FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Projektové řízení a realizace části správy administrace e-shopu
Student:	Vojtěch Beneš
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je ve spolupráci s Bc. Aloisem Koubou, Bc. Vítem Urbanem a Filipem Dubjakem realizovat a nasadit do použitelné formy více doménovou e-shop administraci navázanou na současnou databázi tak, aby byla paralelně provozovatelná s již existující administrací, kterou následně nahradí. Z důvodu velkého rozsahu a také aby byla zajištěna forma izolace od ostatních prací, je předmětem této konkrétní práce projektové řízení společně s realizací dílčí backend a frontend části. Práce klade důraz na budoucí snadnou rozšiřitelnost, údržbu a provoz včetně efektivního využití technologií a lidských zdrojů.

Postupujte v těchto krocích:
1. Navažte na analýzy řešené e-shop administrace a předešlých prací věnující se vývoji daného projektu v backend i frontend oblasti. Zanalyzujte i doposud využívané technologie a dříve aplikované postupy. Neopomeňte vhodné rozdělení práce daného projektu dílčím členům.
2. Na základě analýzy společně se zbytkem týmu navrhněte a zvolte vhodné technologie a společné postupy pro následnou realizaci.
3. Navrhněte vhodný celek frontend a backend vaší části administrace, kterou budete následně implementovat.
4. Návrhy a postupy řádně konzultujte se zadavatelem.
5. Implementujte vámi zvolenou dílčí část.
6. Navrhněte a realizujte vhodné formy testů Vaší zvolené dílčí části.
7. Otestujte výslednou realizaci a řádně zhodnoťte váš manažerský vliv na projektu.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 22. ledna 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

8. Zhodnoťte výsledné řešení a postupy, navrhněte úpravy do budoucna. Zajistěte budoucí manažerskou udržitelnost projektu.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 22. ledna 2024 v Praze.

Bakalářská práce

PROJEKTOVÉ ŘÍZENÍ A REALIZACE ČÁSTI SPRÁVY ADMINISTRACE E-SHOPU

Vojtěch Beneš

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

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu, panu Ing. Jiřímu Hunkovi, za vedení a konzultování této práce. Poté bych chtěl poděkovat Bc. Martinu Dvořákovi za konzultace a časté kontroly našich kódů. Dále bych chtěl poděkovat svým týmovým kolegům, jimiž jsou Bc. Alois Kouba, Bc. Vít Urban a Filip Dubják, za velice příjemnou spolupráci. V poslední řadě bych chtěl poděkovat své rodině a přátelům za podporu během studia a psaní této práce. Speciálně chci poděkovat těm, kteří byli tak ochotní a v některé fázi psaní si tuto práci přečetli a dali mi k ní zpětnou vazbu.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zabývá vývojem backendu administrace e-shopu za pomocí frameworku ASP.NET a jeho napojení na frontendovou část, která je vytvořena pomocí frameworku Vue.js. Součástí práce je analýza, která popisuje historii a stav administrace e-shopu, kterou provozuje firma Jagu s.r.o. Dále řeší týmový vývoj, architekturu software a technologie. Část návrh popisuje nastavení týmové práce, požadavky na systém a samotný návrh implementovaných částí systému. Podle tohoto návrhu je vyvinuto, otestováno a zdokumentováno API, jehož účelem je komunikace s frontendem, který je pro tuto komunikaci upraven. Na závěr je provedeno zhodnocení řešení, postupů a manažerského vlivu autora. Tato práce položila základy nového systému administrace e-shopu.

Klíčová slova administrace e-shopu, backend, API, C#, ASP.NET, frontend, Vue.js, agilní metodika

Abstract

This bachelor thesis deals with the development of the backend of the e-shop administration using the ASP.NET framework and its connection to the frontend part, which is created using the Vue.js framework. The thesis includes an analysis that describes the history and status of the administration of the e-shop operated by the company Jagu s.r.o. It also addresses team development, software architecture and technology. The design part describes the teamwork setup, system requirements and the actual design of the implemented parts of the system. According to this design, an API is developed, tested and documented to communicate with the frontend, which is adapted for this communication. Finally, an evaluation of the solutions, procedures and managerial influence of the author is made. This work has laid the foundations for a new e-shop administration system.

Keywords e-shop administration, backend, API, C#, ASP.NET, frontend, Vue.js, agile methodology

Seznam zkratek

API	Application Programming Interface
ASP.NET	Active Server Pages Network Enabled Technologies
CD	Continuous Deployment
CI	Continuous Integration
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DevOps	Development Operations
DevSecOps	Development Security Operations
DOM	Document Object Manager
EF	Entity Framework
FURPS	Functionality, Usability, Reliability, Performance, Supportability
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MSA	Modern Structured Analysis
MVC	Model View Controller
MVC-L	Model View Controller Language
ORM	Object–Relational Mapping
PHP	Hypertext Preprocessor
REST	Representational State Transfer
SDLC	Software Development Lifecycle
SOAP	Simple Object Access Protocol
SPA	Single Page Application
SQL	Structured Query Language
SWOT	Strengths, Weaknesses, Opportunities, Threats
UP	Unified Process
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCS	Version Control System
XML	Extensible Markup Language
XP	Extreme Programming
XPath	XML Path Language

Úvod

Internetové obchody nebo také e-shopy či e-commerce jsou v dnešní době internetu nedílnou součástí každodenní života. Když si člověk přeje zakoupit libovolnou věc, vyhledá ji na internetu a získá široký seznam internetových obchodů různých velikostí s daným produktem. Pro člověka je nakupování přes internet často pohodlnější než v klasickém kamenném obchodě, ať už kvůli času stráveném dopravou do obchodu, možné nedostupnosti daného produktu na prodejně nebo jiným důvodům.

Základním prvkem e-shopu, ať už vytvořeného pomocí předpřipraveného řešení nebo vlastnoručně naprogramovaného, je správa obsahu neboli administrativní rozhraní. V rozhraní je možné přidávat, odstraňovat a upravovat obsah a měnit různé vlastnosti a nastavení.

Tato práce se zabývá implementací nového systému administrace e-shopů pro firmu *Jagu s.r.o.* a použitím projektového řízení pro týmový vývoj.

Firma *Jagu s.r.o.* provozuje administrační informační systém pro vícero různých e-shopů na různých doménách, který je postaven na platformě OpenCart. Tato aplikace je v provozu již od roku 2009 a během jejího provozu a rozvoje se objevili problémy spojené s rozšiřitelností a udržitelností.

Na možné nové implementaci systému, který by nahradil původní administrační aplikaci, pracují studenti v rámci softwarových týmových projektů nebo v rámci svých bakalářských a diplomových prací. I tento projekt se však kvůli svojí povaze, problémům s kompatibilitou, častým změnám vývojářů a nedostatku času dostal do neideálního stavu, který omezuje možnosti rozvoje a údržby.

Bylo proto rozhodnuto, že bude vytvořena další nová implementace v moderním programovacím jazyce, který dokáže lépe a rychleji zvládat složité operace a operace pro vícero domén najednou. Tuto novou implementaci v rámci svých bakalářských a diplomových prací začne implementovat tým studentů: Vojtěch Beneš, Bc. Alois Kouba, Bc. Vít Urban a Filip Dubják. Tým v rámci projektu navrhne a naimplementuje celé větší části podle provedeného rozdělení celků administrace a dále také navrhne a upraví příslušné části v souvisejícím frontendového projektu.

Za pomocí projektového řízení bude určena vhodná metodika řízení projektu a vývoje softwaru, vybrán podpůrný software projektu, nastavena týmová komunikace a spolupráce, rozdělování úkolů a další součásti projektu. Cílem těchto rozhodnutí bude optimální a efektivní týmová práce v rámci projektu.

Tato práce navazuje na bakalářskou práci Bc. Aloise Kouby a na diplomovou práci Ing. Tomáše Hojka, které se zabývají návrhem a implementací backendové části předchozího nového systému a částečně také na bakalářské práce Bc. Jana Babáka a Bc. Martina Dvořáka, které se zabývají návrhem a implementací frontendové části, která bude procházet změnami.

Jedním z hlavních cílů této práce je naimplementovat dílčí komponenty systému, přiřazené dle provedeného rozdělení. Výsledné komponenty skládající se z backendových a frontendových částí by měli být použitelné a při jejich implementaci by se mělo dbát na efektivní využití techno-

logií a snadnou budoucí údržbu a rozšiřitelnost. Další hlavní cíl je zajištění budoucí manažerské udržitelnosti projektu.

Cíle práce

Teoretická část

Cílem teoretické části práce je analyzovat problematiku, současný stav projektu, využívané technologie a dříve aplikované postupy. Následně navrhnout nové vhodné technologie a postupy a porovnat je. Dále za pomocí projektové řízení popsat a analyzovat metodiky vývoje softwaru, týmovou spolupráci a možný podpůrný software.

Praktická část

Cílem praktické části práce je za pomocí projektového řízení zvolit metodiku vývoje softwaru, vhodně rozdělit části administračního systému členům týmu, vybrat podpůrný software projektu, nastavit fungování týmu a další součásti projektu. Dalším cílem je zvalidovat funkční požadavky z akademických prací, na které tato práce navazuje, zvážit jejich relevantnost a podle potřeby je upravit. Dále je cílem navrhnout a implementovat zvolené části aplikace jak v rámci backendového projektu, tak i v rámci frontendového projektu. Následujícím cílem je navrhnout a realizovat vhodné formy testů pro implementovanou dílčí část. Finální cílem je zhodnocení mého manažerského vlivu na projekt, výsledného řešení a zvolených postupů, navržení možných úpravy do budoucna a zajištění manažerské udržitelnosti projektu.

Přínos

Výsledky práce budou přínosné pro společnost *Jagu s.r.o.* a jejich zákazníky, pro které firma administraci e-shopu provozuje. Práce významně posune stav vývoje nového systému. Dále budou výsledky této práce přínosné pro současné i budoucí členy vývojového týmu, poněvadž jim umožní jednodušší porozumění problematice více doménové administrace e-shopu a orientaci v ní.

Kapitola 1
Analýza

Analýza je neopomenutelnou součástí vývoje software. Velkou mírou ovlivňuje všechny následující vývojové kroky. V této kapitole se zaměřím na analýzu aktuálního stavu systémů, možného budoucího stavu, práce v týmu, metodiky vývoje softwaru, architektury a technologií.

1.1 Aktuální stav

Firma *Jagu s.r.o.* aktuálně pro administraci e-shopů svých zákazníků provozuje systém, který je specifický tím, že dokáže spravovat jeden či více e-shopů najednou. Systém je implementován pomocí přepsané open source e-commerce platformy OpenCart. Tato platforma je monolitická aplikace založena na návrhovém vzoru MVC-L, vytvořena v jazyce PHP s MySQL databázi a HTML komponentami. Je navržena s ohledem na snadnou instalaci a používání. Podporuje také velké množství rozšíření, ať už oficiálních nebo komunitních, které umožňují přidání dalších funkcí dle potřeby (například platební brány nebo analytické služby). [1, 2]

Prvotní nasazení aktuálního systému (dále jen OpenCart) nastalo již v roce 2009. V průběhu času prošel velkým množstvím změn a vylepšení. Systém v době psaní této práce pracuje dobře jak po funkční, tak i po výkonnostní stránce, ale má zastaralé uživatelské rozhraní a kvůli své povaze, velké provázanosti částí, zastaralým technologiím a technickému dluhu se jeho udržovatelům špatně reaguje na změnové požadavky od zákazníků nebo zaučuje nové programátory. Jedním z konkrétních problémů je používaná databáze, která z historických důvodů neobsahuje cizí klíče a může být nepřehledná z důvodu nepoužívaných tabulek či sloupců. Cizí klíče se v době psaní této práce začali postupně ručně doplňovat, ale tento proces je teprve na začátku a ještě chvíli potrvá.

Kvůli zmíněním problémům a omezením vznikla myšlenka implementovat nový systém, který by OpenCart nahradil. Společnost *Jagu s.r.o.* tedy nabídla v roce 2022 tuto myšlenku jako možné téma bakalářských a diplomových prací pro studenty FIT ČVUT. Následně byl vytvořen tým 4 studentů a 2 projekty, mezi které byli studenti rozděleni. Jeden projekt s kódovým označením Porcupine se zaměřil na vývoj backendové části systému za pomocí jazyku PHP s Symphony frameworkem a druhý projekt pod jménem Kangaroo obsahuje implementaci frontendové části systému pomocí Javascript frameworku Vue.js. Vývoje těchto projektů se účastnili i další studenti a to v rámci předmětů *Softwarový týmový projekt 1* (BI-SP1) a *Softwarový týmový* (BI-SP2) nebo dalších bakalářských a diplomových prací. Předměty BI-SP1 a BI-SP2 mají pro jejich úspěšné splnění na studenty určité požadavky, které jim neumožňují se čistě věnovat vývoji. Dokumentace procesů a databázových tabulek se vývojářům předávala konzultacemi se zadavatelem a jeho týmem nebo v již dříve zpracovaných výstupech. Dokumentace API byla dostupná u projektu Porcupine, ale byla psána a upravována ručně, což je v pro projekt tohoto rozsahu zcela neideální.

Dokumentace tak obsahovala spoustu chyb anebo některé informace neobsahovala. [3, 4, 5]

Projekt Kangaroo je stále vyvíjen, ale v rámci udržení udržitelného stavu musela být jeho velká část zmigrována na novější hlavní (major) verzi použitého frameworku. Tento proces není ještě zcela dokončen, protože některé dříve používané komponenty nemají v této verzi svůj plnohodnotný ekvivalent. Projekt Porcupine byl ukončen. Oba projekty trpěli na časté měnění vývojářů, nedostatečnou dokumentaci procesů a databázových tabulek a nedostatek času, což se projevilo zejména v technickém dluhu. Neideální byl také rozdělený způsob vývoje, kdy související frontendové a backendové části vyvíjeli různí lidé, což přinášelo větší nutnost týmové komunikace. Systém skládající se z těchto dvou projektů nebyl nasazen na žádné produkční prostředí kvůli některým chybějícím funkcím a nedostatečnému otestování celku.

Jako náhrada projektu Porcupine vzniknul projekt Pangolin, který s kolegy vyvíjíme a kterému se věnuje tato práce.

1.2 Budoucí stav

V rámci projektu Pangolin se za pomocí moderního programovacího jazyku vytvoří API pro projekt Kangaroo. Návrh API bude postaven na původním návrhu z projektu Porcupine, ale díky modernímu programovacímu jazyku s vhodným frameworkem by se měl výrazně zlepšit výkon aplikace (za předpokladu dodržení správných programovacích přístupů), což by mělo umožnit lepší provádění složitějších operací. Dále by se díky novému jazyku měla zlepšit čitelnost a udržitelnost kódu a také přístup k novým funkcionalitám. Během vývoje vznikne dokumentace API, implementovaných částí a použitých databázových tabulek.

Spojení projektů Pangolin a Kangaroo bude po důkladném otestování a nasazení fungovat souběžně s OpenCart systémem, který v budoucnosti nahradí. Díky novému vlastnímu systému se firma *Jagu s.r.o.* zbaví závislosti na platformě OpenCart, čímž získá lepší možnosti při úpravách a správě administračního systému dle zákaznických či vlastních požadavků.

1.3 Týmový vývoj

Většina softwarových projektů je tvořena v týmu spolupracujících lidí. Není tomu jinak ani u projektu Pangolin, který je vyvíjen jako součást této a dalších akademických prací. Je tedy třeba analyzovat různé faktory práce v týmu.

