uživatelům umožňuje efektivněji pracovat s parametry. Následné uložení formuláře zapříčiní dvě `PUT` operace pro úpravu typu parametru a úpravu hodnot parametrů.



Obrázek 3.11 Navržené rozhraní formuláře pro správu parametrů u produktu

3.5.2 Správa filtrovaných parametrů

Pro správu filtrovaných parametrů je nutné vytvořit formulář, který umožní uživatelům spravovat tyto parametry. Formulář musí splňovat následující funkce:

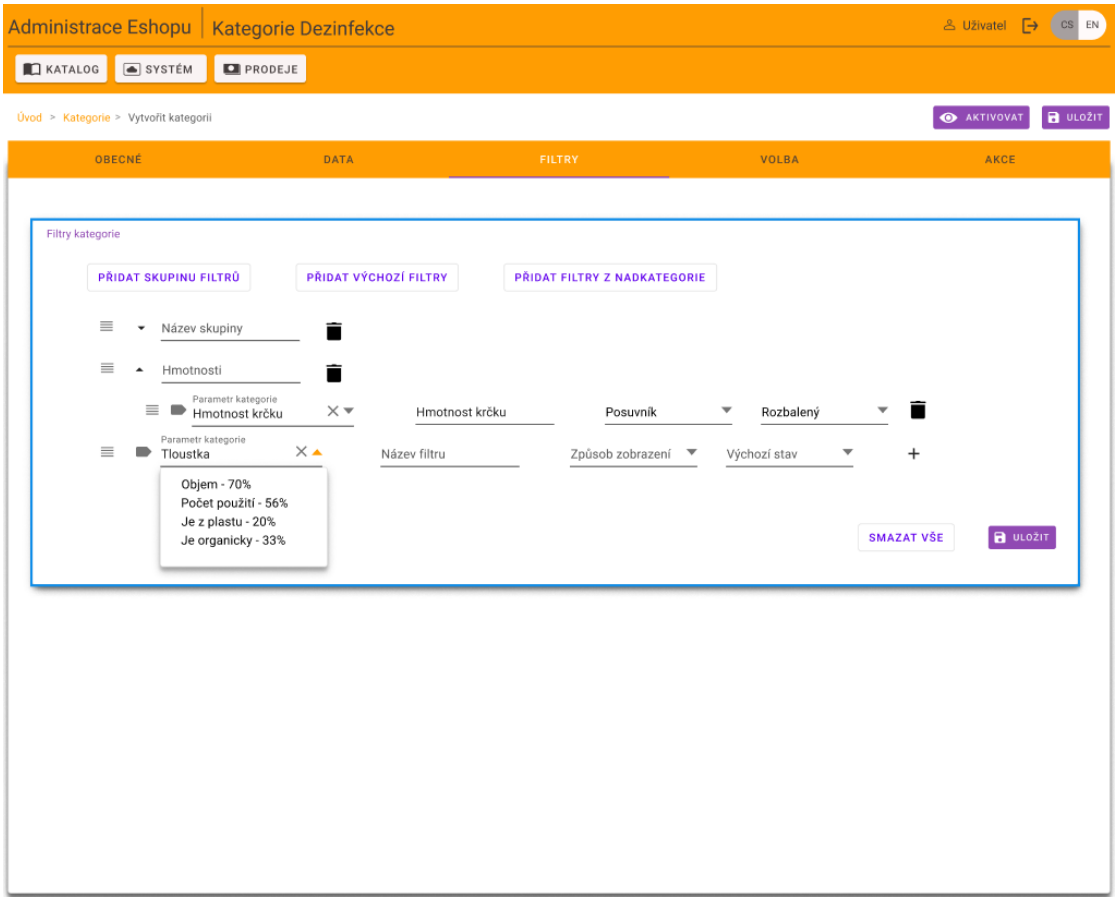
- **Přidání filtrovaného parametru**:
  - Uživatel bude mít možnost vybrat kategorické parametry, které se vyskytují v podstromu dané kategorie, a zobrazí se jim procentuální hodnota vyplněnosti, což je podíl vyplněných

produktových parametrů ku celkovému počtu produktů v kategorii. Dále bude moci uživatel parametry textově filtrovat a vyplnit pole relevantní s tabulkou `oc_feature_filter` 2.4.

- **Tvorba skupiny filtrů**:
  - Uživatel bude moci vytvořit skupinu filtrů s názvem a následně do ní přesouvat filtrované parametry.
- **Úprava filtru**:
  - Uživatel bude mít možnost svévolně upravovat pole spojená s filtrovanými parametry.
- **Smazání**:
  - Uživatel bude moci smazat filtr či skupinu.
- **Kopírování filtru ze společných kategorií**:
  - Uživatel bude moci nahrát filtry použité v jedné ze společných kategorií.
- **Přidání výchozích filtrů**:
  - Uživatel bude moci hromadně přidat výchozí filtry, jako jsou „Skladem“, „Cena“, „Barva“…

Způsob zobrazení je vázaný na typ parametru. Parametry číselného typu lze zobrazovat v posuvníku a výčtu. Ostatní parametry lze zobrazovat akorát výčtem. Jedinou výjimkou jsou systémové parametry, které mají pro svoje zobrazení nastavený speciální styl. Na základě popisu jsem znovu navrhl rozhraní formuláře 3.12.

V této kapitole jsem úspěšně navrhl řešení pro správu parametrů v souladu s funkčními požadavky. Kapitola poskytuje podrobný náhled na strukturu a fungování navrženého systému, který bude sloužit jako základ pro implementaci. Další kroky budou spočívat v konkrétním provedení navržených postupů a funkcionalit, čímž budeme přecházet k fázi implementace.



Obrázek 3.12 Navržené rozhraní formuláře pro správu filtrů

Kapitola 4
Implementace

V této kapitole se podrobněji zaměřím na postup při implementaci backendové a frontendové části mé práce. Prozkoumám použité knihovny, technologie a postupy, které jsem zvolil s cílem vytvořit funkční a efektivní systém. Tato část poskytne čtenáři hlubší vhled do konkrétních technických detailů a rozhodnutí, která jsem při implementaci učinil.

V rámci backendové části se budu zabývat použitými technologiemi pro tvorbu API, správu databáze a zpracování dat. Hlavním frameworkem zde bude .NET, doplněný o knihovnu AutoMapper pro usnadnění mapování objektů a Jagu FilterBundle – balíčkem pro parametrickou filtraci v kontextu databáze.

Pokud jde o frontendovou část, podrobně se zaměřím na použité technologie a nástroje pro vytvoření uživatelského rozhraní. Zahrnuji informace o Vue.js a knihovnách přizpůsobených tomuto frameworku.

4.1 Implementace backendové části

V implementační části backendu se soustředím na několik hlavních aspektů:

- **Tvorba tabulek pro překlady parametrů**:
  - Implementuji databázový model pro ukládání překladů parametrů. To zahrnuje vytvoření nových tabulek, které budou propojené s odpovídajícími tabulkami obsahujícími původní hodnoty parametrů.
- **Operace pro slučování parametrů**:
  - Implementuji logiku pro slučování parametrů. Tato operace bude umožňovat spojování existujících parametrů do jednoho nového parametru.
- **Tvorba endpointů s relevantními daty pro frontend**:
  - Vytvořím API endpointy, které budou poskytovat relevantní data pro frontend. To může zahrnovat seznamy parametrů, překlady, informace o kategoriích a další potřebné informace pro správné zobrazení a manipulaci s daty na frontendu.

4.1.1 Tvorba tabulek pro překlady parametrů

V této části popíšeme postup tvorby databázových tabulek pro ukládání překladů parametrů. Cílem této části je demonstrovat proces vytváření databázové struktury pro podporu multijazyč-

ných překladů parametrů v našem informačním systému. Pro tvorbu tabulek využijeme Entity Framework [27], který je součástí .NET frameworku.

4.1.1.1 Definice tříd pro entity

Pro každou tabulku v databázi, která bude sloužit k ukládání překladů parametrů, vytvoříme ekvivalentní třídu v .NET. Třídy budou obsahovat vlastnosti odpovídající sloupcům v tabulkách 3.6, navíc budou anotované k vytvoření chtěného jména tabulky a určení primárního klíče 1.

```
[Table("feature_translation")]
public class FeatureTranslation
{
    [Key]
    public int FeatureTranslationId { get; set; }
    public int FeatureId { get; set; }
    public int LanguageId { get; set; }
    public Feature Feature { get; set; } = null!;
    public Language Language { get; set; } = null!;
    public string Name { get; set; } = null!;
    public string? Definition { get; set; } = null!;
}
```

Výpis kódu 1 Třída pro překlad `oc_feature`

4.1.1.2 Vytvoření kontextu databáze

V třídě dědící od třídy `DbContext` definujeme vlastnosti typu `DbSet` pro každou entitu, kterou chceme mapovat do databáze 2.

Třída `DbContext` poskytuje prostředky pro práci s daty v databázi, včetně vytváření, čtení, aktualizace a mazání dat.

```
public DbSet<FeatureTranslation> FeatureTranslations { get; set; } = null!;
public DbSet<ProductFeatureTranslation> FeatureProductTranslations { get; set; } = null!;
public DbSet<FeatureGroupTranslation> FeatureGroupTranslations { get; set; } = null!;
public DbSet<FeatureFilterTranslation> FeatureFilterTranslations { get; set; } = null!;
```

Výpis kódu 2 Definice databázového kontextu pro překlady

4.1.1.3 Konfigurace vztahů

Jelikož chceme, aby překladové tabulky byly vázané s překládanými hodnotami, musíme konfigurovat vztahy mezi jednotlivými entitami za pomocí atributů pro konfiguraci vztahů.