Týmovou práci můžeme chápat jako efektivní formu organizace práce, při které členové týmu spolupracují za účelem dosažení společného cíle. Práce v týmu tedy může být pro projekty velmi mocný nástroj, díky kterému je možné rychleji dosahovat cílů a překonávat překážky, ale může být také velkým zdrojem problémů a zdržení. Je proto nutné ji za pomocí projektového řízení správně nastavit podle našich potřeb a zkušeností. Ing. Jiří Mlejnek v přednášce *Úvod do softwarového inženýrství a týmového vývoje* z předmětu Softwarové inženýrství identifikuje jako hlavní problémy týmového vývoje, které nejsou čistě založeny na lidských vztazích a motivacích, komunikaci, sdílení informací a souborů a správu a rozdělování úkolů. K mitigaci těchto problémů a správnému fungování týmu nám můžou dopomoci různé metodiky nebo podpůrné programy. [6, 7]

1.3.1 Životní cyklus vývoje

„The software development lifecycle (SDLC) is the cost-effective and time-efficient process that development teams use to design and build high-quality software. The goal of SDLC is to minimize project risks through forward planning so that software meets customer expectations during production and beyond. This methodology outlines a series of steps that divide the software development process into tasks you can assign, complete, and measure.“ [8]

Kroky SDLC se můžou lišit tým od týmu a také podle použité vývojové metodiky nebo modelu, ale jejich základní rozdělení je následující a je znázorněno na obrázku 1.1:

Plánování: obsahuje úkony jako jsou analýza nákladů a přínosů, analýza rizik, odhad zdrojů a jejich přidělování;

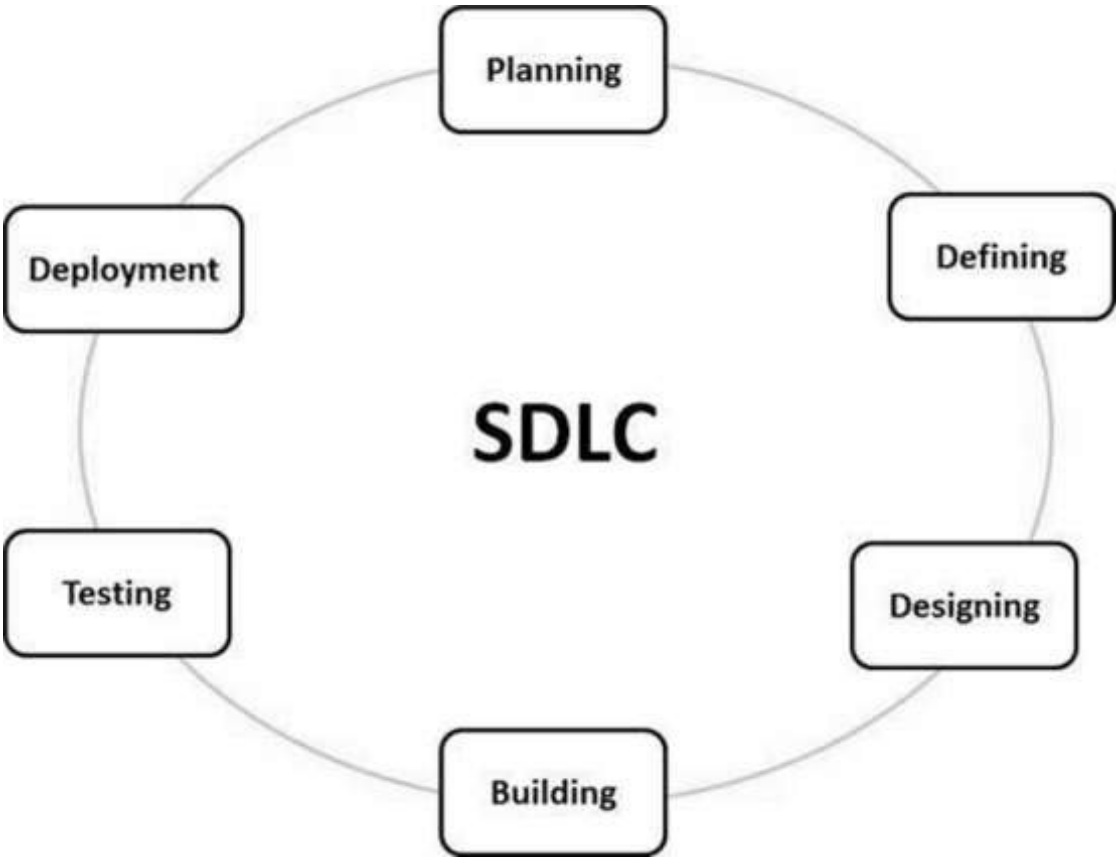
Definování: vývojový tým během tohoto kroku shromažďuje požadavky od zákazníka a vytváří z nich specifikaci softwaru;

Navrhování: v této fázi probíhá analýza požadavků a identifikace nejlepšího řešení k vytvoření zadaného softwaru. Součástí těchto úkonů je například výběr technologií, které budou použity, a vývojových nástrojů;

Stavění: během tohoto kroku probíhá hlavní vývoj software;

Testování: hlavním úkolem této fáze je testovaní vyvinutého softwaru, za pomocí automatických či manuálních testů, k nalezení možných chyb. Jisté formy testování by ale měly prováděny po každém kroku, pro zajištění validity výstupů;

Nasazení: cílem této fáze je nasazení softwaru do zákazníkova produkčního prostředí.



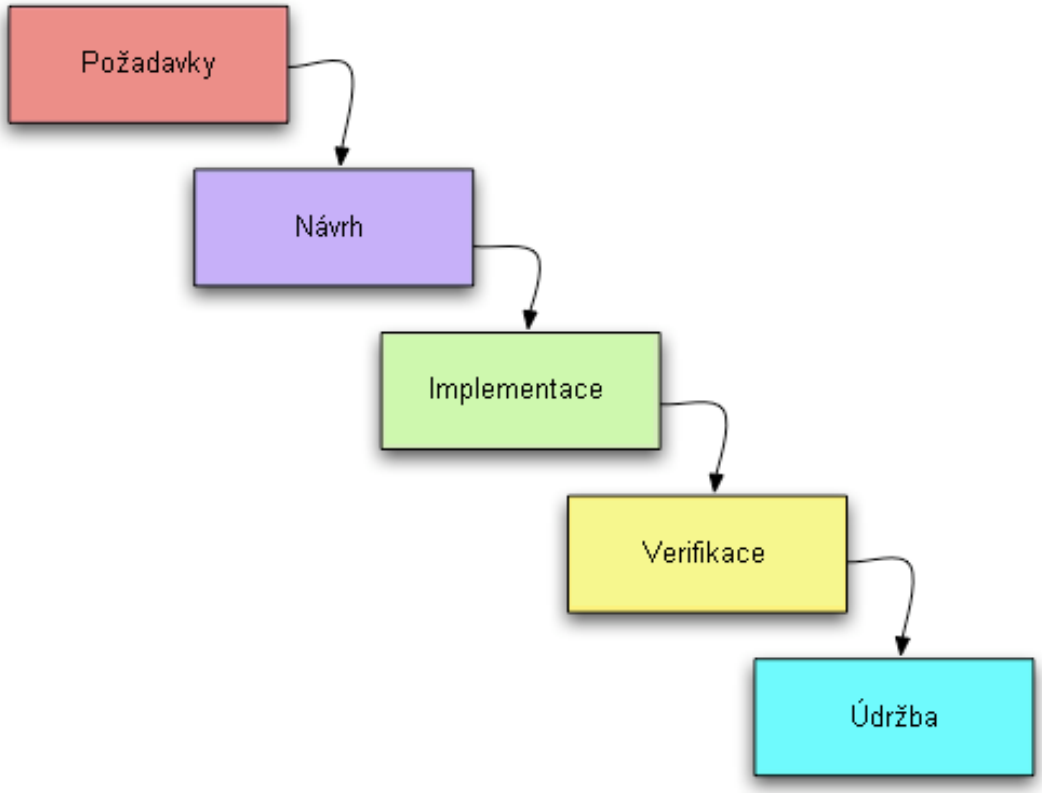
Obrázek 1.1 Fáze SDLC [9]

Modely SDLC jsou organizované a odzkoušené procesy, které slouží firmám k jednodušší implementaci SDLC do jejich práce. V následujícím textu vyjmenuji a popíšu některé populární modely:

Vodopád: v tomto modelu jsou fáze vývoje řazeny postupně tak, že každá fáze závisí na výsledcích té předchozí například jako na obrázku 1.2. Výhodou je, že po každé fázi jsou hmatatelné výsledky a vývoj je přímočarý. Nevýhodou je, že po ukončení fáze je velmi malý prostor pro změny, které by mohly být potřeba;

Iterativní: tento model pracuje v takzvaných iteracích, které si lze představit jako kratší vodopády. Každá iterace pracuje s podmnožinou požadavků a jejím cílem je vytvoření nové verze softwaru. Postupně s dalšími iteracemi se vybuduje výsledný software. Jedním detailnějším typem iterativního modelu je spirálový model, který je naznačen na obrázku 1.3;

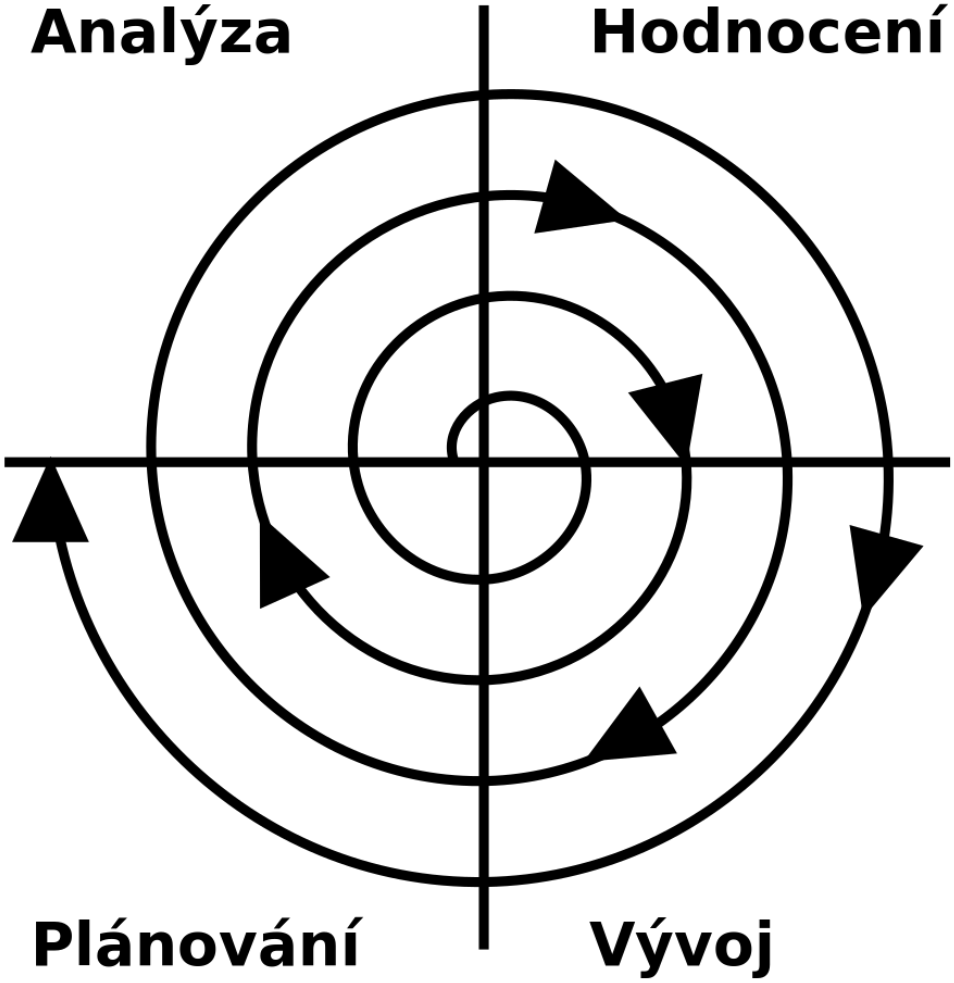
Agilní: agilní model je iterativní model, u kterého je délka iterace zkrácena na minimum. To umožňuje rychle vyvíjet malé části v každé iteraci a zároveň detailněji analyzovat požadavky a plánovat úpravy. Kvůli vysoké rychlosti vývoje je ale časté, že některé části, jako například dokumentace, jsou zanedbávány. [8, 9]



Obrázek 1.2 Vodopádový model [10]

1.3.2 Metodiky vývoje softwaru

Metodiky vývoje softwaru jsou sady doporučených praktik a postupů k řízení vývoje softwarového projektu. Pravidla a postupy můžou být obecné, velice detailní nebo se můžou zaměřovat pouze na určitou část například na implementaci.



Obrázek 1.3 Spirálový model [11]

V rámci předmětu Softwarové inženýrství Ing. Jiří Mlejnek ve své přednášce *Metodiky vývoje* stanovuje, že přínosem metodik je standardizace pracovních postupů, opakovatelnost a zjednodušení řízení projektu díky lepším definicím požadavků, sledování a kontrolování postupu a metrik. Nevýhodou používaní metodiky je přidaná pracnost a zvýšené časové požadavky kladené na členy vývojového týmu.

Existuje mnoho různých metodik, které je možné kombinovat, ať už celé nebo jen jejich části. Při výběru metodiky by se měl brát ohled na různé faktory jako jsou velikost projektu a týmu, složení týmu a další. V následujících podkapitolách popíšu dělení metodik. [3, 12]

1.3.2.1 Klasické

V klasických metodikách je zvykem před samotnou implementací provést důkladné obchodní modelování, sběr a analýzu požadavků, analýzu a design. Toto znamená, že tyto metodiky kladou velký důraz na tvorbu dokumentace, jak také zmiňuje Ing. Jiří Mlejnek v dříve zmíněné přednášce

Metodiky vývoje. Po úplném dokončení této přípravy se teprve začíná se samotnou implementací, která je už přímočará. Nevýhodou těchto metodik je jejich neohebnost, pracnost a administrativní náročnost, protože z analýz mají stanovené hodnoty času a množství dostupných zdrojů. Proto jsou například velice nevhodné v případech, kdy se může ještě měnit zadání či technologie. Mezi tyto metodiky patří například Unified Process (UP) a Modern Structured Analysis (MSA). [3, 12]

1.3.2.2 Agilní

Agilní metodiky vycházejí z Manifestu Agilního vývoje software [13], který vznikl již v roce 2001. Ten krom agilních principů deklaruje následující:

„Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním. Při této práci jsme dospěli k těmto hodnotám:“

- *Jednotlivci a interakce před procesy a nástroji*
- *Fungující software před vyčerpávající dokumentací*
- *Spolupráce se zákazníkem před vyjednáváním o smlouvě*
- *Reagování na změny před dodržováním plánu*

„Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.“ [13]

Z manifestu tedy vychází snaha se oprostit od náročnějších postupů vývoje v zájmu rychlosti vývoje a lepší možnosti reagovat na změny. Denní spolupráce je tedy pro tyto metodiky klíčová. Jednou z vlastností agilních metodik je také jejich velká volnost a možnost adaptace pro potřeby projektu. Příklady těchto metodik jsou Scrum, Kanban a Extrémní programování (XP). [12, 13, 14]

1.3.3 Podpůrný projektový software

Jak jsem již zmínil v úvodním textu kapitoly Týmový vývoj 1.3, problémy týmového vývoje nám mohou pomoci zmírnit nebo úplně vyřešit různé podpůrné projektové softwary. Těchto programů je velká řada, některé se buď cíleně specializují na určitý druh problému nebo obsahují řešení vícero problémů najednou. V následujících podkapitolách popíšu typy aplikací sloužící k mitigaci dříve zmíněných hlavních problémů týmového vývoje a zmíním některé jejich konkrétní příklady. Některé ze zmíněných nástrojů budou zvoleny pro používání při vývoji aplikace, které se týká tato práce.

1.3.3.1 Systémy správy verzí

Pro správu a sdílení zdrojových souborů projektů existují takzvané systémy správy verzí (VCS z anglického Version Control System). Tyto systémy zaznamenávají změny souborů v čase tak, aby se k nim člověk mohl později kdykoliv vrátit. Zaznamenané změny se nazývají verzemi. Tyto verze jsou pak dostupné i ostatním členům týmu. Další výhodou podle přednášky Ing. Jiřího Mlejnka *Úvod do DevOps* ze stejnojmenného předmětu je fakt, že tyto systémy omezují riziko ztráty dat, díky možnosti obnovení verze či ukládání do vzdáleného úložiště neboli repozitáře, a umožňují různé způsoby spolupráce. Vývojáři můžou buď pracovat na stejných souborech zároveň a jejich výstupy následně slučovat nebo může být soubor při úpravě zamknut a změny na něm může vykonávat pouze vývojář, který změnu započal. Systémy správy verzí se dělí na:

Centralizované: všechny verze jsou uchovány v jednom centrálním repozitáři a uživatelé mají lokálně pouze aktuální verze. Slabinou je fakt, že při výpadku centrální repozitáře je veškerá spolupráce ochromena;

Distribuované: na lokálním zařízení je zrcadlen celý vzdálený repozitář, systém je tak odolný proti úplnému ochromení při výpadku. [15]

Git Git je open source distribuovaný systém správy verzí. V rámci Gitu se do verzí ukládají snímky změněných souborů. Pokud nebyl soubor změněn, Git pouze odkáže na poslední změnu. Díky tomu je Git systém s řadou velmi výkonných nástrojů jako jsou například větve. Větve jsou pro vývoj projektů velmi užitečné, protože umožňují oddělení části programu ve vývoji od částí připravených k použití v produkčním prostředí. [15, 16]

Na Gitu jsou založeny platformy, které k tomuto systému přidávají grafické uživatelské rozhraní a rozšířené funkce. Jednou z takových je DevSecOps platforma GitLab. Ta umožňuje jednodušší týmovou spolupráci, zapojení kontinuální integrace (CI) a kontinuálního nasazení (CD) a spoustu dalších možností. GitLab lze hostovat na vlastní doméně. [17, 18]

1.3.3.2 Komunikační systémy

Komunikace je jedním z největších problémů týmového vývoje. Osobní komunikace tváří v tvář má spoustu výhod, ale ne vždy je možné ji ideálně realizovat. Komunikace za pomocí emailů je v některých případech vhodná, ale často je potřeba rychlejší a flexibilnější forma komunikace. K tomu mohou sloužit různé nástroj, které mají různé zaměření podle typu komunikace na internetu. Mezi tyto typy patří synchronní komunikace, která je definována bezprostředními reakcemi jako jsou například u videohovorů, a asynchronní komunikace, u které je běžné, že není potřeba okamžitá reakce. [14, 19]

Slack *„Slack is a messaging app for business that connects people to the information they need. By bringing people together to work as one unified team, Slack transforms the way organizations communicate.“* [20]

Jedním z příkladů nástrojů pro asynchronní komunikaci je Slack. Jedná se o aplikaci, která je dostupná skrz web nebo se dá nainstalovat na počítače či chytré telefony. Hlavní způsobem komunikace v ní, krom soukromých zpráv, jsou komunikační kanály, které mohou být různě organizované podle potřeb týmu. Nástroj umožňuje integraci různých dalších nástrojů a služeb do komunikačních kanálů za účelem zjednodušení a zefektivnění práce. Slack umožňuje i synchronní komunikaci za pomocí funkce Slack Huddle. Aplikaci lze používat zdarma, ale některé možnosti funkcí jsou při tomto používání omezeny. Například zobrazení zpráv v historii je možné jen pro posledních 90 dní. [20]

Google Meet Google Meet je nástroj pro synchronní komunikaci od společnosti Google. Umožňuje plánování a uskutečňování online schůzek a videohovorů na různých zařízeních. Důležitou funkcí je také sdílení obrazovky, které dovoluje účastníkům hovoru prezentovat svoji obrazovku ostatním. Každý, kdo má účet Google, má možnost uskutečnit videohovor až do délky 60 minut. [21]

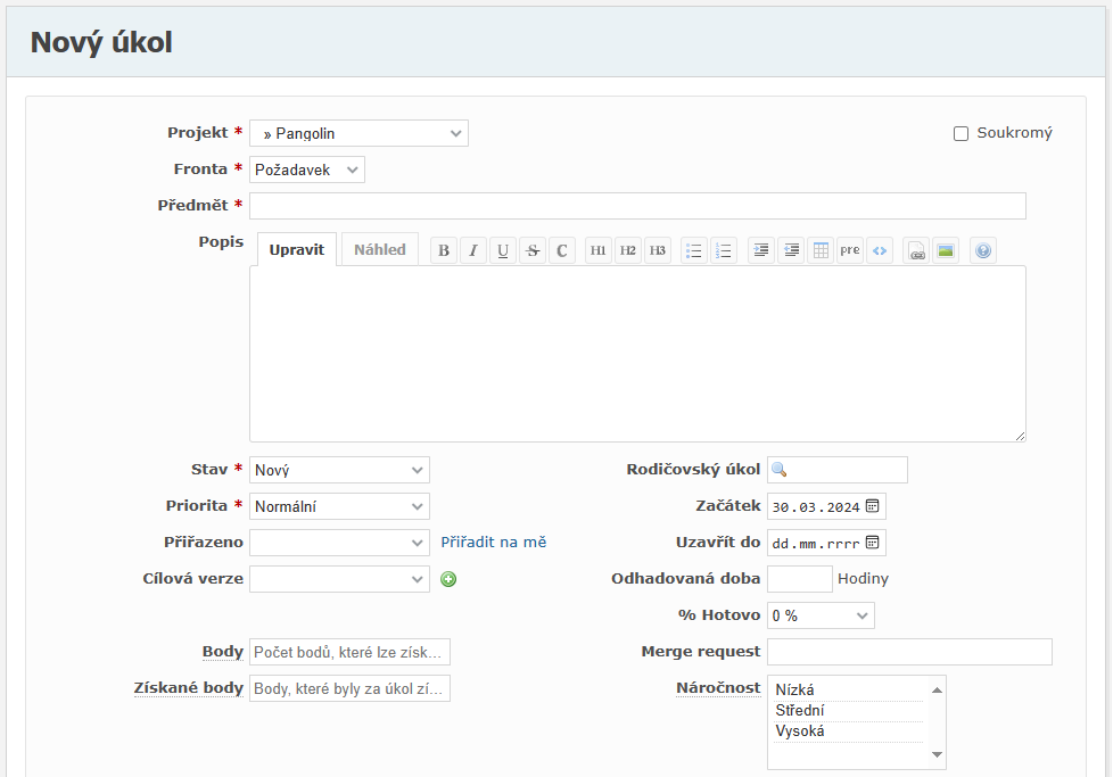
1.3.3.3 Aplikace pro správu úkolů a sdílení informací

Podpůrné aplikace pro správu úkolů se nazývají systémy řízení požadavků (anglicky Issue Tracking Systems). Jsou to informační systémy, které podle přednášky Ing. Jiřího Mlejnka *Úvod do DevOps* ze stejnojmenného předmětu pomáhají firmám v organizaci, rozdělování, kontrolování, řešení a archivaci požadavků. Požadavky jsou základním stavebním kamenem těchto systémů. Každý požadavek má zaevidované určité informace například identifikátor, název, popis, priorita řešení, stav a další. Systémy řízení požadavků nejsou jen pouhými úložišti úkolů, ale v dnešní době zastávají roli důležitého místa ke komunikaci a výměně informací, protože mohou sloužit nejenom samotnému vývojářskému týmu, ale i managementu či dokonce zákazníkům firmy. [22]

Ing. Jiří Mlejnek také v přednášce *Úvod do DevOps* zmiňuje vhodnost ukládání a udržování důležitých a relevantních informací k projektu on-line, tak aby se k nim mohli dostat všichni členové projektu. K tomu mohou sloužit různé editory značkovacích jazyků, buď jako samostatné nástroje nebo jako součásti komunikačních systému nebo systémů řízení požadavků.

Redmine *„Redmine is a flexible project management web application. Written using the Ruby on Rails framework, it is cross-platform and cross-database.“* [23]

Jednou z hlavních funkcí Redmine je flexibilní systém řízení požadavků, který je možné různě customizovat. Pro jednotlivé požadavky lze při vytváření nastavit základní informace, ale také související merge request nebo související úkoly jako například rodičovský úkol jak lze vidět na obrázku 1.4. Redmine také podporuje provoz vícero projektů najednou (s možností tvorby podprojektů) a správu přístupů za pomocí rolí. Mezi další funkce k vyzdvihnutí patří projektové wiki stránky či Ganttův diagram. [23]



Obrázek 1.4 Obrazovka tvorby požadavku v Redmine

GitLab Platformu GitLab jsem již částečně zmínil v sekci Git 1.3.3.1. Projekt je zde reprezentován pomocí Git repozitáře. Krom již zmíněných funkcionalit obsahuje také systém řízení požadavků pro projekt. Požadavky, zde nazývané issues, se v tomto systému dají různě organizovat pomocí štítků, iterací a milníků. K jednodušší organizaci zde slouží takzvaná tabule issues, kde lze monitorovat a upravovat aktuální stav issues. [17, 18]

Jira *„Jira is the #1 agile project management tool used by teams to plan, track, release and support world-class software with confidence. It is the single source of truth for your entire development lifecycle, empowering autonomous teams with the context to move quickly while staying connected to the greater business goal. Whether used to manage simple projects or to*

power your DevOps practices, makes it easy for teams to move work forward, stay aligned, and communicate in context.“ [24]

Jira má specifické funkce, které jsou hlavně přínosné pro agilní metodiky vývoje jako jsou Scrum či Kanban. Jednou z nich jsou je tabulové zobrazení, které pomáhá týmům s plánováním, vizualizací a organizací práce. Sloupce v rámci zobrazení představují kroky nebo stavy vedoucí k dokončení požadavku. Cesta stávající se z těchto kroků se v Jiře nazývá workflow. Workflow může být nastavené pro každý typy požadavku jinak, ať už se jedná o nahlášení chyby či požadavek na vývoj nové funkcionality. [24]

1.4 Architektura

Softwarová architektura je struktura komponent programu/systému, jejich vzájemné vztahy, principy a předpisy řídící jejich návrh a vývoj v průběhu času. Architektura by se měla vybírat podle provozních požadavků na systém. Tyto požadavky jsou například udržitelnost, znovupoužitelnost a rozšiřitelnost. Architektury můžeme dělit podle počtu vrstev, které mezi sebou navzájem spolupracují, na:

- monolitické,
- dvouvrstvé,
- třívrstvé,
- vícevrstvé. [25, 26]

1.4.1 Třívrstvá architektura

Třívrstvá architektura je jedna z nejpopulárnějších vícevrstvých softwarových architektur. Je vhodná jak pro enterprise tak i pro webové aplikace. Každá vrstva plní svůj jasně daný úkol a komunikuje pouze s předchozí či následující vrstvou. Její předností je oddělení vrstev od sebe, díky čemuž můžou běžet na různých infrastrukturách, můžou být vyvíjeny zároveň a můžou být znovuprožívány, vylepšovány a rozšiřovány podle potřeby bez nutnosti zasažení do ostatních vrstev. Vrstvy třívrstvé architektury jsou:

Prezentační vrstva: umožňuje komunikaci koncového uživatele s aplikací, stará se zobrazení a sběr dat;

Aplikační vrstva: obsahuje business logiku, procesy a validace, zajišťuje výpočty;

Datová vrstva: zajišťuje práci s daty pomocí operací a jejich persistenci. [27]

1.4.2 Model-View-Controller

Model-View-Controller (MVC) je architektonický vzor běžně používaný k implementaci uživatelského rozhraní a logiky aplikace. Jeho hlavním principem je oddělení business logiky aplikace a zobrazování dat, což vede k lepší přehlednosti a udržitelnosti zdrojových kódů aplikace. Dělí se na tři části:

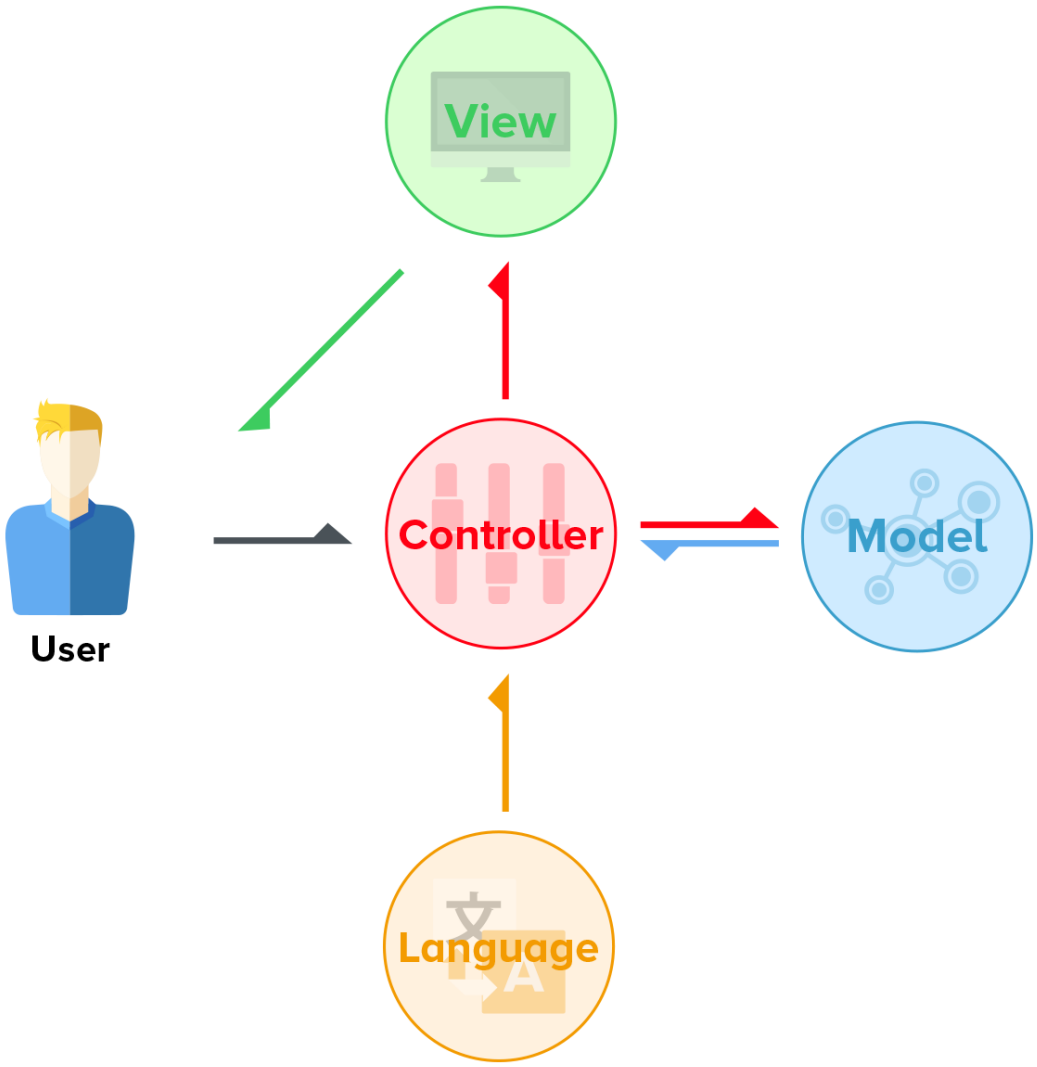
Model: definuje, která data aplikace zobrazí;

View: určuje způsob zobrazení dat;

Controller: obsahuje logiku, která upravuje view nebo model podle uživatelských vstupů. [28, 29]

Model-View-Controller-Language

Systém OpenCart využívá architektonický vzor MVC ještě obohacený o další komponentu Language – tedy jde o vzor Model-View-Controller-Language (MVC-L). Komponenta Language zajišťuje překlady textů podle připravených seznamů s texty. Obrázek 1.5 naznačuje způsob navázání této komponenty na původní vzor MVC. [30]



Obrázek 1.5 Schéma MVC-L Architektury [31]

1.5 Technologie

V této části analyzuji technologie, které byly využívány při vývoji projektů Porcupine a Kangaroo a porovnám je s vybranými technologiemi s obdobným zaměřením. Použité technologie v projektu určují jeho udržitelnost a přístupnost. Jejich volba je tedy velice důležitá.

1.5.1 API

Application Programming Interface (API) je rozhraní využívané při vývoji mobilních i webových aplikací. Slouží ke komunikaci a předávání dat mezi dvěma platformami za pomocí koncových bodů. API je více druhů, ale pro tuto práci je hlavní API pro komunikaci mezi backendem a frontendem tedy přesněji v rámci modelu klient–server. V následujících podkapitolách některé takové API zmíním a základně popíšu. [32]

1.5.1.1 SOAP

SOAP neboli Simple Object Access Protocol, je protokol pro zasílání zpráv založený na XML a HTTP. API postavené na tomto protokolu komunikuje pomocí XML dokumentů s jasně danou strukturou značek, které popisují hlavičku a tělo zprávy. Jeho použití již není zcela tak běžné, protože během let byl nahrazen alternativami, které jsou jednodušší k použití a mají i další výhody jako třeba menší požadavky na síťové komunikace. Stále ale najde využití pro komplexní operace, s nimiž si jiné API nedokážou poradit. [33]

1.5.1.2 REST

Jedním z náhradníků SOAP je REST API (nebo také RESTful API), což je takové API, které dodržuje zásady Representational State Transfer (REST) architektury. Tyto zásady jsou následující:

Klient–Server: aplikace klientu a serveru jsou nezávislé;

Bezstavovost: požadavky musí vždy obsahovat všechny informace neboli server si nezaznamenává stav klienta;

Ukládání do mezipaměti: když je to možné, měly by být prostředky ukládány do mezipaměti jak v klientské tak i v serverové části;

Jednotné rozhraní: server by měl zpřístupnit prostředky jednotným a předvídatelným způsobem;

Vícevrstvý systém: prostředníci mezi klientem a serverem se neovlivňují a nelze poznat, zda se komunikuje přímo nebo skrz prostředníka;

Kód na vyžádání (Volitelné): umožnění posílání spustitelného kódu v odpovědích.

REST API komunikuje za pomocí CRUD operací, které slouží k vytváření, získávání, aktualizaci a mazání záznamů. Tyto operace jsou implementovány pomocí protokolu HTTP, což umožňuje v požadavcích a odpovědích přenášet krom samotných dat i další informace, jako například status kódy. Data je možné posílat v různých formátech jako jsou například JSON, XML, HTML a další. [32, 34]

1.5.2 Backend

Serverovou část pro API lze nazývat backendem, v souvislosti s API můžeme mluvit o tom, že backend zpracovává požadavky a vrací k nim odpovědi. Během zpracovávání požadavků backend komunikuje s databází, pro získání a úpravu relevantních dat, a dalšími službami či rozhraními. Ke komunikaci s databází se často používá objektově relační mapování (ORM), které umožňuje mapovat třídy na tabulky v databázi a zařizuje převody datových typů. [35, 36]

V backendovém projektu Porcupine byl použit jazyk PHP s frameworkem Symfony. Tyto technologie v následujících podkapitolách popíšu a následně je porovnám s alternativními technologiemi – jazykem C# a platformou .NET.