Zde je vysvětlení atributů pro tabulku `oc_feature_product` použitých v kódu 3:

- **EntityTypeBuilder**:
  - Je objekt, který poskytuje metody pro konfiguraci mapování entity.
- **HasMany**:
  - Tato metoda definuje vztah „má mnoho“ mezi dvěma entitami. Zde se nastavuje vztah mezi ProductFeature a ProductFeatureTranslation, kde jeden ProductFeature může mít více ProductFeatureTranslation.

- **WithOne**:
  - Tato metoda definuje druhý konec vztahu „má mnoho“. Zde se určuje, že jedna instance ProductFeatureTranslation je propojena s jedním ProductFeature.
- **HasForeignKey**:
  - Tato metoda určuje cizí klíč v tabulce ProductFeatureTranslation, který odkazuje na primární klíč v tabulce ProductFeature. Pomocí tohoto klíče jsou provázány obě tabulky. Pořadí identifikátorů by mělo odpovídat pořadí, ve kterém jsou tyto identifikátory definovány v kompozitním klíči cílové entity.
- **OnDelete**:
  - Tato metoda určuje akci, která se má provést v případě smazání záznamu v tabulce nadřazené entity (ProductFeature). V tomto případě se nastavuje DeleteBehavior.Cascade, což znamená, že všechny překlady parametru (ProductFeatureTranslation) budou smazány, pokud je smazán příslušný parametr (ProductFeature).

```
public class FeatureProductConfiguration : IEntityTypeConfiguration<ProductFeature>
{
    public void Configure(EntityTypeBuilder<ProductFeature> builder)
    {
        builder.HasMany<ProductFeatureTranslation>(pf => pf.Translations)
            .WithOne(x => x.ProductFeature)
            .HasForeignKey(x => new {x.ProductId, x.FeatureId})
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Výpis kódu 3 Konfigurace vztahů pro tabulku `oc_feature_product`

4.1.1.4 Migrace databáze

Pomocí Entity Frameworku můžeme provést migrace. Pro vytvoření migrace použijeme příkaz `dotnet ef migrations add FeatureTranslations`, který vygeneruje migraci na základě změn v definicích entit. Následně pomocí příkazu `dotnet ef database update` můžeme aplikovat změny do databáze.

Díky tomuto postupu budou entity překladů přidané do databáze a nakonfigurované pro budoucí vývoj.

4.1.2 Operace pro slučování parametrů

Operace sloučení bude asi nejvíc komplexní operace co bych mohl z operací popisovat, žádá se po ní, aby sloučila dané parametry a zároveň v nějakých případech upravila vyplněné hodnoty parametrů podle uživatelsky zadaných vstupů. Sloučení parametru znamená přepis všech zadaných parametrů na stejný identifikátor a převázání předchozích vztahů. Nejprve bych začal tím, že si určíme jak bude žádost vypadat. Kdybychom spojili všechny formuláře pro slučování, které byly vytvořeny při návrhu 3.2.1.2, obdrželi bychom žádost v této podobě.

```
public class MergeRequest
{
    public string NewName { get; set; } = null!;
    public string? NewDefinition { get; set; } = null!;
    public MergeTypeEnum MergeType { get; set; }
    public FeatureMergeRequest[] FeatureMergeRequests { get; set; } = null!;
}
```

Výpis kódu 4 Třída žádosti pro operaci sloučení

```
public enum MergeTypeEnum
{
    Numbers, Booleans, NumbersToBool, BoolToNumbers, Other
}
```

Výpis kódu 5 Typy sloučení

```
public class FeatureMergeRequest
{
    public int FeatureId { get; set; }
    public string FeatureType { get; set; } = null!;
    public int ?Multiplier { get; set; }
    public ComparisonEnum ?Comparison { get; set; }
    public int ?ComparisonValue { get; set; }
    public bool ?Negate { get; set; }
}
```

Výpis kódu 6 Třída žádosti sloučení pro parametr

```
public enum ComparisonEnum
{
    Lt, // <
    Gt, // >
    Eq, // =
    Ne, // !=
    Lte, // <=
    Gte // >=
}
```

Výpis kódu 7 Povolené typy porovnání

Po obdržení žádosti o sloučení se backendová část aplikace musí postarat o provedení samotné operace. Kvůli skutečnosti, že většina tabulek obsahuje primární klíč složený z cizích klíčů, není možné jednoduše provést převázání tabulek. Místo toho je třeba vytvořit transakci, která umožní vytvoření nových záznamů a smazání starých záznamů, místo přímého převázání cizích klíčů.

Jelikož je část kódu příliš dlouhá, akorát popíšu postup:

1. Získání relevantních parametrů:
   - Backendová část aplikace musí identifikovat všechny parametry, které mají být sloučeny na základě poskytnuté žádosti.
2. Vytvoření nových záznamů:
   - Pro každý nový sloučený parametr je třeba vytvořit nový záznam v příslušné tabulce s odpovídajícím primárním klíčem. To zahrnuje vytvoření nových záznamů v tabulkách, které obsahují cizí klíče na parametr. Navíc je třeba podle popsané logiky 3.2.1.2 pro `oc_feature_product`, převést hodnoty, tak aby odpovídaly logice operace.
3. Aktualizace vztahů:
   - Po vytvoření nových záznamů je třeba aktualizovat všechny vztahy, které byly spojeny s původními parametry. To zahrnuje aktualizaci cizích klíčů v příslušných tabulkách (pro překladové tabulky nejsou cizí klíče primární).
4. Smazání původních záznamů:
   - Nakonec je třeba smazat původní záznamy parametrů, které byly sloučeny.

4.1.3 Tvorba endpointů s relevantními daty pro frontend

V této sekci se zaměřím na tvorbu endpointů, které poskytují relevantní data pro frontend. Bude mi primárně jít o demonstraci anotace endpointu, využití AutoMapperu a FilterBundelu k získání potřebných informací.

Jeden z endpointů, nad kterým můžu funkce demonstrovat je `GET` operace pro získání stránkovaných filtrů.

Tento endpoint je `GET` endpoint 8, který slouží k získání seznamu vlastností (features). Zde je popis funkcí a anotací použitých v tomto kódu:

- `[Authorize(Roles = Roles.TopAdmin)]`:
  - Toto atributem omezí přístup k tomuto endpointu pouze na uživatele s rolí TopAdmin. To znamená, že pouze uživatelé s touto rolí budou mít povolený přístup k tomuto endpointu.
- `[HttpGet(Name = "GetFeatures")]`:
  - Tento atribut určuje HTTP metodu `GET` pro tento endpoint a nastavuje jméno routy na ”GetFeatures”.
- `[SwaggerOperation]`:
  - Tento atribut definuje operaci dokumentace Swaggeru. Summary poskytuje stručný popis operace, zatímco OperationId určuje jednoznačný identifikátor operace.
- `[ProducesResponseType]`:

- Tyto atributy určují očekávané typy odpovědí pro tento endpoint a odpovídající HTTP status kódy. `StatusCodes.Status200OK` označuje úspěšnou odpověď s kódem 200, která vrací seznam vlastností ve formátu `PagedResult<FeatureResponse>`.

  `StatusCodes.Status400BadRequest` označuje chybovou odpověď s kódem 400, která vrací informace o neplatném požadavku.

- `[SwaggerFilterable(typeof(FeatureQuery))]`:
  - Tento atribut indikuje, že tento endpoint je filtrovatelný pomocí specifikovaného typu FeatureQuery. To znamená, že uživatelé mohou aplikovat filtry na tento endpoint pomocí parametrů z FeatureQuery 9.
- `public async Task<Results<Ok<PagedResult<FeatureResponse>>, BadRequest<BadRequestResponse>>> GetAll([FromQuery] SearchParams searchParams, CancellationToken cancellationToken)`:
  - Toto je metoda obsluhující `GET` požadavek na získání všech vlastností. Parametr `[FromQuery] SearchParams searchParams` umožňuje přijímat parametry z query řetězce, které mohou být použity pro filtrování nebo stránkování výsledků. Metoda vrací výsledek v obalu Results, který může být buď úspěšný (v případě kódu 200) nebo neúspěšný (v případě chyby).
  - Pokud je požadavek úspěšný, vrátí metoda seznam vlastností ve stránkovaném formátu (`PagedResult<FeatureResponse>` 10) v obalu Ok.
  - Pokud se vyskytne chyba, vrátí metoda chybovou odpověď v obalu BadRequest s odpovídajícími informacemi o chybě.

Tento kód 11 využívá knihovnu AutoMapper v jazyce C# k definici mapování mezi entitou Feature a odpovídající odpovědí FeatureResponse. Každý řádek v mapování specifikuje, jaká data se mají převést z jedné entity do druhé. Použití LINQ (Language Integrated Query) [28] umožňuje provádět složité operace na kolekcích dat, jako je například počítání počtu vyplněných hodnot parametrů.