1.5.2.1 Symfony

„Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community.“ [37]

Symfony je open source PHP framework, který je vyvíjen již od roku 2005 a pyšní se svojí komunitou a dokumentací. Je postaven na již zmíněné MVC architektuře a navržen tak, aby ji využil co nejlépe, ale také zachoval jednoduchost použití. Krom vlastních komponent obsahuje také další PHP frameworky, které se specializují na určité části vývoje. Patří mezi ně například projekt Doctrine, jehož hlavními funkcemi jsou ORM a sjednocení komunikace s databázemi, a PHPUnit, který slouží k automatickému testovaní aplikací. [37, 38, 39, 40]

1.5.2.2 .NET

„.NET je bezplatná open sourcová vývojářská platforma pro vytváření mnoha druhů aplikací. Může spouštět programy napsané ve více jazycích, přičemž jazyk C# je nejoblíbenější. Spoléhá na vysoce výkonný modul runtime používaný v produkčním prostředí mnoha vysoce škálovatelnými aplikacemi.“ [41]

.NET je spravován společností Microsoft a skládá se z velkého množství komponent. Jednou z nich je open source framework ASP.NET Core, který slouží k vytváření celých webových aplikací či API. Díky svému modernímu přístupu, vysokému výkonu a podpoře vícero platforem je velice populární. [41, 42]

Součástí .NET je i moderní ORM zvané Entity Framework (EF), které umožňuje v aplikacích tvořit kvalitní datové vrstvy nebo schémata databázových migrací. [43]

1.5.3 Frontend

Pod termínem frontend je hlavně myšleno grafické uživatelské rozhraní (GUI), se kterým uživatelé přímo interagují. Součástí rozhraní jsou různá navigační menu, designové prvky, tlačítka a další. Toto rozhraní získává a zobrazuje data z backendu, které získá pomocí API. [35]

Frontendový projekt Kangaroo využívá jazyku TypeScript s frameworkem Vue.js k tvorbě Single-Page aplikace (SPA) neboli aplikace, která obsahuje pouze jednu stránku, která se mění dle potřeby. [44] Použitý framework popíšu a následně popíšu i jeho možnou alternativu – framework Blazor, který je součástí platformy .NET.

1.5.3.1 Vue.js

„Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative, component-based programming model that helps you efficiently develop user interfaces of any complexity.“ [45]

Open source framework Vue.js byl vytvořen v roce 2014 a je známý pro svoji jednoduchost a flexibilitu. Také má rozsáhlou a detailní dokumentaci. Ve Vue jsou základním stavebním kamenem komponenty, tedy celé ovládací a zobrazovací funkční prvky. Mezi další výhody tohoto framework patří například virtuální DOM, který umožňuje rychlejší aktualizace. [45, 46, 47]

1.5.3.2 Blazor

„Blazor is a modern front-end web framework based on HTML, CSS, and C# that helps you build web apps faster. With Blazor, build web apps using reusable components that can be run from both the client and the server so that you can deliver great web experiences.“ [48]

Blazor je součástí platformy .NET. Blazor, stejně jako Vue.js, využívá komponent budování stránek a aplikací. Komponenty se zobrazují do reprezentace DOM, která je uložena v paměti, se kterou je možné aktualizovat GUI efektivně. Aplikace v tomto frameworku se mohou vykreslovat

buď na straně serveru, což umožňuje přeskočit tvorbu API, nebo na straně klienta za pomocí WebAssembly. [48, 49]

1.6 Závěr analýzy

V rámci této kapitoly jsem detailně popsal aktuální stav systému administrace e-shopů od firmy *Jagu s.r.o.*, pokusy o jeho nahrazení a i zamýšlený budoucí stav. Následně jsem v části Týmový vývoj 1.3 popsal základy a problémy práce v týmu, na což jsem navázal analýzou životního cyklu vývoje, metodik vývoje softwaru a podpůrných projektových softwarů. Nakonec jsem popsal možné architektury aplikací a technologie, z nichž některé byly již využité v dřívějších projektech administrace e-shopů. Některé analyzované koncepty a technologie využiji v následujících kapitolách této práce.

Kapitola 2
Návrh

V této kapitole popíšu a odůvodním učiněná rozhodnutí v projektu Pangolin, mezi které například patří výběr technologií, nastavení týmových procesů, rozdělení částí k implementaci mezi členy týmu a další. Dále zvaliduji a podle potřeby upravím funkční požadavky z prací, na které navazuji, které odpovídají přidělené části. Následně tyto části popíšu a dle funkčních požadavků provedu jejich návrh.

2.1 Nastavení projektu

Jak jsem již zmínil v části Aktuální stav 1.1 vývoj původního backendového projektu Porcupine byl zastaven a jako náhrada vznikl projekt Pangolin. Vývojový tým, který se ujme prvního vývoje tohoto projektu se kromě mé osoby skládá ještě z Bc. Aloise Kouby, Bc. Víta Urbana a Filipa Dubjáka. S vývojovým týmem spolupracují jako zadavatelé zástupci firmy *Jagu s.r.o.*, Ing. Jiří Hunka a Bc. Martin Dvořák. Pro tento nový projekt a vývoj v něm bylo nutné rozhodnout a nastavit několik věcí, které popíšu v následujících podkapitolách.

2.1.1 Výběr technologií

První otázkou v novém projektu byl výběr technologie, na které bude tento backendový projekt vystavěn. Jedním z cílů nového projektu je snaha o mitigaci problémů, které jsem popsal v části Aktuální stav 1.1. Je předpokládáno, že i projekt Pangolin bude nadále vyvíjen studenty v rámci akademických prací či předmětů BI-SP1 a BI-SP2. Výběr technologií by měl tedy s tímto faktem počítat, aby studenti nemuseli trávit příliš velké množství času na seznamování s nimi. Toto mohlo být faktorem v projektu Porcupine, protože jazyk PHP je v některých svých částech velmi specifický a například debuggovací nástroje je nutné nastavovat extra.

Reprezentanty firmy *Jagu s.r.o.* byla navržena platforma .NET, konkrétně framework pro tvorbu webových aplikací ASP.NET Core. Tento framework a zmíněnou platformu jsem již popsal v sekci .NET 1.5.2.2. Firma využívá tohoto frameworku v projektu přepisování skladového systému, dokonce má pro jisté funkčnosti vyvinuté vlastní rozšiřující balíčky a je s ním velmi spokojena. Některými výhodami tohoto frameworku je objektově orientovaný jazyk C#, který je dost podobný jiným jazykům, a zaštítění velkou organizací Microsoft. Ta se pyšní i vysokým výkonem ASP.NET Core dle srovnávacích testů. [42]

Vzhledem ke zmíněným výhodám, interním firemním balíčkům a dobré možnosti konzultovat vývoj s vývojáři *Jagu s.r.o.* byl pro projekt Pangolin zvolen framework ASP.NET Core. Za pomoci tohoto frameworku vznikne backendová aplikace vystavující API.

Vzhledem k tomuto rozhodnutí se objevila otázka, zda nevyužít i další součásti .NET platformy, a to webového frameworku Blazor, pro nahrazení frontendového projektu Kangaroo. Tato možnost byla ale odmítnuta z několika důvodů. Prvním důvodem byla přidaná náročnost, kvůli nutnosti úprav návrhu a vývoje od začátku. Dalším důvodem je fakt, že některé další aplikace firmy *Jagu s.r.o.* již framework Vue.js využívají a také používají stejný designový standard. Kvůli těmto důvodům bylo rozhodnuto, že projekt Kangaroo zůstane zachován a pouze se upraví některé části dle potřeby.

S rozhodnutími změny vývojového jazyka jsem spokojen. Sice mám s PHP a frameworkem Symfony jisté zkušenosti, díky kterým bych se lépe a rychleji zorientoval, ale práce s C# a ASP.NET je velmi příjemná a hodnotná zkušenost. Druhé rozhodnutí je také validní, ať už z důvodu přidané náročnosti nebo horší možnosti konzultovat.

2.1.2 Rozdělení implementace

Systém administrace e-shopů se skládá z velkého množství částí o různých rozsazích. Části se zabývají například produkty na e-shopech, ale také třeba zákazníky nebo daňovými třídami. Mezi hlavní části patří následující:

- výrobci,
- kategorie,
- produkty,
- objednávky.

Tyto vyjmenované části přestavují základ administrace e-shopu, který je v novém projektu třeba naimplementovat. Souvisí s nimi některé společné části, ale ty byly implementovány dle aktuálních možností vývojářů. Rozdělení bylo provedeno společně se zadavatelem a za přihlédnutí k analýze pomocí SWOT, kterou popíšu v následující sekci.

2.1.2.1 SWOT

SWOT (Strengths, Weaknesses, Opportunities, Threats) analýza je metoda, která popisuje vnější a vnitřní faktory ovlivňující úspěch projektu či firmy. Vznikla již v 60. letech 20. století a lze ji využít k analýze a usnadnění rozhodování pro organizace, ale i pro jednotlivce. Mezi vnitřní faktory patří silné a slabé stránky neboli vlastnosti a nedostatky analyzovaného subjektu. Vnější faktory, které jsou rozděleny na příležitosti a hrozby, jsou události a vlivy, které mohou subjektu být přínosem nebo na něj mohou mít negativní dopad. [50, 51]

Pro každého člena týmu bude provedena personální SWOT analýza za pomocí následujících jednoduchých otázek:

- Jaké jsou tvé silné stránky?
- Jaké jsou tvé slabé stránky?
- Jaké vidíš příležitosti v projektu Pangolin? Co by ti mohla práce na projektu přinést?
- Jaké vidíš hrozby pro tento projekt? Co by tě mohlo ovlivnit při práci na něm?

Vojtěch Beneš Za své silné stránky považuji smysl pro zodpovědnost, zkušenosti z různých jazyků a technologií a také skutečnost, že jsem se již s danou problematikou setkal v předmětu BI-SP1. Mými slabými stránkami jsou sklony k prokrastinaci a také přílišná snaha o zjednodušování těžších problémů. Jako příležitost v tomto projektu shledávám možnost hlouběji se seznámit s novou technologií, která je velice populární a užitečná. Hrozbou pro mé fungování v projektu jsou moje pracovní povinnost, které ale půjdou pro potřeby projektu zredukovat. Analýza je realizována i ve formě tabulky 2.1.

Silné stránky	Slabé stránky
Menší zkušenosti s problémovou doménou Zkušenosti z různých technologií Zodpovědnost	Sklony k prokrastinaci Přílišná snaha o zjednodušování
Příležitosti	**Hrozby**
Seznámení s novou technologií	Pracovní povinnosti

Tabulka 2.1 SWOT Vojtěcha Beneše

Bc. Alois Kouba Silné stránky Aloise Kouby jsou jeho pracovitost, kreativita, ale také fakt, že lépe chápe problémovou doménu, protože ji již řešil ve své bakalářské práci a může se vyhnout některým problémům, na které narazil v minulosti. Mezi slabé stránky Alois řadí častou prokrastinaci a také to, že nemá žádné zkušenosti s vývojem frontendu. Libí se mu, že bude v rámci závěrečné práce řešit skutečný problém a že díky tomu získá mnoho nových zkušeností ohledně technologií, práce na větší projektu a možná také pracovní příležitosti. Za hrozby považuje špatnou komunikaci a podcenění náročnosti problematiky. Analýza je realizována i ve formě tabulky 2.2.

Silné stránky	Slabé stránky
Pracovitost Kreativita Větší porozumění problémové doméně	Sklony k prokrastinaci Žádné zkušenosti s vývojem frontendu
Příležitosti	**Hrozby**
Pracovní příležitosti Seznámení s novou technologií	Špatná komunikace Podcenění problematiky

Tabulka 2.2 SWOT Bc. Aloise Kouby

Bc. Vít Urban Vít Urban mezi své silné stránky řadil schopnost rychle se zorientovat v již vytvořeném kódu a přizpůsobit mu svůj styl implementace. Dále měl smysl pro dochvilnost a zodpovědnost, díky čemuž byl schopný reagovat na zprávy a požadavky velice rychle a navštěvovat všechny schůzky, což podle něj umožní efektivnější práci v týmu. Jako jednu ze svých slabých stránky určil neschopnost udržet dlouho pozornost a snadné rozptýlení, pokud se nachází v domácím prostředí. Další jeho slabou stránkou je tendence odkládat těžší problémy. Vít si myslí, že mu projekt může výrazně pomoci s pochopením a naučením jazyk C# s nímž neměl předchozí zkušenosti a s prací v týmu. Za hrozby považuje možnost nekonzistence projektu, což by způsobilo jeho nejednotvárnost a zmatečnost, a náročnější proces učení s technologiemi. Analýza je realizována i ve formě tabulky 2.3.

Filip Dubják Filip Dubják za svoje silné stránky považuje schopnost neustále se učit, zlepšovat se a také jeho zkušenosti různých dalších programovacích jazyků a technologií. Za svoji

Silné stránky	Slabé stránky
▬ Rychlá orientace v kódu ▬ Přizpůsobivost ▬ Dochvilnost a zodpovědnost	▬ Sklony k prokrastinaci ▬ Odkládání těžších problémů
Příležitosti	**Hrozby**
▬ Seznámení s novou technologií ▬ Práce v týmu	▬ Konzistence projektu ▬ Náročnější proces učení technologií

Tabulka 2.3 SWOT Bc. Víta Urbana

slabou stránku považuje to, že mu trvá příliš mnoho času seznámit a porozumět nezdokumentovanému kódu, který napsal někdo cizí, zvláště pokud je to v jazyce, s kterým teprve přichází do styku. Doufá, že díky projektu se seznámí a zdokonalí v používaných technologiích a také s doménou e-shopů a jejími problémy. Hrozbami jsou pro Filipa komplexnost problematiky a nutnost úspěšného zopakování předmětu pro pokračování na projektu. Analýza je realizována i ve formě tabulky 2.4.

Silné stránky	Slabé stránky
▬ Schopnost učit a zlepšovat se ▬ Zkušenosti z různých technologií	▬ Horší porozumění kódu bez dokumentace
Příležitosti	**Hrozby**
▬ Seznámení s doménou ▬ Seznámení s novou technologií	▬ Opakování předmětu ▬ Komplexnost problematiky

Tabulka 2.4 SWOT Filipa Dubjáka

2.1.2.2 Rozhodnutí

Na základě konzultace a dohody se zadavatelem a přihlédnutí k osobní SWOT analýze bylo rozdělení částí k implementaci provedeno následovně:

- Vzhledem k jeho předchozím zkušenostem s problémovou doménou a faktu, že se pro něj jedná o diplomovou práci, dostal Alois Kouba přidělenou jednu z větších částí systému, a to část produktů.
- Vítu Urbanovi, pro kterého se také jedná o diplomovou práci, byla přidělal druhá větší část systému, která se zabývá objednávkami.
- Vojtěch Beneš v rozdělení dostal dvě zbývající hlavní části, a to výrobci a kategorie.
- Filip Dubják, který do procesu rozdělování přišel později, dostal v rámci jeho bakalářské práce na starost komplexní řešení parametrů produktů, které jsou využívány ke filtraci a kategorizaci zboží.

Toto rozdělení mělo za snahu maximální izolaci mezi částmi. Implementace částí se skládala z implementace backendové části a napojení a upravení odpovídající frontendové části.

2.1.3 Nastavení týmové práce

Po rozdělení částí k implementaci bylo na řadě nastavení týmové práce, komunikace a výběr podpůrného softwaru.

Vzhledem k povaze a časovému omezení projektu byla zvolena agilní metodika vývoje, kterou jsem již popsal v části Agilní 1.3.2.2. Pro tento projekt je nutná častá komunikace se zadavatelem, spolupráce v týmu a dobrá možnost reakce na změny, proto je tato metodika velice vhodná. Nedostatek dokumentace během vývoje bude mitigován automaticky generovanou dokumentací API, vytvořením wiki stránky, která shromáždí dostupné informace o projektu, a samotnými akademickými pracemi.

Práce na projektu probíhala ve vývojových iteracích neboli sprintech o délce 1 týdne. Na začátku každého sprintu se na schůzce rozdělili úkoly na tuto iteraci. Na další schůzce se tým navzájem seznámil se stavem těchto úkolů a rozdělili se další úkoly.

Jak jsem již zmínil, hlavní schůzky se pojili se začátkem a koncem sprintů. Odehrávali se 1 týdně, buď osobně nebo za pomoci nástroje Google Meet. Další týmová komunikace probíhala v komunikační aplikaci Slack. Volbu těchto aplikací zdůvodním v následující části Vybraný podpůrný software 2.1.3.1. Dále bylo možné uskutečnit vedlejší schůzky například ohledně vývoje aplikací s frameworkem Vue.js.

Podle zvoleného frameworku ASP.NET a jazyku C# byly také nastaveny konvence kódu a pravidla pro pojmenování identifikátorů. V kódu byla například použita Pascalova notace pro jména tříd a metody a velbloudí notace pro argumenty metod a lokální proměnné. Cílem těchto pravidel byla snaha o udržení čitelnosti kódu a mitigace technického dluhu.

2.1.3.1 Vybraný podpůrný software

V této sekci zdůvodním výběr podpůrného softwaru pro projekt Pangolin. Vybrané softwary jsem již popsal v částech podkapitoly Podpůrný projektový software 1.3.3. V souvislosti s některými softwary také popíšu případné nastavení jejich používání.

Slack Pro asynchronní komunikaci byla vybrána aplikace Slack, zejména kvůli tomu, že byla již využívána pro předchozí projekty i pro předměty BI-SP1 a BI-SP2 a faktu, že ji firma *Jagu s.r.o.* hojně využívá. Pro komunikaci vývojového týmu vznikli dva nové kanály, veřejný kanál *#dotnet-api*, pro řešení otázek okolo platformy .NET, a privátní kanál *#dotnet-thesis*, pro komunikaci vývojového týmu.

Google Meet Nástroj Google Meet byl vybrán za účelem synchronních schůzek v případech, kdy se tým nemůže sejít osobně. Důvodem výběru je jeho jednoduchost a možnost plánování schůzek do kalendáře.

Redmine Issue tracker Redmine byl vybrán jednak z důvodu dřívějšího použití pro projekty Porcupine a Kangaroo, ale také protože firma *Jagu s.r.o.* provozuje aplikaci Timer2Ticket, která vznikla a je stále vyvíjena v rámci akademických prací a slouží k automatické či manuální synchronizaci časových záznamů, úkolů a projektů mezi Redmine a aplikacemi pro záznam času jako je například Toggl Track. Díky této možnosti synchronizace bylo možné zařadit záznam času stráveného na úkolu do nastavení týmové práce. Další součástí tohoto nastavení bylo založení hlavního úkolů, takzvaného epiku, pro každého člena vývojového týmu. Pod tento úkol byly vkládány všechny další úkoly pro daného člena týmu. Do informací úkolů byl zapsán sprint, ve kterém byl úkol vykonáván, a další údaje.

Git a GitLab Verzovací systém Git byl jednoznačná volba vzhledem k jeho popularitě, jeho výhodám a faktu, že se s ním všichni vývojáři z vývojového týmu již setkali. Stejně tak se setkali s vybranou platformou GitLab. Důvodem jejího zvolení je ten fakt, že firma *Jagu s.r.o.* provozuje vlastní instanci této platformy, která obsahuje dříve zmíněné starší projekty a vlastní balíčky firmy. Pro vývojový tým byl nastaven postup práce s větvemi. Ty před zakomponováním do hlavní větve byly podrobeny kontrole od některých ostatních vývojářů. Dále byla nastaveno pravidlo pro pojmenovávání větví a verzí za pomocí číselných identifikátorů odpovídajících úkolů z Redmine. Pro projekt byl v GitLabu nastaven nástroj CI, který umožňuje automatické spuštění testů a analýz po provedení operací v Gitu.

2.2 Požadavky

V přednášce *Analýza a sběr požadavků* v rámci předmětu Softwarové inženýrství Ing. Jiří Mlejnek stanovuje za cíle analýzy požadavků vymezit hranice a omezení systému, umožnění přesnějšího odhadu pracnosti a vyjasnění zadání se zákazníkem. Následně definuje základní rozdělení na funkční a nefunkční požadavky a rozdělení FURPS, které znamená následující:

- **F** (Functionality) – funkčnost
- **U** (Usability) – použitelnost
- **R** (Reliability) – spolehlivost
- **P** (Performance) – výkon
- **S** (Supportability) – rozšiřitelnost

Jak jsem již dříve zmínil, v této práci navazuji hlavně na bakalářskou práci Aloise Kouby a na diplomovou práci Tomáš Hojka. Vycházím tedy z funkčních a nefunkčních požadavků z těchto prací, které jsem zvalidoval, v případně nutnosti upravil a u některých také popsal způsob jejich splnění. Soustředil jsem se hlavně na požadavky, které jsou obecné nebo přímo souvisí s částmi systému, které mi byly přiděleny, což jsou výrobci a kategorie. Z tohoto důvodu jsou požadavky řešeny až v této kapitole, a ne v kapitole Analýza. Všechny vyjmenované požadavky rozdělím pomocí rozdělení FURPS a příslušně je označím.

2.2.1 Funkční požadavky

Funkční požadavky identifikují požadované funkčnosti systému a jeho chování.

Projekt Pangolin by měl kopírovat a rozšiřovat funkčnost předešlého projektu Porcupine, který podobně vycházel ze systému Opencart.

1. **Výrobci**
   - **F1 – Získání seznamu všech výrobců** [F]
   - **F2 – Získání detailu výrobce** [F]
   - **F3 – Přidání nového výrobce** [F]
   - **F4 – Upravení výrobce** [F]
   - **F5 – Smazání výrobce** [F]
   - **F6 – Úprava čísla řazení pro výrobce** [F]
   - **F7 – Upravení výrobce pro určitou doménu** [F]

2. **Kategorie**
   - **F8 – Získání seznamu všech kategorií** [F]
   - **F9 – Získání detailu kategorie** [F]
   - **F10 – Přidání nové kategorie** [F]
   - **F11 – Upravení kategorie** [F]
   - **F12 – Smazání kategorie** [F]
   - **F13 – Úprava čísla řazení pro kategorie** [F]
   - **F14 – Upravení kategorie pro určitou doménu** [F]
   - **F15 – Upravení rodičovské kategorie** [F]
3. **Parametry kategorií**
   - **F16 – Získání seznamu všech dostupných parametrů** [F]
   - **F17 – Získání seznamu všech parametrů pro zvolenou kategorii** [F]
   - **F18 – Úprava parametrů kategorie** [F]
4. **Filtry kategorií**
   - **F19 – Získání seznamu všech filtrů pro zvolenou kategorii** [F]
   - **F20 – Úprava filtrů kategorie** [F]
5. **Bannery kategorií**
   - **F19 – Získání seznamu všech bannerů pro zvolenou kategorii** [F]
   - **F20 – Úprava bannerů kategorie** [F]
6. **Externí kategorie**
   - **F21 – Manuální spouštění aktualizace externích kategorií** [F]
   - **F22 – Automatické a pravidelné spouštění aktualizace kategorií** [F]
   - **F23 – Propojování externích kategorií s kategoriemi** [F]
7. **F24 – Kontrola dat** [F]
   - Aplikace bude kontrolovat zadané data a v případě špatných vstupů řádně zabrání uložení dat a informuje uživatele.
8. **F25 – URL slug** [F]
   - Pro výrobce a kategorie bude aplikace pracovat také s řetězci znaků známými jako URL slug, které budou unikátní pro každou doménu a slouží jako unikátní identifikátory stránek a také jako faktor optimalizace vyhledávačů. [52] Systém bude nastavovat a kontrolovat tyto hodnoty podle určitých pravidel. Například při změně URL slugů se původní záznam nepřepisuje, ale vytvoří se nový, na který původní záznam pouze odkazuje.
9. **F26 – Výchozí doména** [F]
   - Systém získá možnost jednodušeji určovat výchozí doménu pro entity.
10. **F27 – Autentizace a autorizace** [F/S]
    - Administrace e-shopů může být ovládána pouze oprávněnými osobami. Aplikace bude umět autentizovat uživatele, kteří budou mít možnost provádět pouze ty akce, ke kterým byli autorizovaní. K autentizaci bude aplikace využívat službu Keycloak.

2.2.2 Nefunkční požadavky

Nefunkční požadavky popisují obecné vlastnosti nebo omezující podmínky systému.

1. **N1 – Architektura** [S]
   - *„Aplikace bude rozdělena na jednotlivé vrstvy podle vhodného návrhového vzoru.“* [3]
2. **N2 – Kompatibilita** [S]
   - *„Aplikace bude kompatibilní s původním systémem administrace. Je nutné zajistit, aby nově přidané funkce nenarušily paralelní běh staré administrace.“* [3] Tato kompatibilita také znamená nutnost použití již existující databáze, ve které, jak jsem zmiňoval v sekci Aktuální stav 1.1, například chybí cizí klíče.
3. **N3 – Technologie** [S]
   - *„V průběhu návrhu budou zohledněny technologie, které již firma Jagu, s. r. o., používá pro jiné vyvíjené informační systémy. Tento požadavek byl přijat pro snížení různorodosti používaných technologií, což obecně vede ke snazší správě portfólia používaných technologií.“* [4] Splnění tohoto požadavku jsem již popsal v části Výběr technologií 2.1.1.
4. **N4 – Uživatelské rozhraní** [U]
   - Aplikace bude mít uživatelské rozhraní. Tento požadavek je již splněn existencí frontendového projektu Kangaroo, který byl pouze částečně upraven.
5. **N5 – API** [U]
   - *„Aplikace bude vystavovat API rozhraní, pomocí kterého bude aplikace komunikovat s vnějším prostředím.“* [4]
6. **N6 – Dokumentace** [U]
   - *„Rozhraní aplikace bude řádně zdokumentováno.“* [3, 4]

2.3 Návrh výrobců

Výrobci jsou firmy, které vyrábějí a dodávají produkty nabízené v e-shopech. Databázové tabulky související s výrobci a jejich vztahy jsou znázorněny na obrázku 2.1. Základní význam daných tabulek je následující:

oc_manufacturer obsahuje základní informace o výrobcích.

oc_manufacturer_description obsahuje rozšířené informace o výrobcích.

oc_domain_override slouží k přepisování informací pro ostatní domény.

oc_language obsahuje seznam jazyků, které lze v systému použít.

oc_url_alias slouží ke správě URL slugů.

Pro splnění funkčních požadavků jsem vyšel z návrhu API výrobců z bakalářské práce Aloise Kouby a diplomové práce Tomáše Hojka, který jsem znázornil v tabulce 2.5.

Metoda	URI	Popis
GET	/manufacturers	Získání seznamu všech výrobců
GET	/manufacturers/{id}	Získání detailu výrobce
POST	/manufacturers	Přidání nového výrobce
DELETE	/manufacturers/{id}	Smazání výrobce
PUT	/manufacturers/{id}/image	Úprava obrázku pro výrobce
PUT	/manufacturers/{id}/order	Úprava čísla řazení pro výrobce
GET	/manufacturers/{id}/domains/{domain}	Získání detailu výrobce s přepsáním pro určitou doménu
POST	/manufacturers/{id}/domains	Vytvoření přepsání výrobce pro určitou doménu
PUT	/manufacturers/{id}/domains/{domain}	Úprava přepsání výrobce pro určitou doménu
DELETE	/manufacturers/{id}/domains/{domain}	Smazání přepsání výrobce pro určitou doménu

Tabulka 2.5 Návrh API výrobců

2.4 Návrh kategorií

Kategorie umožňují kategorizaci produktů, která určuje rozdělení a vlastnosti jednotlivých produktu a tím ulehčuje uživatelům hledání na e-shopu. Databázové tabulky související s kategoriemi a jejich vztahy jsou znázorněny na obrázku 2.2 a jejich základní význam je následující:

oc_category obsahuje základní informace o kategoriích.

oc_category_description obsahuje rozšířené informace o kategoriích.

oc_category_banner obsahuje data o reklamních bannerech kategorií.

oc_category_cache obsahuje předpočítané údaje o kategoriích.

oc_feature obsahuje seznam parametrů, které jsou společné pro kategorii.

oc_feature_category je vazební tabulka mezi oc_feature a oc_category.

oc_feature_default obsahuje výchozí nastavení filtru pro kategorie.

oc_feature_filter obsahuje definice filtrů, podle kterých by se měli filtrovat produkty v dané kategorii.

oc_feature_group obsahuje definice skupin filtrů.

oc_external_category obsahuje kategorie z externích zdrojů.

oc_domain_override slouží k přepisování informací pro ostatní domény.

oc_language obsahuje seznam jazyků, které lze v systému použít.

oc_url_alias slouží ke správě URL slugů.

Kategorie není jen samostatná entita, ale také součást stromové struktury. Ta je tvořena podle odkazů na rodičovské kategorie. Při práci s daty je tedy nutné kontrolovat data, aby nevznikly smyčky. Dalším omezením, které je třeba hlídat, je konzistence domén u stromu. To znamená, že kategorie nemůže mít jiné domény, než ty co má její rodičovská kategorie.

Návrh API kategorií a souvisejících částí také z velké části vychází z dříve zmíněných akademických prací Aloise Kouby a Tomáše Hojka. Oproti původnímu návrhu byl do hlavního API kategorií přidán koncový bod pro vytvoření a smazání vedlejší domény. Důvodem této změny bylo opuštění používání skupin domén, což více vysvětlím v kapitole Implementace kategorií 3.3. Tyto návrhy jsou znázorněny v tabulkách 2.6, 2.7, 2.8 a 2.9.

Metoda	URI	Popis
GET	/categories	Získání seznamu všech kategorií
GET	/categories/{id}	Získání detailu kategorie
POST	/categories	Přidání nové kategorie
DELETE	/categories/{id}	Smazání kategorie
PUT	/categories/{id}/parent-category	Upravení rodičovské kategorie
PUT	/categories/{id}/order	Úprava čísla řazení pro kategorii
GET	/categories/{id}/domains/{domain}	Získání detailu kategorie s přepsáním pro určitou doménu
POST	/categories/{id}/domains	Vytvoření přepsání kategorie pro určitou doménu
PUT	/categories/{id}/domains/{domain}	Úprava přepsání kategorie pro určitou doménu
DELETE	/categories/{id}/domains/{domain}	Smazání přepsání kategorie pro určitou doménu

Tabulka 2.6 Návrh API kategorií

2.5 Návrh frontendu

Části výrobců a kategorií ve frontendovém projektu Kangaroo byly navrhnuty v bakalářských pracích Martina Dvořáka a Jana Babáka [44, 53]. Jedná se hlavně o návrh přehledů ve formě tabulek a detailů ve formě formulářů. Tento návrh nebyl nijak zásadně pozměněn, protože v rámci této práce byl projekt Kangaroo upraven pouze tak, aby fungoval s novým API, nebo na základě konzultací a testování s uživateli.