```
[Authorize(Roles = Roles.TopAdmin)]
[HttpGet(Name = "GetFeatures")]
[SwaggerOperation(Summary = "Get list of features", OperationId = "GetFeatures")]
[ProducesResponseType(typeof(PagedResult<FeatureResponse>), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(BadRequestResponse), StatusCodes.Status400BadRequest)]
[SwaggerFilterable(typeof(FeatureQuery))]
public async Task<Results<Ok<PagedResult<FeatureResponse>>,
BadRequest<BadRequestResponse>>> GetAll(
    [FromQuery] SearchParams searchParams, CancellationToken cancellationToken)
{
    try
    {
        return Ok(await filterableMapper.GetPaginated<Feature, FeatureQuery,
        FeatureResponse>(searchParams,
            cancellationToken));
    }
    catch (BaseFilterableException e)
    {
        return BadRequest(new BadRequestResponse(e));
    }
}
```

Výpis kódu 8 `GetFeatures` endpoint

```
public class FeatureQuery : FilterMetadata<Feature>
{
    public FeatureQuery()
    {
        MapColumn("featureId");
        MapColumn("name");
        MapColumn("type");
        MapColumn("sortOrder");
        MapCollection("categoryFeatures", mapping =>
        {
            mapping.MapColumn("categoryId");
        });
        WithPrimaryKey("featureId");
        AddDefaultSort("sortOrder", true);
    }
}
```

Výpis kódu 9 Třída FeatureQuery

```
public class FeatureResponse
{
    public int FeatureId { get; set; }
    public string Name { get; set; } = "";
    public string Type { get; set; } = "";
    public string Unit { get; set; } = "";
    public int Filled { get; set; } = 0;
    public string[] CategoryNames { get; set; } = null!;
    public int[] CategoryIds { get; set; } = null!;
    public int[] ProductIds { get; set; } = null!;
    public string[] CategoryDomains  { get; set; } = null!;
    public ICollection<TranslationResponse> Translations { get; set; } = null!;
}
```

Výpis kódu 10 Třída FeatureResponse

```
CreateMap<Feature, FeatureResponse>()
    .ForMember(dst => dst.Unit, opt => opt.MapFrom(src => src.Definition))
    .ForMember(dst => dst.Filled,
        opt => opt.MapFrom(src =>
            src.ProductFeatures.Count(pf =>
            pf.ValueTextual != null || pf.ValueNumeric != null)))
    .ForMember(dst => dst.CategoryNames,
        opt => opt.MapFrom(src =>
            src.CategoryFeatures.Select(cf =>
            cf.Category.CategoryDescriptions.Select(cd =>
            cd.Name).FirstOrDefault() ?? "Neznámá kategorie")))
    .ForMember(dst => dst.CategoryIds,
        opt => opt.MapFrom(src => src.CategoryFeatures.Select(cf => cf.CategoryId)))
    .ForMember(dst => dst.CategoryDomains,
        opt => opt.MapFrom(src =>
            src.CategoryFeatures.Select(cf => cf.Category.DefaultDomain)))
    .ForMember(dst => dst.ProductIds,
        opt => opt.MapFrom(src => src.ProductFeatures.Select(pf => pf.ProductId)));
```

Výpis kódu 11 Namapování Feature na FeatureResponse

4.2 Implementace frontendové části

V této sekci se zaměříme na implementaci frontendové části systému. Budeme se věnovat několika hlavním aspektům:

- **Reaktivita ve Vue.js**
  - Vue.js je framework, který je známý svou reaktivní povahou. V této části popíšeme, jak využíváme reaktivních vlastností Vue.js k efektivní manipulaci s daty a synchronizaci s uživatelským rozhraním. [1]
- **Použití knihovny komponent Vuetify**
  - Vuetify je populární knihovna komponent pro Vue.js, která poskytuje bohatou sadu předem navržených komponent a stylů. V této sekci vysvětlíme, jak Vuetify využíváme k rychlému a jednoduchému vytváření uživatelského rozhraní s moderním designem. [29]
- **Použití Vuex store**
  - Vuex je správce stavu pro Vue.js aplikace, který umožňuje centrálně spravovat stav aplikace. V této části popíšeme, jak používáme Vuex k uchovávání a správě stavu aplikace, a jak propojujeme data mezi komponentami a stavem Vuex. [30]

Tímto způsobem tato kapitola poskytne komplexní přehled o implementaci frontendové části systému s důrazem na reaktivitu, design komponent a správu stavu.

4.2.1 Reaktivita ve Vue.js

Reaktivita ve Vue.js je základním konceptem, který umožňuje automatickou aktualizaci UI v reakci na změny dat. Tento mechanismus je založen na sledování datových objektů a propojení s nimi v DOM, což umožňuje efektivní a dynamickou aktualizaci uživatelského rozhraní.

Klíčovými prvky reaktivity ve Vue.js jsou:

- Data Binding (Vázání dat):
  - Vue umožňuje propojení dat v JavaScriptových objektech s HTML šablonami pomocí tzv. data bindingu. To znamená, že změny v datech jsou automaticky propagovány do DOM a naopak. [31]
- Reaktivní vlastnosti (Reactive Properties):
  - Vue sleduje datové objekty a detekuje jejich změny. Kdykoliv se data změní, Vue automaticky způsobí překreslení relevantních částí uživatelského rozhraní. [31]
- Watchers (Sledovače):
  - Vue umožňuje sledovat změny v datech pomocí tzv. watcherů. Tyto funkce umožňují spouštět určité akce nebo manipulovat s daty v reakci na jejich změny. [32]

Komponenty ve Vue.js jsou znovupoužitelné a samostatné bloky kódu, které obsahují šablonu, logiku a styly. Každá komponenta může mít své vlastní data, metody a životní cyklus. Tyto komponenty mohou být následně vnořeny do jiných komponent nebo použity v rámci celé aplikace. [33]

Tento kód 12 vytváří záložky pro navigaci mezi překladovými tabulkami pomocí komponenty `<v-tabs>`. Hlavní vlastností je `v-model`, která určuje aktivní záložku. V záložkách jsou iterovány položky z pole `tabs`, které obsahují názvy záložek a komponenty, které mají být zobrazeny, `langPath` určuje cestu k lokalizovaným názvům záložek, `chosenTab` udržuje aktivní záložku pomocí reaktivní reference a `tabs` je vypočítaná vlastnost, která vrátí pole objektů reprezentujících jednotlivé záložky.

```
<v-tabs
  v-model="chosenTab"
  bg-color="primary"
  slider-color="secondary"
  slider-size="3"
  grow
>
  <!--tabs-->
  <v-tab v-for="(tab, tabIndex) in tabs" :key="'main-tab-' + tabIndex">
    {{ $t(langPath + tab.name) }}
  </v-tab>
</v-tabs>
...
const langPath = "translations.tabs.";
const chosenTab = ref(0);
const tabs = computed((): Tab[] => {
  return [
    {
      name: "parameters",
      component: ParameterTranslationsTable,
      idx: 0,
    },
    {
      name: "productParameters",
      component: ProductParameterTranslationsTable,
      idx: 1,
    },
    {
      name: "parameterFilters",
      component: ParameterFilterTranslationsTable,
      idx: 2,
    },
    {
      name: "parameterGroups",
      component: ParameterGroupTranslationsTable,
      idx: 3,
    },
  ];
});
```

Výpis kódu 12 Reaktivnost a data binding mezi záložkami pro překladové tabulky

4.2.2 Použití Vuex store

Vuex je knihovna pro správu stavů ve Vue.js aplikacích, která umožňuje udržovat globální stav aplikace a řídit jeho změny prostřednictvím předdefinovaných mutací. Tento stav je obvykle uchováván v úložišti (store), ke kterému lze přistupovat z libovolné komponenty v aplikaci.

V poskytnutém kódu 13 je vytvořeno Vuex úložiště pro stránku překladu parametrů. Úložiště obsahuje různé položky stavu, jako jsou seznamy jazyků a požadavky na překlad různých typů parametrů. Úložiště je rozděleno do několika částí, včetně stavu, mutací, akcí a getterů:

- **Stav (State)**: Obsahuje instance třídy `State`, která definuje různé položky stavu, jako jsou seznamy jazyků a seznamy požadavků na překlad parametrů.
- **Mutace (Mutations)**: Obsahují metody pro změnu stavu. Každá mutace přijímá aktuální stav a novou hodnotu a provádí změnu stavu na základě těchto hodnot. Například metoda `setLanguages` nastaví seznam jazyků ve stavu na základě poskytnutého pole jazyků.
- **Akce (Actions)**: Obsahují metody pro asynchronní změny stavu nebo složitější operace. Tyto akce mohou provádět různé operace, jako je načítání dat ze serveru nebo zpracování akcí uživatele. V tomto případě jsou akce jednoduše delegovány na odpovídající mutace.
- **Gettery (Getters)**: Poskytují způsob přístupu k stavu úložiště a získání dat z něj. Tyto gettery jsou často používány v komponentách k získání aktuálního stavu aplikace.

Tímto způsobem je Vuex úložiště použito k udržování a správě stavu aplikace pro stránku překladu parametrů, což umožňuje konzistentní správu dat a snadnou synchronizaci mezi různými částmi aplikace.

V této kapitole jsem se zaměřil na shrnutí klíčových aspektů implementace backendové a frontendové části projektu. Bylo prezentováno pouze omezené množství implementačních detailů, které poskytují základní vhled do procesu vytváření systému pro správu parametrů v administraci e-shopu. Další implementace ostatních funkcí nebyla zahrnuta, protože není v této fázi projektu nezbytná.

```
class State {
  languages = [] as LanguageResponse[];
  featureTranslationRequests = [] as FeatureTranslationRequest[];
  featureFilterTranslationRequests = [] as FeatureFilterTranslationRequest[];
  featureGroupTranslationRequests = [] as FeatureGroupTranslationRequest[];
  productFeatureTranslationRequests = [] as FeatureTranslationRequest[];
}

const translations = {
  namespaced: true,
  state: new State(),
  mutations: {
    setLanguages: (state: State, newValue: LanguageResponse[]): void => {
      state.languages = newValue;
    },
...
  actions: {
    setLanguages: (
      { commit }: { commit: Commit },
      newValue: LanguageResponse[],
    ): void => {
      commit("setLanguages", newValue);
    },
...
  getters: {
    languages: (state: State): LanguageResponse[] => state.languages,
...
```

Výpis kódu 13 Store pro stránku překladů parametrů

Kapitola 5
Testování

V této kapitole se zaměřím na testování implementovaných funkcionalit, přičemž detailně popíšu testy z hlediska použitých technologií a postupů. První část bude věnována automatickým testům, kde se budu zabývat jak jednotkovými (unit) testy, tak integračními testy a E2E testy. Následně se budu věnovat i manuálnímu testování, které zahrnuje uživatelské testování. To je klíčové pro ověření uživatelské přívětivosti a kvality uživatelského prostředí systému. Tato kapitola poskytne ucelený přehled o testování implementovaných funkcí a zajištění kvality systému z hlediska různých perspektiv.

5.1 Automatické testy

Automatické testy jsou testovací procedury, které jsou prováděny automaticky bez nutnosti přímé interakce člověka s aplikací. Mohou být vytvořeny pomocí různých testovacích frameworků a knihoven a jsou obvykle integrovány do procesu vývoje softwaru.

5.1.1 Statická analýza

Na frontendové části byl v rámci statické analýzy použit nástroj ESLint. ESLint je nástroj pro statickou analýzu kódu v JavaScriptu, který slouží k identifikaci a upozornění na potenciální chyby, nekonzistence a špatné praktiky v kódu. Jeho použití přispívá k zlepšení kvality a konzistence kódu v projektu tím, že poskytuje vývojářům zpětnou vazbu na problematické části kódu a umožňuje dodržování definovaných pravidel a standardů. [34] V rámci použití ESLintu byla aplikována pravidla zaměřená na analýzu TypeScriptu, Vue.js a formátování kódu.

5.1.2 Jednotkové (Unit) testy

Jednotkové testy slouží k ověření správnosti funkcionality jednotlivých částí (jednotek) kódu, obvykle metod nebo funkcí, izolovaně od zbytku aplikace. Cílem unit testů je detekovat a odstranit chyby v implementaci jednotlivých funkcí a zajistit, že tyto funkce fungují podle očekávání.

Tyto testy 14 jsou napsány pomocí frameworku Xunit, který je populární pro psaní unit testů v .NET [35]. V těchto testech se zaměřujete na funkci `ProductFeatureNumberToBool` třídy `ProductFeatureService`, která slouží k převodu číselného parametru na boolean hodnotu podle zadaných podmínek.

První testovací metoda ověřuje, zda metoda `ProductFeatureNumberToBool` správně nastaví hodnotu číselného parametru na 1 v případě, že číselná hodnota splňuje podmínku zadanou v `FeatureMergeRequest`.

Druhá testovací metoda testuje, zda metoda `ProductFeatureNumberToBool` vyhodí výjimku typu `ComparisonNotSupportedException` v případě, že zadaná podmínka v `FeatureMergeRequest` není podporována.

Obě testovací metody používají aserce z frameworku Xunit pro ověření očekávaných výsledků a výjimek. Tyto testy pomáhají zajistit, že převod číselných parametrů na boolean hodnoty probíhá správně a že se řádně zachází s nepodporovanými podmínkami.

```
public class ProductFeatureServiceTests
{
    [Fact]
    public void ProductFeatureNumberToBool_ShouldSetCorrectValue()
    {
        var productFeature = new ProductFeature { ValueNumeric = 5 };
        var feature = new Feature { Type = FeatureConstants.ParameterTypeUnit };
        var featureMergeRequest = new FeatureMergeRequest()
        { Comparison = ComparisonEnum.Gt, ComparisonValue = 3 };

        ProductFeatureService.ProductFeatureNumberToBool(
        productFeature, feature, featureMergeRequest);

        Assert.Equal(1, productFeature.ValueNumeric);
    }

    [Fact]
    public void ProductFeatureNumberToBool_ShouldThrowExceptionForUnsupportedComparison()
    {
        var productFeature = new ProductFeature { ValueNumeric = 5 };
        var feature = new Feature { Type = FeatureConstants.ParameterTypeUnit };
        var featureMergeRequest = new FeatureMergeRequest
        { Comparison = (ComparisonEnum)999, ComparisonValue = 3 };