Metoda	URI	Popis
GET	/categories/{id}/banners/domains	Získání domén kategorie, které mají nastavený banner
POST	/categories/{id}/banners/domains	Vytvoření nového banneru pro kategorii
GET	/categories/{id}/banners/domains/{domain}	Získání bannerů pro určitou kategorii a doménu
PUT	/categories/{id}/banners/domains/{domain}	Úprava bannerů pro určitou kategorii a doménu
DELETE	/categories/{id}/banners/domains/{domain}	Smazání bannerů pro určitou kategorii a doménu

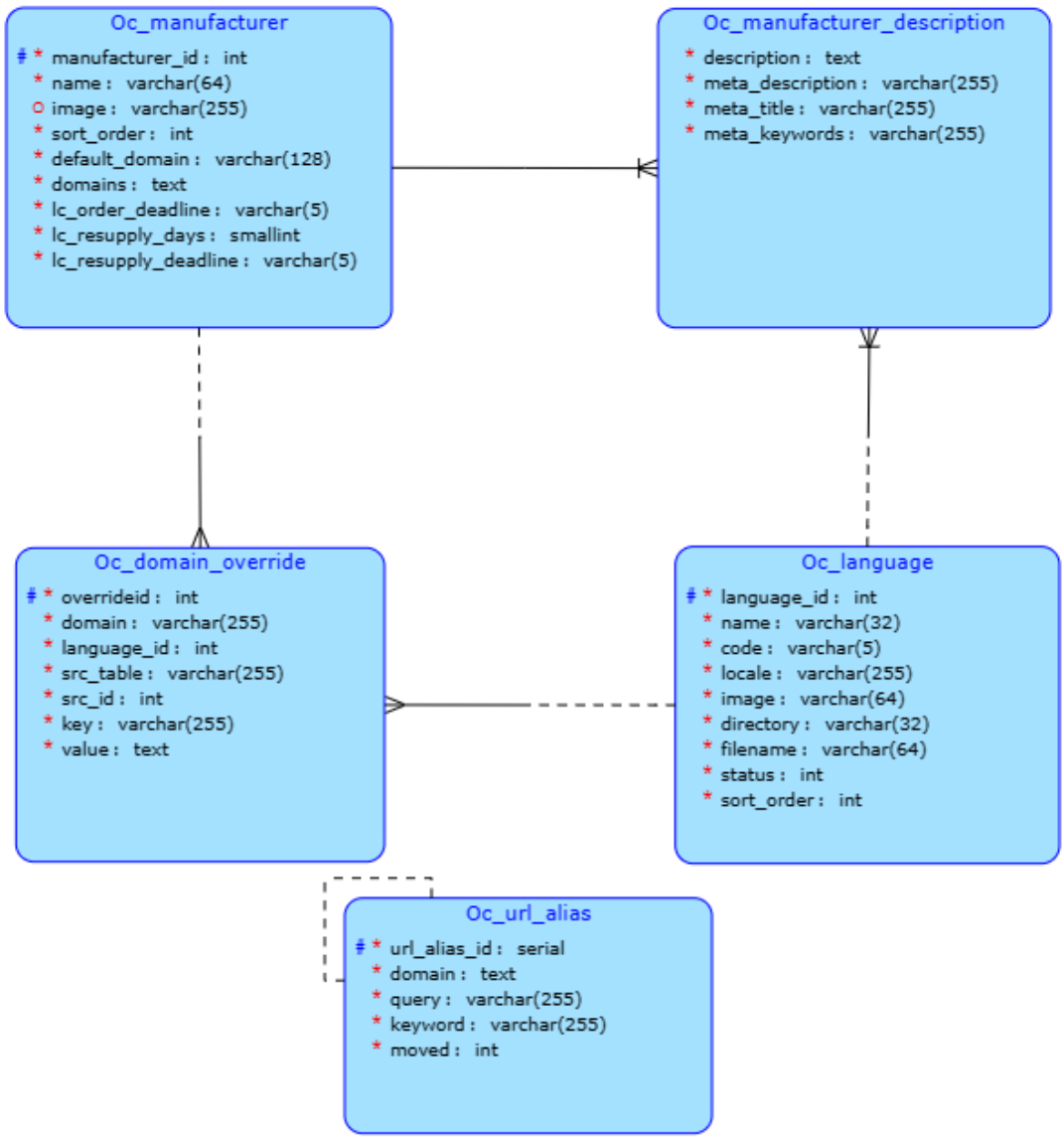
Tabulka 2.7 Návrh API bannerů

Metoda	URI	Popis
GET	/categories/parameters	Získání seznamu všech dostupných parametrů
GET	/categories/{id}/parameters	Získání seznamu všech parametrů pro zvolenou kategorii
PUT	/categories/{id}/parameters	Úprava parametrů kategorie
GET	/categories/{id}/filters	Získání seznamu všech filtrů pro zvolenou kategorii
PUT	/categories/{id}/filters	Úprava filtrů kategorie

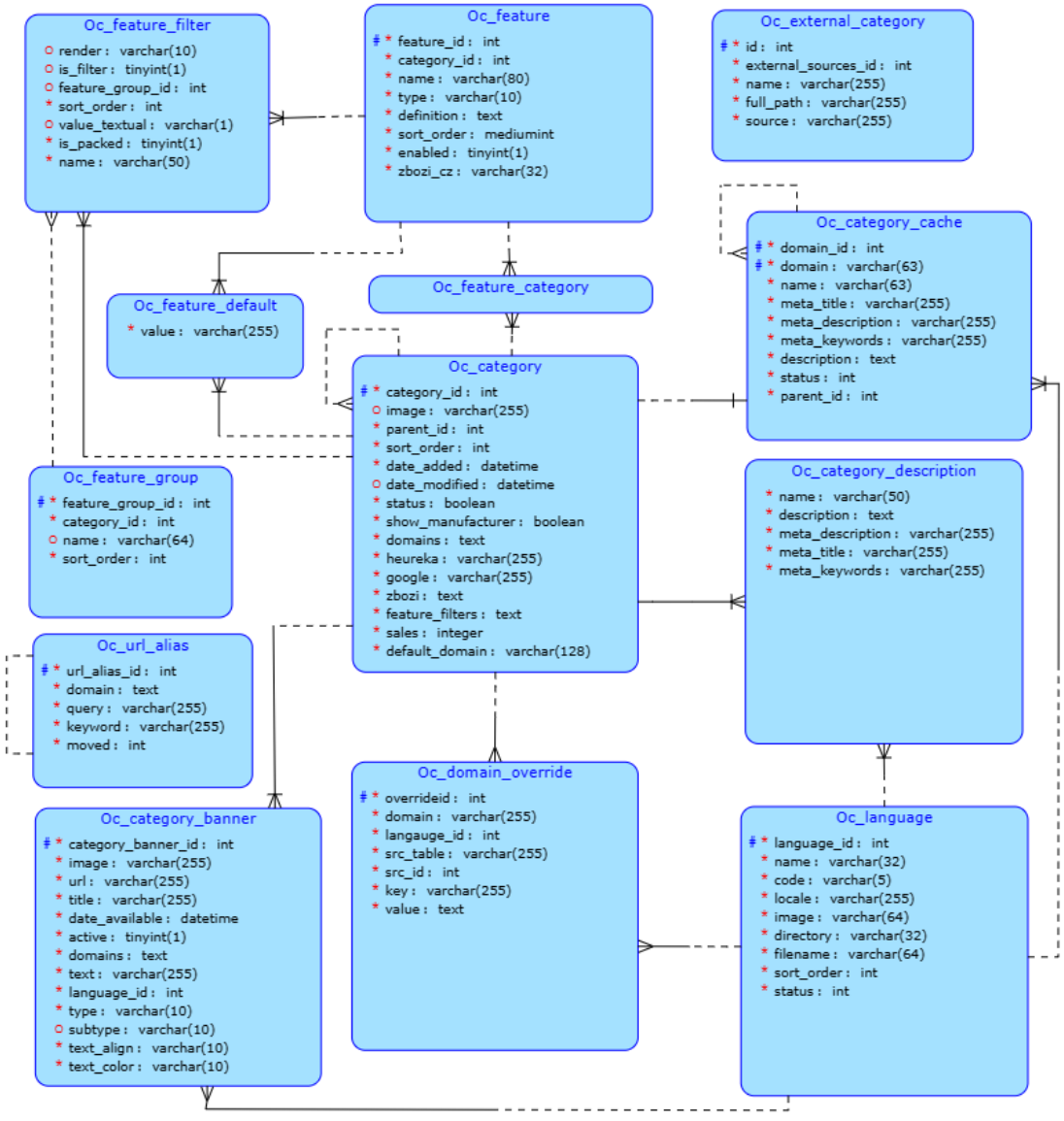
Tabulka 2.8 Návrh API parametrů a filtrů

Metoda	URI	Popis
GET	/externalcategories/{source}	Získání externích kategorií z určitého zdroje
PUT	/externalcategories	Načtení externích kategorií ze zdrojů

Tabulka 2.9 Návrh API externích kategorií



Obrázek 2.1 Výrobci – Diagram databázových tabulek



Obrázek 2.2 Kategorie – Diagram databázových tabulek

Kapitola 3
Realizace

Tato kapitola se bude věnovat základnímu nastavení vyvíjené aplikace a implementaci některých společných částí, následně se zaměří na implementaci navržených částí systému a nakonec na napojení vytvořeného API na frontendový projekt Kangaroo a změny provedené v něm. V každé části popíšu hlavně zajímavé nebo důležité detaily a rozhodnutí, které jsem při vývoji učinil.

3.1 Základní nastavení projektu Pangolin

Po vybrání frameworku ASP.NET Core a vytvoření nového repozitáře pro projekt bylo nutné zvolit balíčky, které budou použity pro vybudování aplikace a implementaci žádaných funkcí.

Knihovny funkcí jsou v rámci platformy .NET nazývány balíčky. Mohou být oficiální, interní nebo od třetích stran. Pro přidání, instalaci balíčků do projektů a správu jejich závislostí slouží nástroj NuGet. [54]

Některé balíčky byly přidávány v průběhu vývoje podle aktuálních potřeb či na doporučení zadavatele. Pro začátek projektu Pangolin byly využity tyto hlavní balíčky:

AutoMapper: umožňuje konzistentní mapování objektů na objekty napříč celou aplikací;

Microsoft.EntityFrameworkCore: poskytuje funkčnost ORM a další funkce pro práci s databázemi;

Pomelo.EntityFrameworkCore.MySql: slouží k napojení databáze typu MySQL do aplikace;

FluentValidation: umožňuje tvorbu validačních pravidel ke kontrole hodnot v objektech;

Swashbuckle.AspNetCore: provádí generování dokumentace koncových bodů API na základě anotací v kódu;

JaguExtensions.Filterable: interní balíček firmy *Jagu s.r.o.*, který umožňuje filtrování, řazení, vyhledávání a paging u koncových bodů API;

Microsoft.AspNetCore.OpenApi: používá se k nastavení cest koncových bodů API podle specifikace OpenAPI;

AspNet.Security.OAuth.Keycloak: slouží k nastavení použití nástroje Keycloak pro autorizaci a autentizaci;

Hangfire slouží k provádění výpočtů na pozadí a pro plánování jejich spuštění.

Některé funkčnosti z balíčků se nastavují v souboru Program.cs, který obsahuje veškerý kód, který je třeba provést při spuštění aplikace. Ve výpisu kódu 3.1 lze vidět nastavení balíčků Swashbuckle, díky kterému bude automaticky generována dokumentace API a bude tak splněn požadavek N6, AutoMapper, kterému je přiřazena třída MappingProfile s definicemi mapování, jejichž příklad je možné vidět ve výpisu kódu 3.2, Filterable, který má povoleno využívat balíčku AutoMapper a generovat další dokumentaci a FluentValidation. Dále lze vidět registrování FluentValidation validační třídy pro třídu ManufacturerRequest a nastavení middlewaru, který zachytává výjimky a vrací upravené zprávy s odpovídajícími návratovými kódy. Nakonec nelze opomenou nastavení připojení k MySQL databázi za pomocí připojovacího řetězce.

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSwaggerGen(options =>
{
    options.SupportNonNullableReferenceTypes();
    options.DescribeAllParametersInCamelCase();
    options.EnableAnnotations(true, true);
});
builder.Services.AddControllers();
builder.Services.AddRouting(options => { options.LowercaseUrls = true; });
builder.Services.AddAutoMapper(typeof(MappingProfile));

var connectionString = builder.Configuration.GetConnectionString("Default");
builder.Services.AddDbContext<DatabaseContext>(options =>
{
    options.UseMySql(connectionString,
        ServerVersion.AutoDetect(connectionString));
    options.UseSnakeCaseNamingConvention();
});

builder.Services.AddTransient<ExceptionHandlingMiddleware>();

builder.Services.AddFilterableExtension(options =>
{
    options.UseAutoMapper = true;
    options.GenerateSwaggerDocumentation = true;
    options.RegisterDatabaseContext<DatabaseContext>();
});

builder.Services.AddFluentValidationAutoValidation();
builder.Services.AddScoped<IValidator<ManufacturerRequest>,
    ManufacturerValidator>();
```

Výpis kódu 3.1 Ukázka kódu Program.cs

V projektu Porcupine byly vytvořeny migrace pro úpravu databáze, které například přidali do tabulek *oc_manufacturer*, *oc_category* a *oc_product* důležitý sloupec *default_domain*, který obsahuje základní doménu dané entity a řeší požadavek F26. Tyto migrace bylo tedy nutné přidat zprovoznit i pro projekt Pangolin. V frameworku ASP.NET tuto funkčnost poskytuje EF. Nové migrace byly vytvořeny za pomocí příkazu **dotnet ef migrations add NázevMigrace** a bylo nastaveno jejich spouštění v souboru Program.cs. Kód migrace obsahuje metody, které se spustí při spuštění a vypnutí aplikace. Pro zachování konzistence s produkční databází není možné

```
CreateMap<Manufacturer, ManufacturerResponse>()
    .ForMember(dst => dst.Domains, opt =>
        opt.MapFrom(src => src.GetDomainsArray()));
CreateMap<Manufacturer, ManufacturerDetailResponse>()
    .ForMember(dst => dst.DefaultDomain, opt => opt.Ignore());
CreateMap<Manufacturer, DomainManufacturerResponse>()
    .ForMember(dst => dst.ManufacturerName, opt => opt.MapFrom(src => src.Name));
```

Výpis kódu 3.2 Příklad definic pro mapování objektů

v rámci migrací mazat tabulky či sloupce.

Soubor Program.cs se v průběhu vývoje velmi rozrůstal. Jedním z důvodů je fakt, že se zde také provádí registrace tříd za účelem jejich injekce do dalších tříd. Dále zde kolega Alois Kouba nastavil autorizaci a autentizaci za použití nástroje Keycloack a také práci se souborovým systémem. Tyto nastavení kolega popsal ve své diplomové práci.

Po provedení základního nastavení byly implementované části, která jsou společné pro další části aplikace. Mezi tyto části patří URL slugy, domény a jazyky. Logika jazyků a domén je jednoduchá, skládá se ze získávání entit a kontrol. URL slugy jsou specifické svou logikou vytváření a úprav. Místo úpravy slugu se totiž vždy vytvoří slug nový a do dat starého slugu se vloží odkaz na nový. Děje se tak z důvodu přesměrování. Také je nutné, aby klíčová slova aktivních URL slugů byla unikátní a rozlišují se záznamy pro hlavní a vedlejší domény.

3.2 Implementace výrobců

Jako první jsem implementoval část výrobců. Díky použití balíčku Filterable bylo možné implementovat koncové body, které vrací seznamy entity tak, že je tyto seznamy možné filtrovat, řadit a stránkovat. To je značné vylepšení oproti projektu Porcupine, kde tato funkcionalita nebyla. K určení základního chování a sloupců, podle kterých je možné řadit a filtrovat, slouží takzvané Query třídy. Výpis kódu 3.3 ukazuje implementaci koncového body pro získání seznamu výrobců za pomocí balíčku Filterable. Dále si ve výpisu lze všimnout atributů, které slouží k vystavení koncového bodu s metodou GET a jeho dokumentaci.

Jednou z výhod použití frameworku ASP.NET a EF je možnost získávání dat z více tabulek najednou, což je pro tento projekt velice vhodné například pro tabulky *oc_manufacturer_description* a *oc_manufacturer*, kde hlavní data obsahuje první tabulka, ale často je potřeba pracovat i s daty z tabulky druhé. Podmínkou pro tuto funkci je definování vazeb mezi entitami v konfiguračním souboru a navigační vlastnosti ve třídě entity. Výpis kódu 3.4 ukazuje konfigurační třídu pro výrobce, kde jsou definovány vazby s entitami popisů a hodnotami k přepsaní.

3.3 Implementace kategorií

Během konzultací se zadavatelem bylo zjištěno, že projekt Porcupine využíval pro implementaci logiky kategorií uměle vytvořené skupiny domén, které sloužily pro kontrolu omezení kategorií. Tyto skupiny ale omezovali možnosti nastavení domén u kategorií, což bylo špatně. Bylo proto rozhodnuto těchto skupin nevyužívat. Místo skupin domén byly navrženy a implementovány dva nové koncové body a důsledné kontroly domén kategorií a dalších dat. Tyto kontroly a způsoby kontrol dat popíšu v následujícím textu.

Validace příchozích dat je pro webové aplikace velmi důležitá, ať už z pohledu bezpečnosti, ale také pro správné fungování aplikací. Při vývoji projektu Pangolin jsme pro jednoduché kontroly dat využívali atributů pro vlastnosti. Tyto atributy dokáží kontrolovat překročení délky řetězce, povolené hodnoty a další. Některé nastavené kontrolní atributy se také propisují do generované

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

Výpis kódu 3.3 Implementace koncového bodu s balíčkem Filterable

dokumentace API. Pro složitější kontroly nám posloužil balíček FluentValidation. V části Základní nastavení projektu Pangolin 3.1 jsem se zmínil o registraci validační třídy. Výpis kódu 3.5 ukazuje příklad takové validační třídy pro třídu související s tvorbou kategorií. Validace jsou v ní definovány pomocí pravidel pro jednotlivé vlastnosti objektu.

Krom implementování API navrženého v části 2.4 byl pro kategorie přidán koncový bod, který vrací seznam kategorií obohacený o celé jméno kategorie. Toto jméno je tvořeno jménem kategorie a jmény rodičovských kategorií a představuje průchod stromovou strukturou. Cílem tohoto koncového bodu bylo umožnění stránkování, filtrování a řazení pro seznam kategorií tak, aby zároveň zobrazoval celá jména kategorií. Jeho aktuální implementace spočívá v získání stránkované kolekce a její následné obohacení o celá jména. Tato implementace ale není ideální, protože neumožňuje filtrování a řazení pro přidaná celá jména. Do budoucna bude tedy potřeba najít lepší řešení. Aktuálně implementace kategorií také naráží na limity interního balíčku Filterable, který v době psaní této práce neumožňuje řazení podle vlastnosti z vnořené entity. U kategorií se jedná konkrétně o vlastnost jméno, které je nutné získávat z tabulky *oc_category_description*.

3.3.1 Externí kategorie

Externí kategorie slouží pro mapování kategorií a produktů ze spravovaných e-shopů na kategorie externích portálů. Těmito portály jsou Heureka, Zboží.cz a Google. Každý portál veřejně vystavuje svoje kategorie v souborech různých formátů. Pro splnění funkčních požadavků na tuto část je třeba každý tento soubor jinak zpracovat a data z nich uložit. Pro ukládání zpracovaných dat slouží tabulka *oc_external_category*.

Portál Zboží.cz vystavuje svoje kategorie v souboru s formátem csv, který obsahuje hodnoty