        Assert.Throws<ComparisonNotSupportedException>(() =>
            ProductFeatureService.ProductFeatureNumberToBool
            (productFeature, feature, featureMergeRequest));
    }
}
```

Výpis kódu 14 Unit test pro převod číselného parametru na booleovský

5.1.3 Integrační testy

Integrační testy jsou automatické testy, které se zaměřují na ověření správné interakce mezi jednotlivými komponentami či moduly aplikace. Na rozdíl od unit testů, které testují jednotlivé části kódu izolovaně, integrační testy se zaměřují na testování integrace a spolupráce mezi různými částmi aplikace. [36]

Důležitost integračních testů spočívá v jejich schopnosti odhalit chyby a problémy v komunikaci mezi jednotlivými částmi aplikace, které by mohly být přehlédnuty při testování jednotlivých

komponent samostatně. Integrační testy nám pomáhají zajistit, že naše aplikace funguje jako celek a že všechny komponenty spolu správně komunikují a integrují se. [36]

V rámci mého projektu jsem se rozhodl implementovat základní integrační testy, které jsou velmi minimálního rozsahu. Tyto testy se zaměřují převážně na odesílání požadavků na server a kontrolují, zda odpověď odpovídá očekávanému formátu. Při těchto testech se nestarám příliš o obsah odpovědi, ale spíše o to, zda požadavek projde a jestli je vrácena odpověď s očekávaným statusem.

Rozhodl jsem se pro tuto minimalistickou formu integračních testů z důvodu časového omezení a zaměření na nejzákladnější kontrolu funkcionality aplikace. I přesto, že tyto testy nejsou komplexní, poskytují určitou úroveň jistoty, že základní komunikace funguje správně. Z časových důvodů jsem nestihl upravit data pro testování natolik, aby testy byly funkční, tudíž je třeba ještě na tvorbě testovacích dat zapracovat.

5.1.4 E2E testy

End-to-end (E2E) testy jsou automatické testy, které simuluji reálné uživatelské scénáře, při kterých jsou testovány aplikace od začátku až do konce, včetně všech interakcí s uživatelským rozhraním a backendovou funkcionalitou. Tyto testy se snaží o simulaci chování skutečných uživatelů a ověření, že aplikace funguje správně jako celek. [37]

V našem případě jsme se k E2E testům nedostali z několika důvodů. Zaprvé, náš projekt nebyl nikde nasazený, což by bylo nezbytné pro spuštění E2E testů v reálném prostředí. Za druhé, pro úspěšné spuštění E2E testů bychom museli zajistit dostatečné množství testovacích dat, což by vyžadovalo dodatečný čas a zdroje, které jsme momentálně neměli k dispozici. Nakonec, vzhledem k časovému tlaku v týmu jsme se rozhodli prioritizovat jiné aktivity a nedostali jsme se k implementaci E2E testů.

5.2 Manuální testování

Při manuálním testování, které probíhalo za pochodu vývoje aplikace, jsem se opíral o Nielsen-Normanovy heuristiky, což jsou osvědčené zásady používání a designu uživatelských rozhraní. Tyto heuristiky mi pomohly identifikovat potenciální problémy a nedostatky v uživatelském rozhraní mé aplikace. [38] Zde jsou některé z hlavních heuristik, které jsem použil a jak jsem je aplikoval:

- **Jednoduchost a minimalismus:**
  - Zkontroloval jsem, zda je rozhraní jednoduché a snadno použitelné. Minimalizoval jsem nepotřebné prvky a zjednodušil jsem navigaci.
- **Konzistence a standardy:**
  - Zajistil jsem, že veškeré interakce a navigace v aplikaci jsou konzistentní a odpovídají běžným standardům uživatelského designu.
- **Flexibilita a efektivita použití:**
  - Testoval jsem flexibilitu a efektivitu aplikace v různých scénářích použití a zjišťoval jsem, zda je možné dosáhnout cílů uživatele různými způsoby.
- **Chybové zprávy a prevence chyb:**
  - Testoval jsem, zda jsou chybové zprávy srozumitelné a informativní a zda je aplikace navržena tak, aby minimalizovala vznik chyb.

[38]

5.2.1 Uživatelské testování

Uživatelské testování je proces, který slouží k ověření použitelnosti, efektivity a uživatelské spokojenosti s daným produktem nebo systémem. Cílem uživatelského testování je získat užitečnou zpětnou vazbu od skutečných uživatelů a identifikovat potenciální problémy nebo nedostatky, které mohou být při používání produktu nebo systému.

Pro testování jsem si připravil několik věcí:

- **Úvodní dotazník**:
  - Dotazník sloužil k zjištění zkušenosti a pohodlí uživatelů se stávajícím systémem nebo podobnými aplikacemi.
- **Dotazník spokojenosti**:
  - Dotazník byl určen k vyplnění po testu a zkoumal spokojenost uživatelů s nově testovanou verzí mého projektu.
- **Testovací scénáře**:
  - Připravil jsem scénáře testování, které obsahovaly úkoly a kroky, jež uživatelé měli provést během testování. Tyto scénáře byly navrženy tak, aby pokryly většinu funkcí a možností mého projektu a umožnily uživatelům interagovat s klíčovými částmi aplikace.

5.2.1.1 Průběh uživatelského testování

Příprava na samotné uživatelské testování byla relativně jednoduchá, protože mi vedoucí projektu pomohl sehnat skupinu uživatelů, kteří již používají stávající administraci. Stačilo se s nimi domluvit na čase a způsobu provedení testování.

Samotné provedení testování probíhalo následovně:

1. Uživatele jsem poprosil, aby se připojili ke mně do hovoru, abychom mohli komunikovat a já mohl vysvětlit postup testování.
2. Poté jsem jim poskytl referenční kód aplikace AnyDesk [39], což je nástroj pro vzdálený přístup k počítači. Tímto způsobem měli možnost vidět a interagovat s mým projektem přímo na svém zařízení.
3. Před samotným začátkem testování jsem uživatele zdvořile požádal o souhlas s nahráváním hovoru. Tento souhlas mi umožnil zaznamenat průběh testování a mít tak k dispozici záznam, který jsem si mohl později prohlédnout a analyzovat.
4. Posledně následovalo vyplnění úkolů ze strany uživatele. Do průběhu testování jsem zasahoval pouze tehdy, kdy si uživatel nevěděl rady a řekl si o pomoc.

Tento způsob provedení uživatelského testování mi umožnil získat přímou zpětnou vazbu od uživatelů a lépe porozumět jejich potřebám a zkušenostem s mou aplikací. Zároveň mi poskytl možnost zpětně prozkoumat průběh testování a analyzovat jednotlivé interakce a reakce uživatelů.

5.2.1.2 Výsledky uživatelského testování

V rámci provedeného uživatelského testování jsem pracoval s čtyřmi uživateli, z nichž tři byli zkušení uživatelé současné administrace a jeden uživatel, který s administrací nikdy předtím nepracoval. Ale přesto se uživatel bez zkušenosti s administrací dokázal v systému zorientovat podobně jako zkušení uživatelé.

Cíle testování byly obecně zadány, což vedlo k různým přístupům, kterými uživatelé řešili úkoly. Tato variabilita v přístupech poskytovala ucelenější pohled na to, jak uživatelé interagují

se systémem a jak se s ním adaptovali. Taková široká škála reakcí je z hlediska uživatelského testování velmi hodnotná, protože umožňuje odhalit různé uživatelské preference, schopnosti a potřeby.

Například při úkolu registrace parametrů pro dva nové produkty, pro které ještě nebyl vytvořen žádný parametr v rámci kategorií. Uživatelé si buď zvolili cestu úpravy každého produktu zvlášť, nebo po úpravě jednoho produktu dokázali rovnou definovat kategorické parametry v rámci kategorického stromu a vyplnit tak parametry jednodušeji. Někteří uživatelé si této možnosti všimli až později.

Při testováni jsem zanalyzoval i nějaké chyby. Mezi nejčastějšími chyby, s kterými jsem se při uživatelském testování setkal, byly stylistického typu.

▶ **Příklad 5.1.** Uživatel se při posouvání dostal do části stránky, kde tlačítko pro uložení nebo validační chyby formuláře nebyly viditelné, což vedlo k mylnému dojmu, že jsou data uložená a mají úkol hotový.

▶ **Příklad 5.2.** Uživatel přeskočil informativní text, který byl příliš malým fontem, protože nabýval dojmu, že není nijak důležitý.

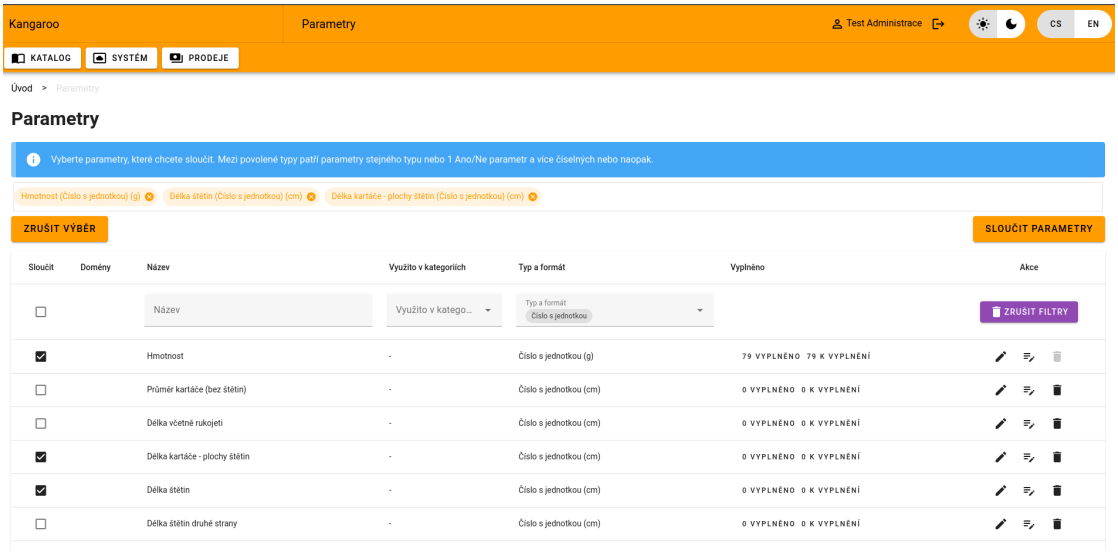
▶ **Příklad 5.3.** Uživateli splynulo tlačítko s textem, když bylo znepřístupněno, protože černá na tmavě oranžové je špatně čitelná.

Dalším zajímavým poznatkem bylo zjištění, že při snaze o překlad parametrů a tvorbě kategorických parametrů vznikly nové požadavky. Uživatelé vyjádřili zájem o možnost přímého vyplnění překladů pro daný jazyk při úpravě parametru nebo filtru, pokud produkt či kategorie existuje pod některou cizojazyčnou doménu. Také vznikl požadavek na možnost vytvoření výchozího parametru pro kategorii s hodnotou, která by byla stejná pro všechny přidané produkty do kategorie, aby uživatel nemusel zadávat hodnotu zbytečně vícekrát, když bude vždy stejná.

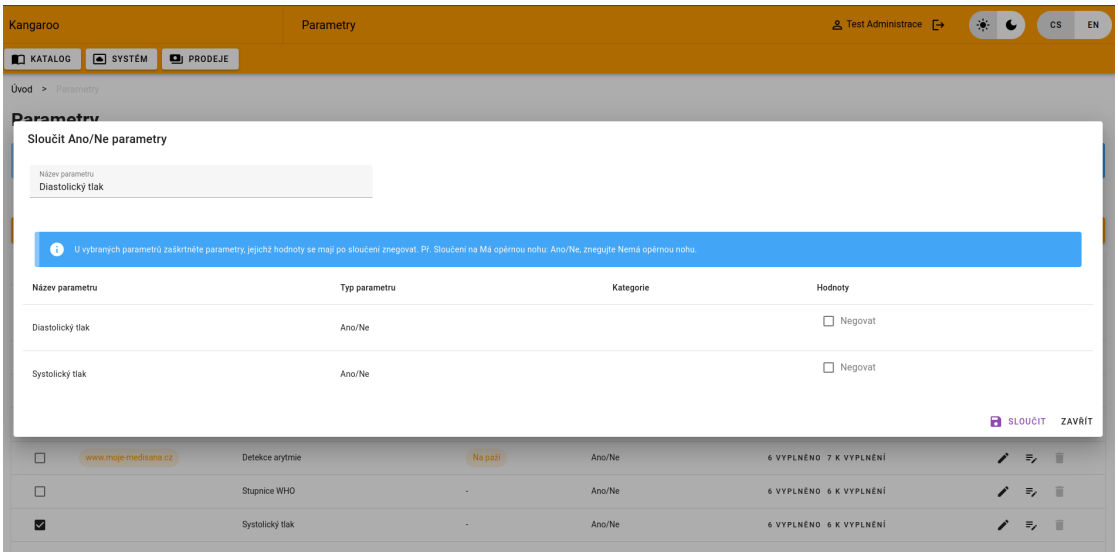
Okraj toho uživatelé z hlediska intuitivnosti hodnotili správu parametrů velmi kladně. Největší úspěch měly funkcionality:

- Možnost hromadného vyplňování hodnot parametrů napříč celým systémem.
- Možnost přehledné editace parametru společně s jeho hodnotami.
- Možnost posunu kategorického parametru po cestě ke kořeni kategorie.
- Možnost slučování různorodých parametrů s nastavováním hodnot pro konverzi.

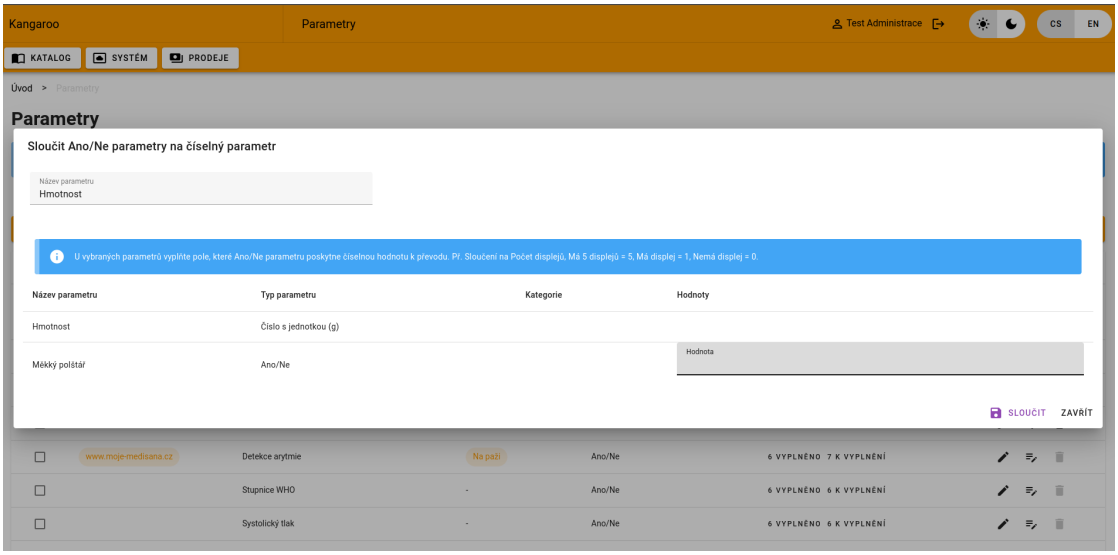
Na závěr kapitoly testování lze konstatovat, že přestože byly implementovány pouze základní testy a provedeno omezené uživatelské testování, tyto aktivity přispěly k identifikaci určitých nedostatků a problémů v aplikaci. Nedostatky týkající se vzhledu jsem se ještě pokusil vyřešit. Pro ukázku zde ponechávám snímky obrazovky, které nejlépe shrnují výsledky mé práce.



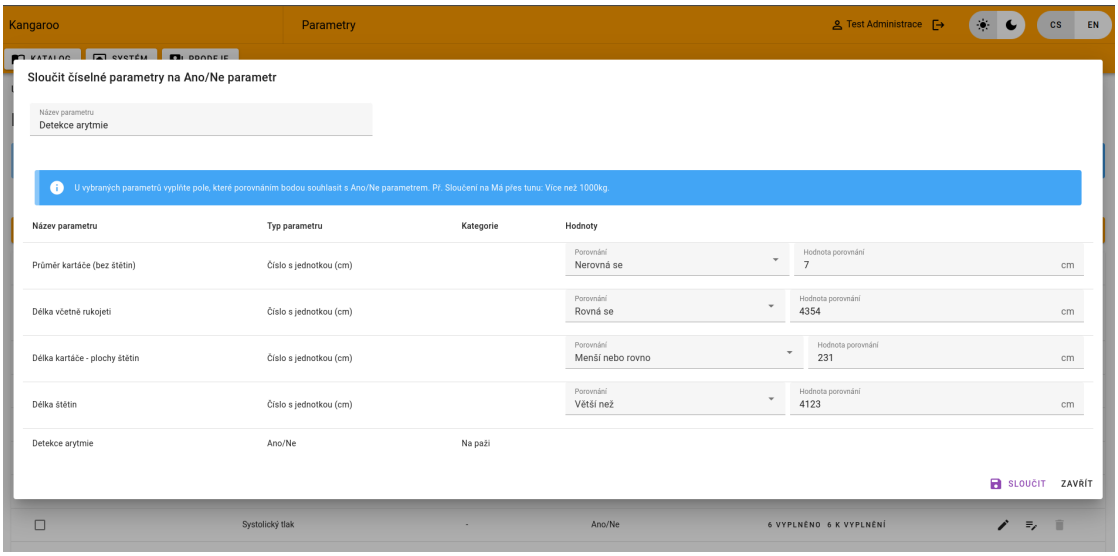
Obrázek 5.1 Tabulka parametrů



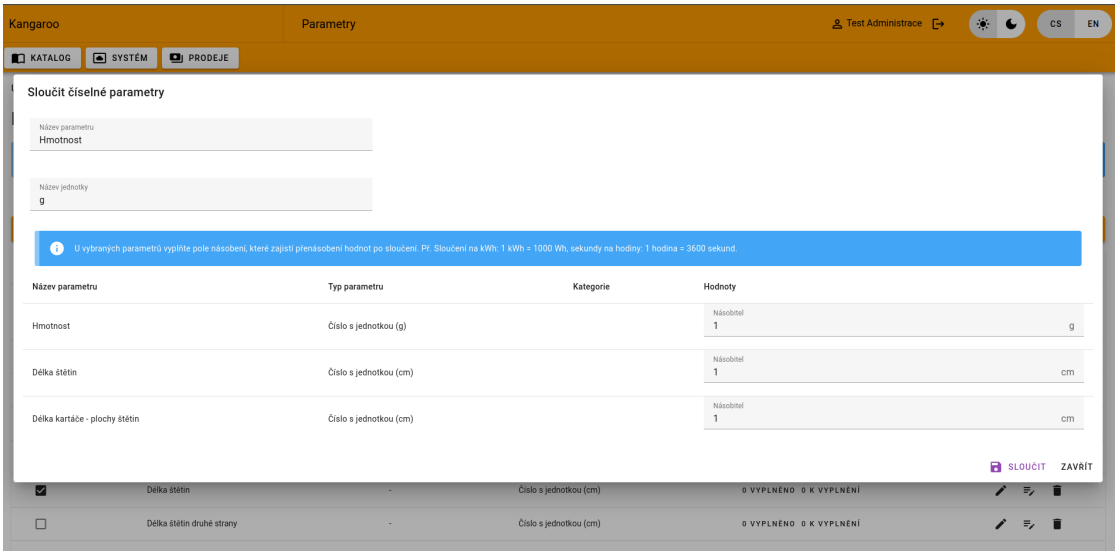
Obrázek 5.2 Sloučení booleovských parametrů



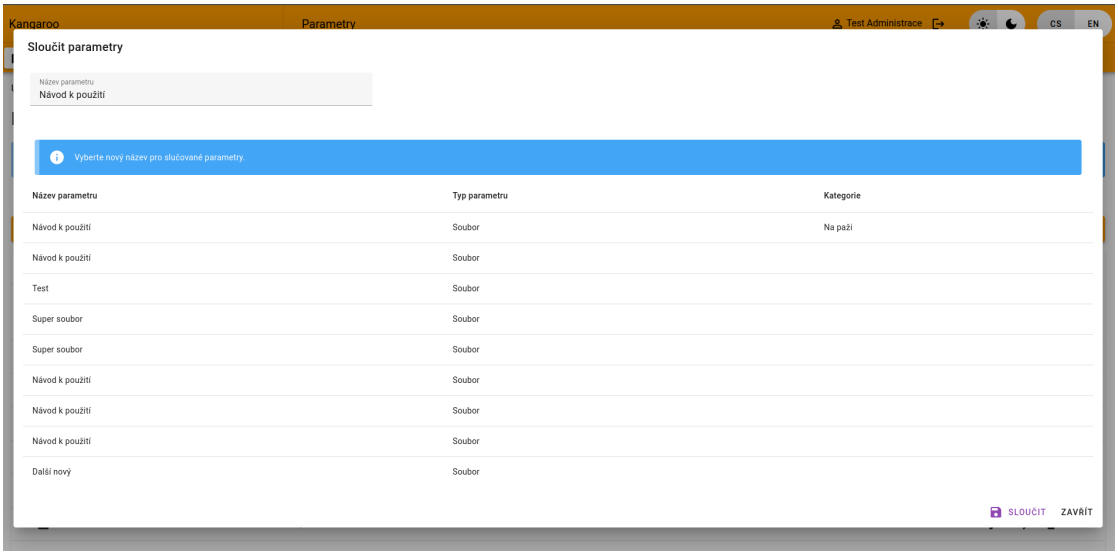