```
public class ManufacturerConfiguration : IEntityTypeConfiguration<Manufacturer>
{
    public void Configure(EntityTypeBuilder<Manufacturer> builder)
    {
        builder.HasMany(m => m.DomainOverrides)
            .WithOne()
            .HasForeignKey(o => o.SrcId)
            .HasPrincipalKey(m => m.ManufacturerId);
        builder.HasMany(m => m.ManufacturerDescriptions)
            .WithOne()
            .HasForeignKey(md => md.ManufacturerId)
            .HasPrincipalKey(m => m.ManufacturerId);
    }
}
```

Výpis kódu 3.4 Konfigurační třída pro výrobce

oddělené středníky. Pro zpracování tohoto souboru jsem tedy využil třídy pro parsování textových souborů, do které jsem jako oddělovač nastavil středník. Soubor jsem ale nejprve musel převést do kódování UTF-8, aby nedošlo k uložení poškozených nebo nečitelných dat.

Společnost Google své kategorie vystavuje obyčejném textovém souboru, ve kterém je identifikátor kategorie oddělen od názvu pomlčkou. Tento soubor jsem tedy pomocí regulárního výrazu rozdělil na řádky, které jsem následně rozdělil na identifikátor a název a ty uložil.

Soubor od portálu Heureka využívá formátu XML. Kategorie jsou v něm reprezentovány jako stromy a jejich data jsou označena pomocí odpovídajících tagů. Pro zpracování tohoto souboru jsem využil třídy ke čtení XML souborů a za pomocí XPath vyhledávacích výrazů jsem soubor rekurzivně procházel a četl data, které jsem následně ukládal.

Celý proces načítání externích kategorií je časově náročnější, většinou trvá okolo 5 minut, a to hlavně z důvodu velkého množství dat v každém souboru. To je také důvod pro funkční požadavek F22.

Pro nastavení automatické a opakující úlohy jako je spouštění načítání externích kategorií byl využit balíček Hangfire. Systém administrace obsahuje více podobných synchronizačních úloh, proto jsem připravil třídu RecurringJobWorker, do které přijdou všechna nastavení takových úloh. Ve výpisu kódu této třídy 3.6 lze vidět statickou metodu, která je volána ze souboru Program.cs při spuštění aplikace, a také nastavení úlohy pro načtení externích kategorií na spuštění jednou týdně v pondělí v 1 hodinu ráno.

3.4 Implementace frontendu

Jak jsem zmínil v části Návrh frontendu 2.5, projekt Kangaroo byl upraven pouze tak, aby fungoval s novým API, nebo na základě konzultací a testování s uživateli nebo na základě konzultací a testování s uživateli. V následujícím textu popíšu hlavní úpravy, které byly provedeny.

Struktura datových objektů, které jsou používány pro komunikaci s API, byly dříve v projektu Kangaroo ručně definovány, což vedlo k možným problémům při změnách API nebo nekonzistentnosti. Díky generované dokumentaci API z projektu Pangolin bylo možné využít nástroje openapi-typescript, který za pomocí jediného příkazu vygeneruje definice všech objektů. Následně byl vytvořen soubor api.ts, který mapuje tyto definice a slouží k jejich importování do ostatních částí kódu. Nově při úpravách API stačí znovu vygenerovat definice a v určitých případech upravit mapování v souboru api.ts.

Jak jsem se dříve zmínil, bylo rozhodnuto přestat používat uměle vytvořené skupiny domén. Tato rozhodnutí znamenalo nevystavení koncového body, který byl dříve v projektu Kanga-

roo používán k získání všech skupin domén. Bylo tedy potřeba odstranit části kódu, které se skupinami domén pracovali a zároveň zajistit správné fungování i po jejich odstranění. Nejvíce takových částí kódu se nacházelo v komponentě DomainSettings, která je společná pro kategorie a výrobce.

V části Implementace výrobců 3.2 jsem popsal možnosti, které do API přinesl balíček Filterable. Pro práci s koncovými body API, které tento balíček využívají, byl do projektu Kangaroo přidán interní balíček Jagu Rest Api Filters Client, který slouží k přípravě filtrů, a kolega Alois Kouba upravil společnou komponentu tabulky tak, aby pracovala se stránkováním.

Kolega Kouba si také pro své potřeby upravil komponentu zobrazující obrázky tak, aby dokázala zobrazovat více obrázků najednou. Jelikož u výrobců a kategorií je možné přidat pouze jeden obrázek, přidal jsem do komponenty vedle vlastnosti od kolegy také vlastnost, která slouží k předání jednoho řetězce obrázku. Podle toho, která z těchto vlastností je inicializována při vytvoření, komponenta pozná, jak se správně chovat.

Díky novému API a těmto úpravám bude budoucí vývoj frontendového projektu Kangaroo udržitelnější a pohodlnější.

```
public class DomainCategoryValidator :
    EntityValidator<DomainCategoryRequest>
{
    public DomainCategoryValidator(DatabaseContext databaseContext):
        base(databaseContext)
    {
        RuleFor(dc => dc.DomainName)
            .MustAsync(DomainExists).WithMessage("'Domain' doesn't exist");
        RuleFor(dc => dc.ParentCategoryId)
            .MustAsync(CategoryExists)
            .When(dc => dc.ParentCategoryId != 0)
            .WithMessage("Parent category with 'ParentCategoryId'
                doesn't exist");
        RuleFor(dc => dc)
            .MustAsync(async (dc, cancel) =>
                await CheckParentDomains(dc.ParentCategoryId,
                    dc.DomainName, cancel))
            .When(dc => dc.ParentCategoryId != 0)
            .WithMessage("Invalid 'DomainName' - not included
                in parent category");
        RuleFor(dc => dc.Image)
            .Must(File.Exists)
            .When(dc => dc.Image != string.Empty)
            .WithMessage("'Image' doesn't exist");
    }

    private async Task<bool> CheckParentDomains(int parentId,
        string domain,
        CancellationToken cancellationToken = default)
    {
        var parent = await DatabaseContext.Categories.FirstAsync(c =>
            c.CategoryId == parentId, cancellationToken);
        return parent.GetDomainsArray().Contains(domain);
    }
}
```

Výpis kódu 3.5 Validační třída pro kategorie