Obrázek 5.3 Sloučení ano/ne parametrů na číselný parametr



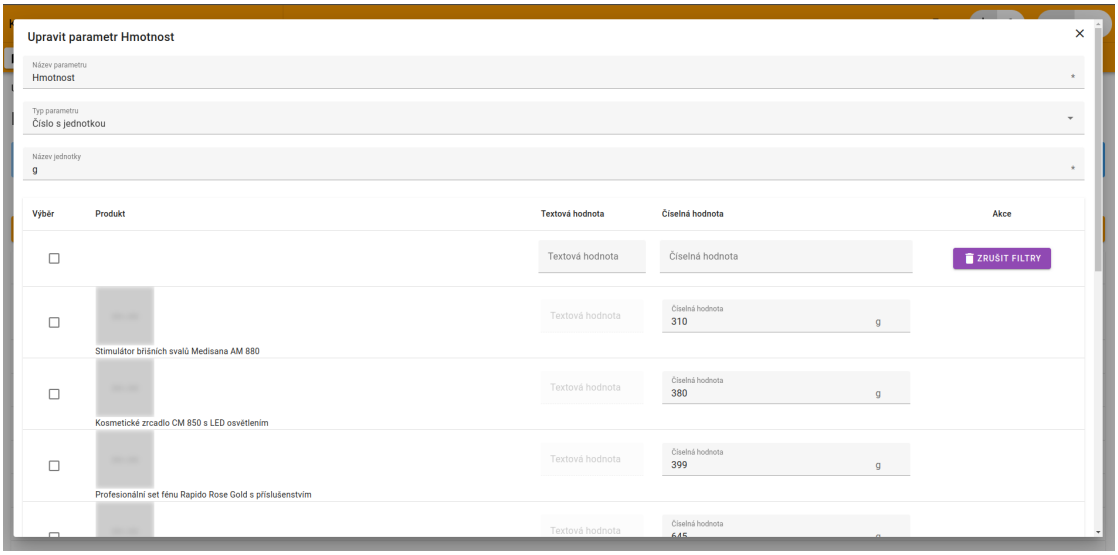
Obrázek 5.4 Sloučení číselných parametrů na ano/ne parametr



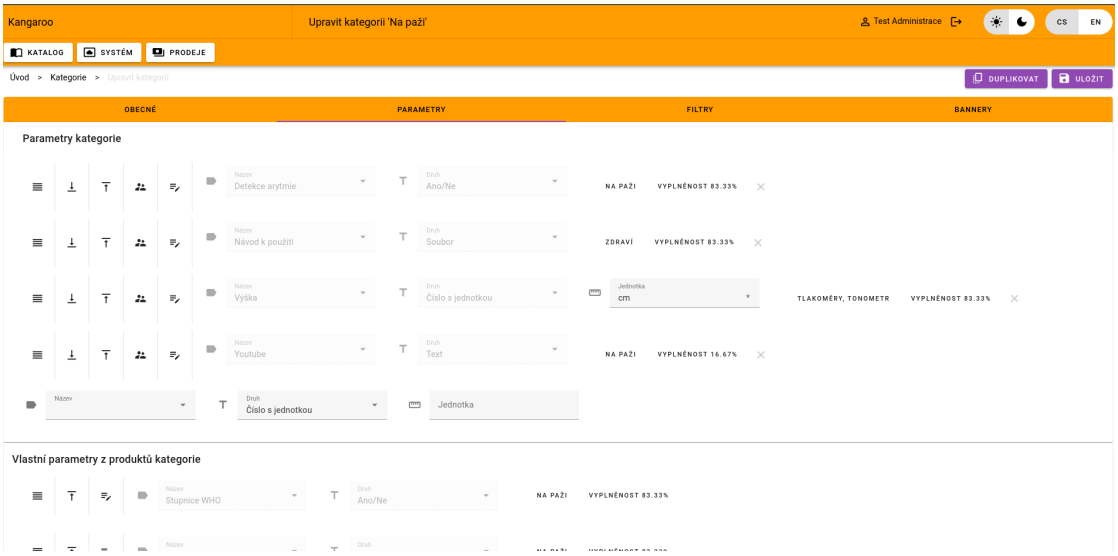
Obrázek 5.5 Sloučení číselných parametrů



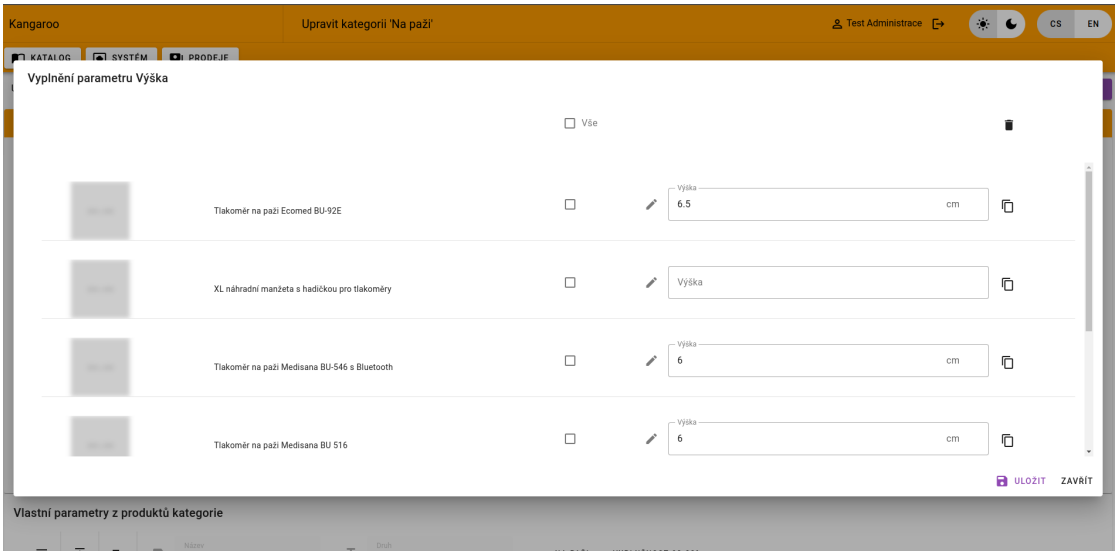
Obrázek 5.6 Sloučení textových/souborových parametrů



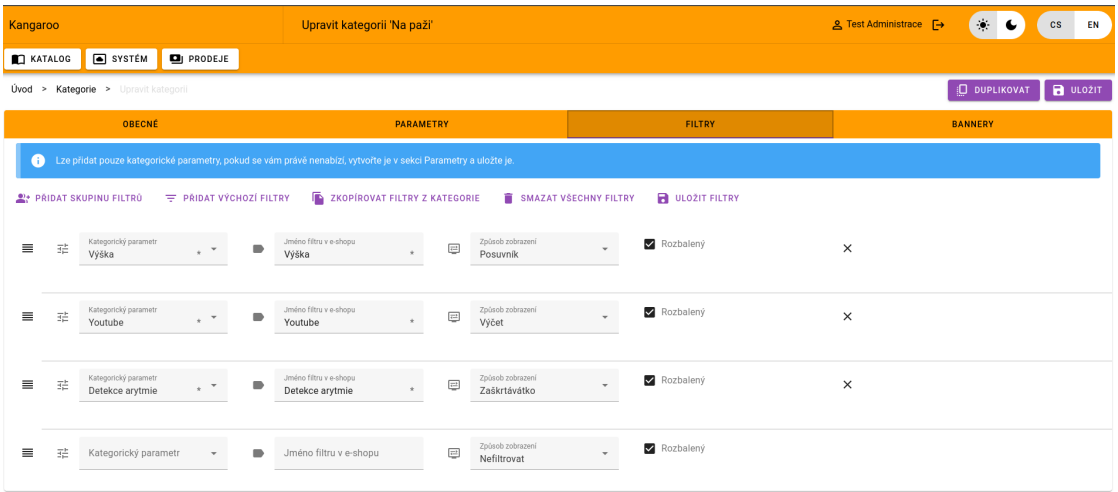
Obrázek 5.7 Formulář pro úpravu parametru



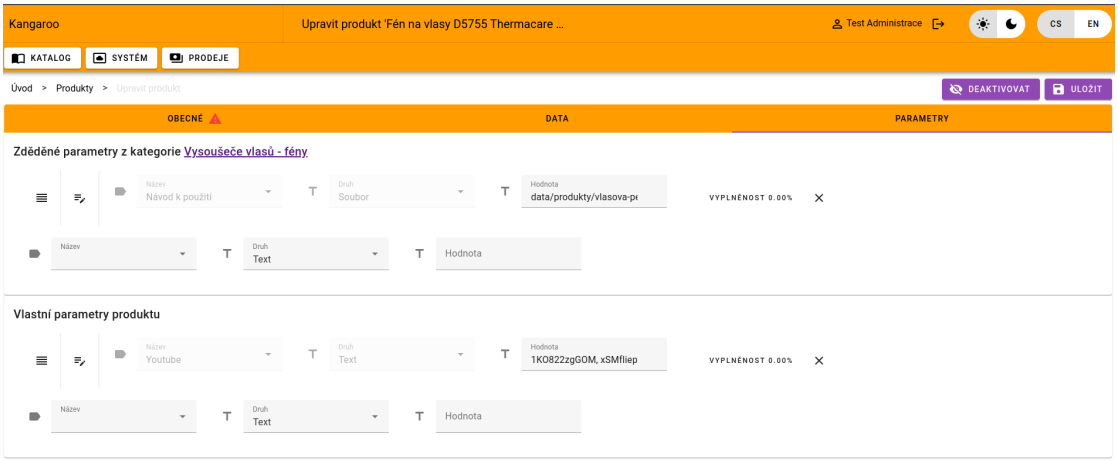
Obrázek 5.8 Formulář tvorby/úpravy kategorických parametrů



Obrázek 5.9 Vyplňovač hodnot parametru pro produkty



Obrázek 5.10 Formulář tvorby/úpravy filtrů kategorie



Obrázek 5.11 Formulář tvorby/úpravy parametrů produktu

Kapitola 6
Závěr

Závěr této práce přináší shrnutí dosažených výsledků a reflektuje na splnění stanovených cílů. Práce se zcela jistě povedla v dosažení svého hlavního cíle, kterým bylo vytvoření intuitivního prostředí pro správu parametrů napříč celým více doménovým e-shop administračním systémem propojeným se stávající databází. Podařilo se nám vyvinout nový systém, který má nahradit stávající administraci a přinést modernější a efektivnější prostředí pro správu e-shopu.

Je důležité zdůraznit, že se podařilo navrhnout a implementovat lepší řešení pro správu parametrů napříč celou administrací e-shopu. Tento nový přístup k administraci parametrů přináší větší efektivitu a uživatelskou přívětivost, což v konečném důsledku přináší zlepšení uživatelské zkušenosti a efektivity práce s e-shopem. Díky tomuto novému řešení je možné lépe a snadněji manipulovat s parametry produktů a provádět jejich správu v rámci administrace e-shopu.

I když se podařilo dosáhnout významného pokroku, zůstává ještě prostor pro vylepšení. Jednou z hlavních oblastí, která vyžaduje další práci, je řazení kategorických a produktových parametrů. V tomto směru je nutné provést další analýzu a implementovat vhodné řešení pro usnadnění procesu třídění.

Během práce se ukázalo, že další vylepšení je potřeba při vytváření kategorií. Zvláštní pozornost je třeba věnovat procesu vytváření, který se dosud neaplikuje po krocích, tudíž není v kartě filtrů možné získat vytvořené parametry, které vznikly při vytváření kategorie. Navíc je třeba splnit požadavky na překlady a výchozí parametry vzniklé při uživatelském testování. Tento krok pomůže zlepšit uživatelskou zkušenost a usnadnit proces správy parametrů.

Vzhledem k úspěchu této práce a identifikovaným oblastem vylepšení se nabízí prostor pro navazující práce. V budoucnosti je možné rozšířit aplikaci o další funkcionality nebo zdokonalit existující prvky. Je důležité pokračovat v monitorování a analýze uživatelské zpětné vazby, abychom mohli reagovat na potřeby uživatelů a neustále zlepšovat administraci e-shopu.

Celkově lze tedy říci, že práce přinesla hodnotné výsledky a otevřela dveře pro další rozvoj a optimalizaci správy e-shopu v souladu s požadavky a potřebami uživatelů.

Bibliografie

1. VUE.JS. *Introduction to Vue.js* [online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://vuejs.org/guide/introduction`.
2. SYMFONY. *What is Symfony?* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://symfony.com/what-is-symfony`.
3. MICROSOFT. *What is .NET?* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet`.
4. KOUBA, Alois. *E-shop*. 2024. České vysoké učení technické v Praze. V tuto chvíli ještě není dostupné.
5. DVOŘÁK, Martin. *Frontend administrace e-shopu*. 2022. Dostupné také z: `http://hdl.handle.net/10467/102105`. Bakalářská práce. České vysoké učení technické v Praze.
6. BABÁK, Jan. *Frontend administrace e-shopů*. 2022. Dostupné také z: `http://hdl.handle.net/10467/102225`. Bakalářská práce. České vysoké učení technické v Praze.
7. GITLAB. *About GitLab* [online]. GitLab, 2024. [cit. 2024-04-16]. Dostupné z: `https://about.gitlab.com/company/`.
8. SLACK. *What is Slack?* [Online]. Slack, 2024. [cit. 2024-04-16]. Dostupné z: `https://slack.com/help/articles/115004071768-What-is-Slack-`.
9. GOOGLE WORKSPACE. *Video Conferencing* [online]. Google Workspace, 2024. [cit. 2024-04-16]. Dostupné z: `https://workspace.google.com/resources/video-conferencing`.
10. REDMINE. *Redmine* [online]. Redmine, 2024. [cit. 2024-04-16]. Dostupné z: `https://www.redmine.org/`.
11. MLEJNEK, Jiří. *FIT ČVUT V PRAZE. Metodiky a agilní přístup* [online]. [cit. 2024-04-10]. Dostupné z: `https://moodle-vyuka.cvut.cz/pluginfile.php/780139/mod_resource/content/8/12.prednaska.pdf`. Dostupné po přihlášení.
12. OPENAPI INITIATIVE. *What is OpenAPI?* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://www.openapis.org/what-is-openapi`.
13. MLEJNEK, Jiří. *FIT ČVUT V PRAZE. Projektové řízen* [online]. [cit. 2024-04-10]. Dostupné z: `https://moodle-vyuka.cvut.cz/pluginfile.php/780135/mod_resource/content/8/11.prednaska.pdf`. Dostupné po přihlášení.