```
public static class RecurringJobWorker
{
    public static void StartRecurringJobs(this IApplicationBuilder app)
    {
        RecurringJob.AddOrUpdate<ExternalCategoryService>("ext_cat_update",
            ec => ec.UpdateAsync(CancellationToken.None),
            Cron.Weekly(DayOfWeek.Monday, 1));
    }
}
```

Výpis kódu 3.6 Třída pro opakující se úlohy

Kapitola 4
Testování

Testování je způsob ověření, že chování aplikace je v souladu s očekáváním. Také nám poskytuje informace o kvalitě aplikace, umožňuje nám budovat důvěru v produkt a pomáhá ke snižování rizik spojených s vývojem a dodáním aplikace. Testování je důležitou součástí jakéhokoliv modelu životního cyklu vývoje softwaru (SDLC). V rámci vývojového cyklu záleží na včasnosti a rozsahu testování, protože chyby nalezené v dřívějších fázích se snadněji, a tedy i levněji, opravují. Testování se různě dělí například na statické a dynamické, podle typů testů, automatické a manuální a další. V této kapitole se zaměřím na způsoby testování a kontroly kvality kódu, které byly použity při vývoji projektu Pangolin.

4.1 Code Review

„A code review is a peer review of code that helps developers ensure or improve the code quality before they merge and ship it.“ [55]

Jedná se tedy o kontrolu kódu, který je připraven ke sloučení například do hlavní větve, dalším vývojářem nebo několika vývojáři za účelem zajištění jeho větší kvality. K dalším výhodám těchto kontrol patří například sdílení zkušeností a znalostí mezi vývojáři, zajištění vývojových konvencí a zvýšení spolupráce mezi vývojáři. Na druhou stranu tyto kontroly mohou trvat netriviální množství času, zvláště pokud se jedná o velké množství změn, a také odvádějí pozornost od jiných úkolů. Vývojáři mohou tyto kontroly pojmout různě. Mohou se zaměřovat na určité věci, procházet kód po verzích, používat určité nástroje ke kontrole nebo přímo zkoušet aplikaci. [55]

Již v části Git a GitLab v sekci Vybraný podpůrný software 2.1.3.1 jsem se o nastavení kontrol kódu pro projekt Pangolin zmínil. Jejich konkrétní nastavení bylo následující:

1. Vývojář, který dokončil vývoj větve, vytvořil v GitLabu žádost o sloučení neboli merge request.
2. K žádosti přiřadil jiného vývojáře z týmu.
3. Jiný vývojář provedl kontrolu a zapsal do žádosti zpětnou vazbu.
4. Vývojář vyřešil problémy ze zpětné vazby a k žádosti přiřadil Bc. Martina Dvořáka.
5. Martin Dvořák provedl kontrolu a zapsal do žádosti zpětnou vazbu.
6. Vývojář znovu vyřešil problémy ze zpětné vazby.
7. Větev byla připravena ke sloučení a následně sloučena.

Osobně jsem při kontrolách kódu využíval nástrojů pro statickou analýzu o nichž se více zmíním v následující sekci. Procházel jsem změněné soubory a kontroloval domluvené vývojové konvence, nastavení kontrol vstupních dat, nepoužívané kousky kódu a další. Nakonec jsem aplikaci spustil a zkontroloval varovné hlášky, jejichž počet jsem se snažil co nejvíce minimalizovat.

4.2 Statická analýza kódu

Dle přednášky Ing. Jiřího Mlejnka *Kvalita kódu, DevSecOps*, ze předmětu Úvod od DevOps, je statická analýza kódu způsob zlepšení kvality kódu, který je založený na detekci chybových vzorů a který nevyžaduje spuštění aplikace. Díky této analýze je možné odhalit různé typy problémů. Mezi tyto problémy patří technický dluh, zbytečné nebo přílišné opakování kódu a problémy se spolehlivostí či zabezpečením. Statickou analýzu kódu je možné využívat v různých variantách například v podobě různých rozšíření do vývojového prostředí.

Během vývoje projektu Pangolin jsme využívali statické analýzy kódu hned několika způsoby. Při samotném vývoji nám námi využívané vývojové prostředí Rider od společnosti JetBrains analyzovalo kód pomocí základních pravidel, ale díky rozšíření ReSharper také podle pravidel platformy .NET. Seznam těchto pravidel je velice obsáhlý. Obsahuje totiž nejenom pravidla pro pojmenování a udržovatelnost, ale také pravidla pro výkon aplikace, které jsou velmi vhodná. Například pravidlo *CA1859* doporučuje v některých případech využít konkrétní datový typ namísto rozhraní nebo abstraktního typu, což vede k redukci zbytečných volání. [56] Ve frontendovém projektu Kangaroo byl pro analýzu kódu nastaven nástroj ESLint. Osobně jsem při vývoji obou projektů ke statické analýze kódu také využíval rozšíření do vývojového prostředí SonarLint, protože se mi osvědčilo při práci na jiných projektech.

Díky použití nástroje GitLab CI jsme při vývoji také využívali možnost statické analýzy kódu při zaslaní změn do vzdáleného repozitáře. Kolega Kouba do konfiguračního souboru tohoto nástroje v projektu Pangolin přidal dva oficiální přídavné konfigurační soubory a to *SAST.gitlab-ci.yml* a *Code-Quality.gitlab-ci.yml*. První zmíněný slouží k bezpečnostnímu testování, při kterém hledá známé zranitelnosti. Díky této analýze jsem objevil a vyřešil zranitelnost související se čtením XML souboru, o kterém jsem se zmínil v části Externí kategorie 3.3.1. Druhý konfigurační soubor při spuštění kontroluje kvalitu a složitost kódu a následně upozorňuje na příliš dlouhé nebo složité funkce a další vady.

4.3 Integrační testy

Krom manuálního testování API pomocí programu Postman bylo vhodné projekt Pangolin otestovat také pomocí testů automatických. Výhodou automatických testů je úspora času pro vývojáře a jejich opakovatelnost, například při spouštění pomocí CI. K testování aplikací vybudovaných na frameworků ASP.NET existuje více testovacích balíčků. Pro projekt Pangolin byl vybrán balíček xUnit, který vyniká v izolaci testů a rozšiřitelnosti. [57]

Pro otestování mnou implementovaných částí v backendovém projektu jsem zvolil integrační testy, které testují části aplikace společně. Pomocí těchto testů jsem otestoval průchod celou logikou koncových bodů, od zpracování HTTP požadavku po návratovou hodnotu. V testech jsem vždy nejdříve vytvořil HTTP klienta a pro některé koncové body i objekt do požadavku. Následně jsem daný koncový bod přes klienta zavolal a zkontroloval jsem návratovou hodnotu. Vytvářel jsem testovací případy s pozitivními, ale i s negativními výsledky, abych zkontroloval, zda fungují kontroly a validace. Nejdříve jsem použil koncové body pro vytvoření záznamu, které jsem poté upravil, následně jsem zavolal koncové body pro získání, a nakonec jsem záznamy smazal. Takto jsem otestoval API výrobců a základní API kategorií.

4.4 Uživatelské testování

Pro webovou aplikaci, jakou dohromady tvoří projekty Pangolin a Kangaroo, je velice vhodné provést manuální uživatelské testování, ať už pro ověření správné funkčnosti a návrhu, ale také pro získání námětů k vylepšení. Ke spojení s účastníky testování jsem využil nástroje Google Meet a AnyDesk.

Jako první jsem absolvoval testování se zadavatelem, tedy s Ing. Jiřím Hunkou. Jelikož to bylo moje první uživatelské testování, tak moje připravené testovací případy nebyly moc kvalitní. I přes tento fakt jsme testování provedli. Pan Hunka prošel aplikací k formuláři výrobců a samotným formulářem bez jakýchkoliv problémů. Zadávání dat do formuláře kategorií pan Hunka také zvládl bez problémů, ale při uložení se objevila chybová hláška, která byla způsobena odesláním špatného požadavku. Stalo se tak, protože pole s rodičovskou kategorií bylo vynulováno, což způsobilo, že v něm byla neplatná hodnota. Pan Hunka dále navrhl filtrování výběru rodičovských kategorií podle zvolené domény a navrhnutí hodnot pro externí kategorie na základě zadaného názvu kategorie.

Na základě zpětné vazby od Ing. Jiřího Hunky jsem pro další uživatelské testování připravil dotazník se základními otázkami o testujícím uživateli, testovacími případy vytvoření výrobce a kategorie, které byly napsány tak, aby připomínaly reálnou situaci, a zhodnocením částí po otestování.

Druhým uživatelem, který aplikaci testoval, byla paní Veronika Šmídová. Paní Šmídová s aktuální administrací pracuje již 4 roky, a to z pozice administrátora. Při tvorbě výrobce a kategorie správně identifikovala tlačítko pro přidání nové domény a úkoly zvládla bez větších problémů. Nevýhodou jejího řešení bylo nevyužití zkopírování dat z hlavní domény. Paní Šmídová považuje testované části za jednoduché k ovládání a intuitivní. Nová aplikace ji přijde lepší než aktuální systém.

Třetím účastníkem testování byl pan Libor Kudrna, který má s aktuálním systémem desetiletou zkušenost. Při výběru obrázku pro výrobce označil více obrázku a musel je následně odoznačit. Panu Kundrnovi, narozdíl od paní Šmídové, tlačítko pro přidání další domény nepřišlo tak intuitivní a nalezl ho až s lehkou pomocí. Vytvoření kategorie již proběhlo bez větších potíží. Aplikace se mu líbí a má podle něj lépe řešené vytváření kategorie a výrobce než aktuální systém. V aplikaci mu chybí možnost překládat vložené texty do jiných jazyků a dialog pro zabránění ztráty dat při odchodu z formuláře. Dále by se mu líbila možnost zobrazení více než dvou domén najednou.

Testoval jsem také s uživatelem, který neměl s původním systémem administrace e-shopů žádné zkušenosti. Tester se v aplikaci orientoval a pohyboval přirozeně, jednoduše nalezl cestu do seznamu výrobců a kategorií a následně i do vytvářecích formulářů. Jako první chtěl vložit obrázek přetažením a také chvíli hledal tlačítko pro přidání domény. Testované části systému se mu líbily. Aktuální systém nezná, ale po shlédnutí poskytnutých obrázků tohoto systému preferuje nový. Při testování nenastali žádné chyby.

Pro příští vývoj bych doporučil uživatelské testovaní provozovat průběžně během vývoje, protože názor a pozorování uživatele jsou velmi cenné. Zvláště pokud jde o uživatele, který by měl v budoucnu s aplikací pracovat skoro každý den.

Kapitola 5
Zhodnocení

V této kapitole zhodnotím svůj přístup k vývoji, jeho výsledky a následně i nastavení týmové spolupráce a můj manažerský vliv. Navrhnu také možné budoucí vylepšení.

Svůj přístup k vývoji hodnotím relativně kladně. Naimplementoval jsem dvě části systému, které splňují zadané požadavky a jsou jako celky funkční, a následně jsem je i základně otestoval. Kvůli obavám z vývoje frontendu jsem se jeho vývoji dlouho vyhýbal. Pro efektivnější vývoj je ale lepší implementovat backend i frontend zároveň, protože bych mohl lépe a včas usměrňovat vývoj obou částí dle potřeby. Během vývoje části výrobců jsem navrhnul nové API k této části, které jsem následně i naimplementoval. Toto API pracovalo se všemi doménami výrobce naráz a lépe tak využívalo efektivitu a rychlost frameworku ASP.NET. Kvůli již zmíněním obavám s vývojem frontendu, nedostatku času a nejisté korektnosti, jsem se ale nakonec rozhodl využít původního návrhu z navázaných prací a API jsem předělal. Pro budoucí vývoj projektu navrhuji API výrobců znovu předělat, tak aby pracovalo se všemi doménami naráz, a zapracovat do projektu Kangaroo. Pokud by se toto API osvědčilo, je dle mého názoru vhodné tuto úpravu provést i u kategorií.

Při vývoji bylo využito starších výpisů z produkčních databází. Aktuální produkční databáze obsahuje změny, které je třeba v budoucnu zapracovat i do projektu Pangolin. Jedná se například o přidání identifikátorů externích kategorií k entitám namísto řetězců.

Nastavení týmové spolupráce bylo podle mého názorů velmi dobré. Agilní metodika nám umožnila zabývat se ve sprintech vždy aktuálně potřebnými věcmi. S kolegy jsme se shodli, že pravidelné týdenní schůzky byly pro nás ideální, protože jsme se díky nim navzájem informovali o naší aktuální práci a mohli jsme podle toho upravit naše plány. Schůzky nám také dodávali motivaci k práci. Nástroje, které jsme využívali k týmové komunikaci a spolupráci, byly pro nás velmi prospěšné a nijak nám ve spolupráci nebránily.

Můj manažerský vliv na tým také hodnotím pozitivně. Vedl jsem schůzky ke sprintům, organizoval další vedlejší schůzky, žádal o přístupy a spravoval úkoly. Na důkladnou práci s úkoly jsem mohl klást větší důraz, ale na druhou stranu jsem osobně také málo využíval některých funkcí spojených s úkoly jako například poznámek u úkolů.

Kapitola 6
Závěr

Jedním z hlavních cílů této práce bylo navázat na předešlé práce věnující se administraci e-shopů a dle nich navrhnout a implementovat funkční backend části nové aplikace za použití moderního programovacího jazyku a následně je napojit na již existují frontend. Dalším hlavním cílem bylo nastavení spolupráce vývojářského týmu o čtyřech lidech.

Tyto cíle byly z velké části splněny. Jako první byla provedena analýza současného stavu administrace e-shopu a prvního backendového projektu Porcupine, který byl vyvíjen jako možná náhrada aktuálního řešení, možného budoucího stavu a technologií. Poté proběhla analýza týmového vývoje a s tím souvisejících aspektů. Na základě těchto analýz vzniklo nastavení nového backendového projektu Pangolin, související týmové spolupráce a rozdělení částí systému administrace k implementaci.

Z akademických prací Ing. Tomáše Hojka a Bc. Aloise Kouby, na které tato práce navazuje, byly vybrány funkční a nefunkční požadavky, které se týkaly projektu Porcupine. Následně byla zvalidována jejich relevantnost pro nový projekt. Některé požadavky byly upraveny nebo vznikly nově. Dle těchto požadavků byl vytvořen návrh přidělených částí, který taktéž vychází ze zmíněných prací.

Na základě vytvořených návrhů byly následně jednotlivé části administrace e-shopů implementovány. Jako první bylo ale provedeno základní zprovoznění projektu, dle vybrané architektury, a výběr balíčků s funkcemi. Následně byla implementována část správy výrobců společně s několika společnými částmi jako byly URL slugy a získávání domén a jazyků. Poté byla implementována část kategorií a všechny s ní související podčásti, jako jsou například bannery, parametry a filtry. Z obou částí bylo vystaveno API, které bylo zdokumentováno a následně zakomponováno do frontendového projektu Kangaroo.

Další kapitola se zaměřila na testování v různých podobách. Řeší testování v podobně manuální kontroly kódu dalším vývojářem, automatickou statickou analýzu kódu za pomocí nástrojů, automatické testování pomocí jednotkových a integračních testů a uživatelské testování.

Předposlední kapitola obsahuje zhodnocení průběhu implementace a výsledného řešení, ale také zhodnocení mého manažerského vlivu na projekt a nastavení týmové spolupráce.

Výstupem této práce je základ systému administrace e-shopů, který dokáže spravovat údaje pro několik domén najednou. Aplikace je napsána v programovacím jazyk C# s frameworkem ASP.NET Core a splnila všechny funkční a nefunkční požadavky. Tento systém kopíruje funkčnost aktuálního systému Opencart, ale zároveň ji rozšiřuje a dovoluje i širší možnosti tohoto rozšiřování. Do budoucnosti je pro splnění záměru souběhu s aktuálním systémem a následného jeho nahrazení nutné naimplementovat i další funkcionality systému a následně i zařídit nasazení aplikace do testovacího či produkčního prostředí.

Bibliografie

1. OPENCART. *OpenCart - Open Source Shopping Cart Solution* [online]. 2024. [cit. 2024-03-24]. Dostupné z: https://www.opencart.com.
2. CLICKPOST. *Everything You Need to Know About Opencart* [online]. 2024-02. [cit. 2024-03-24]. Dostupné z: https://www.clickpost.ai/blog/opencart.
3. KOUBA, Alois. *Backend administrace e-shopu* [online]. 2022. [cit. 2024-03-23]. Dostupné z: https://dspace.cvut.cz/handle/10467/102088. Bakalářská práce. České vysoké učení technické v Praze.
4. HOJEK, Tomáš. *Backend a CI administrace e-shopu* [online]. 2023. [cit. 2024-03-23]. Dostupné z: https://dspace.cvut.cz/handle/10467/107230. Diplomová práce. České vysoké učení technické v Praze.
5. GOLMGREN, Nikita. *Backend e-shopu - nákupní košík a zpracování objednávek* [online]. 2023. [cit. 2024-04-05]. Dostupné z: https://dspace.cvut.cz/handle/10467/109631. Bakalářská práce. České vysoké učení technické v Praze.
6. CEMI. *Tajemství týmové práce: Proč je důležité, aby tým táhl* [online]. 2022-02. [cit. 2024-03-29]. Dostupné z: https://www.cemi.cz/blog/tajemstvi-tymove-prace.
7. HM PARTNERS. *Týmová práce – Teoretická část* [online]. [cit. 2024-03-29]. Dostupné z: https://bestpractices.cz/seznam-praktik/tymova-prace/teoreticka-cast/.
8. AMAZON. *What is SDLC? - Software Development Lifecycle Explained - AWS* [online]. [cit. 2024-04-05]. Dostupné z: https://aws.amazon.com/what-is/sdlc/.
9. TUTORIALS POINT. *SDLC - Overview* [online]. [cit. 2024-04-05]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm.
10. HOADLEY, Paul A. *Vodopádový model* [online]. 2008-12. [cit. 2024-04-05]. Dostupné z: https://commons.wikimedia.org/wiki/File:Vodopadovy_model.png.
11. DEE. *Spirálový přístup k vývoji sofware - analýza, hodnocení, vývoj a plánování* [online]. 2010-08. [cit. 2024-04-05]. Dostupné z: https://commons.wikimedia.org/wiki/File:Software_Development_Spiral_cz.svg.
12. KOLMISTR, Ondřej. *Metodiky agilního řízení aplikované pro vývoj programového vybavení* [online]. 2011. [cit. 2024-04-15]. Dostupné z: https://dspace.tul.cz/handle/15240/10847. Diplomová práce. Technická Univerzita v Liberci.
13. BECK, Kent; ET AL. *Manifest Agilního vývoje software* [online]. 2001. [cit. 2024-04-06]. Dostupné z: https://agilemanifesto.org/iso/cs/manifesto.html.
14. MALEC, Oldřich. *Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS* [online]. 2017. [cit. 2024-03-25]. Dostupné z: https://dspace.cvut.cz/handle/10467/69399. Bakalářská práce. České vysoké učení technické v Praze.

15. CHACON, Scott. *Pro Git* [online]. CZ.NIC, 2009 [cit. 2024-03-30]. ISBN 978-80-904248-1-4.
16. GIT. *About - Git* [online]. [cit. 2024-03-29]. Dostupné z: https://git-scm.com/about.
17. GITLAB. *About | GitLab* [online]. 2024. [cit. 2024-03-29]. Dostupné z: https://about.gitlab.com/.
18. GITLAB. *GitLab Documentation* [online]. 2024. [cit. 2024-04-03]. Dostupné z: https://docs.gitlab.com/.
19. CZ.NIC. *Jak na Internet - Komunikace přes Internet* [online]. 2012-11. [cit. 2024-03-31]. Dostupné z: https://www.jaknainternet.cz/page/1236/komunikace-pres-internet/.
20. SLACK. *What is Slack? | Slack* [online]. [cit. 2024-04-03]. Dostupné z: https://slack.com/help/articles/115004071768-What-is-Slack-.
21. GOOGLE. *Online videokonference se službami Google Meet a Duo* [online]. [cit. 2024-04-03]. Dostupné z: https://workspace.google.com/resources/video-conferencing/.
22. BERTRAM, Dane. *The Social Nature of Issue Tracking in Software Engineering* [online]. 2010. [cit. 2024-03-25]. Dostupné z DOI: 10.11575/PRISM/3368. Diplomová práce. University of Calgary.
23. LANG, Jean-Philippe. *Redmine* [online]. 2023-03. [cit. 2024-03-30]. Dostupné z: https://www.redmine.org.
24. ATLASSIAN. *Welcome to Jira Software* [online]. [cit. 2024-04-04]. Dostupné z: https://www.atlassian.com/software/jira/guides/getting-started/introduction.
25. GARLAN, David; PERRY, Dewayne E. Introduction to the Special Issue on Software Architecture. *IEEE Transactions on Software Engineering* [online]. 1995, roč. 21, č. 04, s. 269–274 [cit. 2024-03-24]. ISSN 1939-3520. Dostupné z DOI: 10.1109/TSE.1995.10003.
26. MICROSOFT. *Layered Application* [online]. 2015-03. [cit. 2024-03-24]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff650258(v=pandp.10).
27. IBM. *What Is Three-Tier Architecture?* [Online]. [cit. 2024-03-24]. Dostupné z: https://www.ibm.com/topics/three-tier-architecture.
28. MOZILLA. *MVC - MDN Web Docs Glossary: Definitions of Web-related terms* [online]. 2023-12. [cit. 2024-03-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC.
29. HARTINGER, David. *MVC architektura* [online]. [cit. 2024-03-25]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.
30. OPENCART TUTORIALS. *MVCL pattern, code flow and request & response in OpenCart defined* [online]. 2018-12. [cit. 2024-03-24]. Dostupné z: https://www.youtube.com/watch?v=X6bsMmReT-4.
31. OPENCART. *MVC-L* [online]. 2016. [cit. 2024-03-24]. Dostupné z: https://docs.opencart.com/image/mvcl.png.
32. KOĎOUSKOVÁ, Barbora. *Co je to API a jaké jsou možnosti jeho využití?* [Online]. 2024-04. [cit. 2024-04-04]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-api/.
33. INDEED. *What Is SOAP API? (Plus Comparison to REST API and Benefits) | Indeed.com* [online]. 2023-02. [cit. 2024-04-06]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-soap-api.
34. IBM. *What is a REST API? | IBM* [online]. [cit. 2024-04-06]. Dostupné z: https://www.ibm.com/topics/rest-apis.
35. AMAZON. *Front End vs Back End - Difference Between Application Development - AWS* [online]. [cit. 2024-04-06]. Dostupné z: https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/.

36. ABBA, Ihechikara Vincent. *What is an ORM – The Meaning of Object Relational Mapping Database Tools* [online]. 2023-10. [cit. 2024-04-06]. Dostupné z: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/.
37. SYMFONY. *Symfony, High Performance PHP Framework for Web Development* [online]. [cit. 2024-04-06]. Dostupné z: https://symfony.com/what-is-symfony.
38. SYMFONY. *Chapter 2 - Exploring Symfony's Code (symfony 1.4 legacy version)* [online]. [cit. 2024-04-06]. Dostupné z: https://symfony.com/legacy/doc/gentle-introduction/1_4/en/02-Exploring-Symfony-s-Code.
39. DOCTRINE. *Doctrine: PHP Open Source Project* [online]. [cit. 2024-04-06]. Dostupné z: https://www.doctrine-project.org/.
40. BERGMANN, Sebastian. *PHPUnit: The PHP Testing Framework* [online]. [cit. 2024-04-06]. Dostupné z: https://phpunit.de/index.html.
41. MICROSOFT. *Úvod do technologie .NET - .NET | Microsoft Learn* [online]. 2024-01. [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/core/introduction.
42. MICROSOFT. *What is ASP.NET Core? | .NET* [online]. [cit. 2024-04-06]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.
43. MICROSOFT. *Centrum dokumentace pro Entity Framework | Microsoft Learn* [online]. [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/cs-cz/ef/.
44. DVOŘÁK, Martin. *Frontend administrace e-shopu* [online]. 2022. [cit. 2024-04-06]. Dostupné z: https://dspace.cvut.cz/handle/10467/102105. Bakalářská práce. České vysoké učení technické v Praze.
45. VUE.JS. *Introduction | Vue.js* [online]. [cit. 2024-04-06]. Dostupné z: https://vuejs.org/guide/introduction.html.
46. JOHNS, Robert. *10 Best JavaScript Frameworks in 2024 [Updated]* [online]. 2024-05. [cit. 2024-05-05]. Dostupné z: https://hackr.io/blog/best-javascript-frameworks#toc-4-vue-js.
47. PATEL, Jeel. *10 Best Front end Frameworks for Web Development in 2024* [online]. 2024-02. [cit. 2024-04-06]. Dostupné z: https://www.monocubed.com/blog/best-front-end-frameworks/.
48. MICROSOFT. *Blazor | Build client web apps with C# | .NET* [online]. [cit. 2024-04-06]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor.
49. MICROSOFT. *ASP.NET Core Blazor | Microsoft Learn* [online]. 2024-02. [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-8.0.
50. KONOPLYANIKOVA, Natalia. *Co je to SWOT analýza a proč ji podniky potřebují? | Laba Czech* [online]. 2023-08. [cit. 2024-04-14]. Dostupné z: https://l-a-b-a.cz/blog/46-co-je-to-swot-analyza-a-proc-ji-podniky-potrebuji.
51. GENERALI. *Co je SWOT analýza a jak ji vypracovat* [online]. 2022-08. [cit. 2024-04-14]. Dostupné z: https://www.generaliceskaprofi.cz/ze-zivota/co-je-swot-analyza-a-jak-ji-vypracovat.
52. PAVLIK, Vlado; PARUCH, Zach. *What Is a Slug? URL Slugs and Why They Matter for SEO* [online]. 2022-12. [cit. 2024-04-20]. Dostupné z: https://www.semrush.com/blog/what-is-a-url-slug/.
53. BABÁK, Jan. *Frontend administrace e-shopu* [online]. 2022. [cit. 2024-05-08]. Dostupné z: https://dspace.cvut.cz/handle/10467/102225. Bakalářská práce. České vysoké učení technické v Praze.

54. MICROSOFT. *Co je NuGet a co to dělá? | Microsoft Learn* [online]. 2023-10. [cit. 2024-04-10]. Dostupné z: https://learn.microsoft.com/cs-cz/nuget/what-is-nuget.

55. GITLAB. *What is a code review? | GitLab* [online]. [cit. 2024-04-22]. Dostupné z: https://about.gitlab.com/topics/version-control/what-is-code-review/.

56. MICROSOFT. *CA1859: Pro zvýšení výkonu používejte konkrétní typy, pokud je to možné - .NET | Microsoft Learn* [online]. 2023-09. [cit. 2024-04-24]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/fundamentals/code-analysis/quality-rules/ca1859.

57. SHETH, Himanshu. *NUnit vs. XUnit vs. MSTest: Unit Testing Frameworks | LambdaTest* [online]. 2021-03. [cit. 2024-05-12]. Dostupné z: https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/.

Obsah příloh