14. OPENCART. *OpenCart - Company Information* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://www.opencart.com/index.php?route=cms/company`.
15. JETBRAINS. *IntelliJ DataGrip* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: `https://www.jetbrains.com/datagrip/?var=light`.

16. SEZNAM.CZ. *Nápověda - Zobrazení a párování položek: Parametry* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: https://napoveda.zbozi.cz/zobrazeni-a-parovani-polozek/parametry/.

17. MLEJNEK, Jiří. *FIT ČVUT V PRAZE. Analýza a sběr požadavků* [online]. [cit. 2024-04-10]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/780099/mod_resource/content/10/03.prednaska.pdf. Dostupné po přihlášení.

18. Software Quality Attributes. In: *Essential Software Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 23–39. ISBN 978-3-540-28714-8. Dostupné z DOI: 10.1007/3-540-28714-0_3.

19. BOGARD JIMMY. *Automapper Documentation* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://docs.automapper.org/en/stable/Getting-started.html.

20. MLEJNEK, Jiří. *FIT ČVUT V PRAZE. Architektonické vzory* [online]. [cit. 2024-04-10]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/780114/mod_resource/content/6/06.prednaska.pdf. Dostupné po přihlášení.

21. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dostupné také z: https://www.epai-ict.ch/nexus/repository/course-material/m133/fielding_dissertation.pdf. DISSERTATION. University of California, Irvine. Submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Information and Computer Science.

22. FIGMA. *About Figma* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: https://www.figma.com/about/.

23. HUNKA, Jiří. *Konzultace - Analýza, sběr funkčních požadavků*. 2024. Místo konání: Budova FIT, Thákurova 9, 160 00 Praha 6.

24. BUREAU INTERNATIONAL DES POIDS ET MESURES. *The International System of Units (SI)*. 2019. Dostupné také z: https://www.bipm.org/documents/20126/41483022/SI-Brochure-9.pdf/fcf090b2-04e6-88cc-1149-c3e029ad8232. Accessed on April 13, 2024.

25. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 639-2:1998(en) Codes for the representation of names of languages – Part 2: Alpha-3 code*. 1998. Dostupné také z: https://www.iso.org/obp/ui/en/#iso:std:iso:639:ed-2:v1:en. Accessed on April 13, 2024.

26. MATOUŠEK, Jan. *Konzultace o možných překladech*. 2024. Místo konání: Budova FIT, Thákurova 9, 160 00 Praha 6.

27. MICROSOFT. *Entity Framework Core Documentation* [Online]. 2022. [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/.

28. MICROSOFT. *LINQ (Language-Integrated Query) - C# Guide* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/linq/.

29. VUETIFY. *Why Vuetify? - Introduction to Vuetify* [Online]. 2024. [cit. 2024-04-16]. Dostupné z: https://vuetifyjs.com/en/introduction/why-vuetify/#what-is-vuetify3f.

30. VUE.JS. *What is Vuex?* [Online]. 2022. [cit. 2024-04-16]. Dostupné z: https://vuex.vuejs.org/.

31. VUE.JS CONTRIBUTORS. *Reactivity in Vue.js* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://vuejs.org/guide/essentials/reactivity-fundamentals.

32. VUE.JS CONTRIBUTORS. *Watchers in Vue.js* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://vuejs.org/guide/essentials/watchers.html.

33. VUE.JS CONTRIBUTORS. *Component Basics in Vue.js* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://vuejs.org/guide/essentials/component-basics.html.

34. ESLINT CONTRIBUTORS. *ESLint - Pluggable JavaScript linter* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://eslint.org/docs/latest/use/getting-started.

35. CONTRIBUTORS, Xunit. *xUnit.net* [online]. 2024. [cit. 2024-04-18]. Dostupné z: https://xunit.net/.

36. REIS, Sacha; METZGER, Andreas; POHL, Klaus. Integration Testing in Software Product Line Engineering: A Model-Based Technique. In: DWYER, Matthew B.; LOPES, Antónia (ed.). *Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 321–335. ISBN 978-3-540-71289-3.

37. PAUL, Raymond; TSAI, Wei-Tek; CHEN, Yinong; FAN, Chun; CAO, Zhibin; HUANG, Hai. End-to-End (E2E) Testing and Evaluation of High-Assurance Systems. In: *Springer Handbook of Engineering Statistics*. Ed. PHAM, Hoang. London: Springer London, 2006, s. 443–476. ISBN 978-1-84628-288-1. Dostupné z DOI: 10.1007/978-1-84628-288-1_24.

38. NIELSEN, Jakob; MOLICH, Rolf. *10 Usability Heuristics for User Interface Design* [online]. 2024. [cit. 2024-04-16]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

39. ANYDESK SOFTWARE GMBH. *AnyDesk* [online]. 2024. [cit. 2024-04-13]. Dostupné z: https://anydesk.com/en.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl.txt` ..... zdrojové kódy implementace - kde je najít
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF